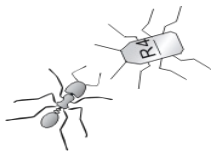
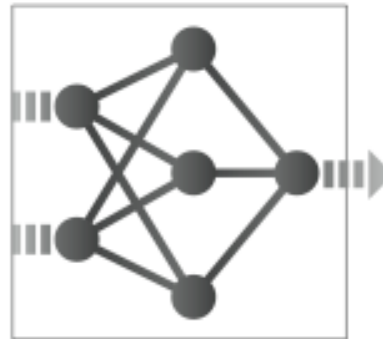


Neural Systems

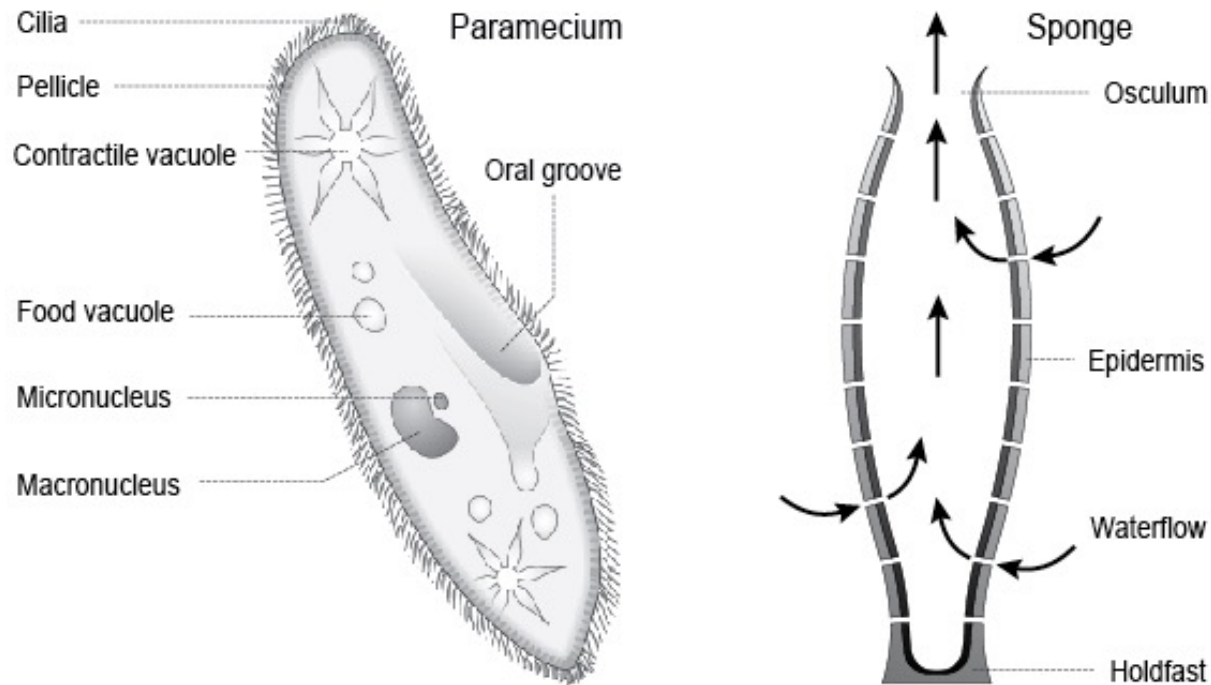
Unsupervised and Supervised Learning



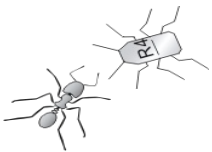
What you will learn today

- Elements of biological nervous systems
- Artificial neuron models
- Neural architectures
- Input encodings
- Unsupervised learning
 - Feature extraction and representations
 - Topological Maps
- Supervised learning
 - From error correction to backpropagation

Do animals need nervous systems?

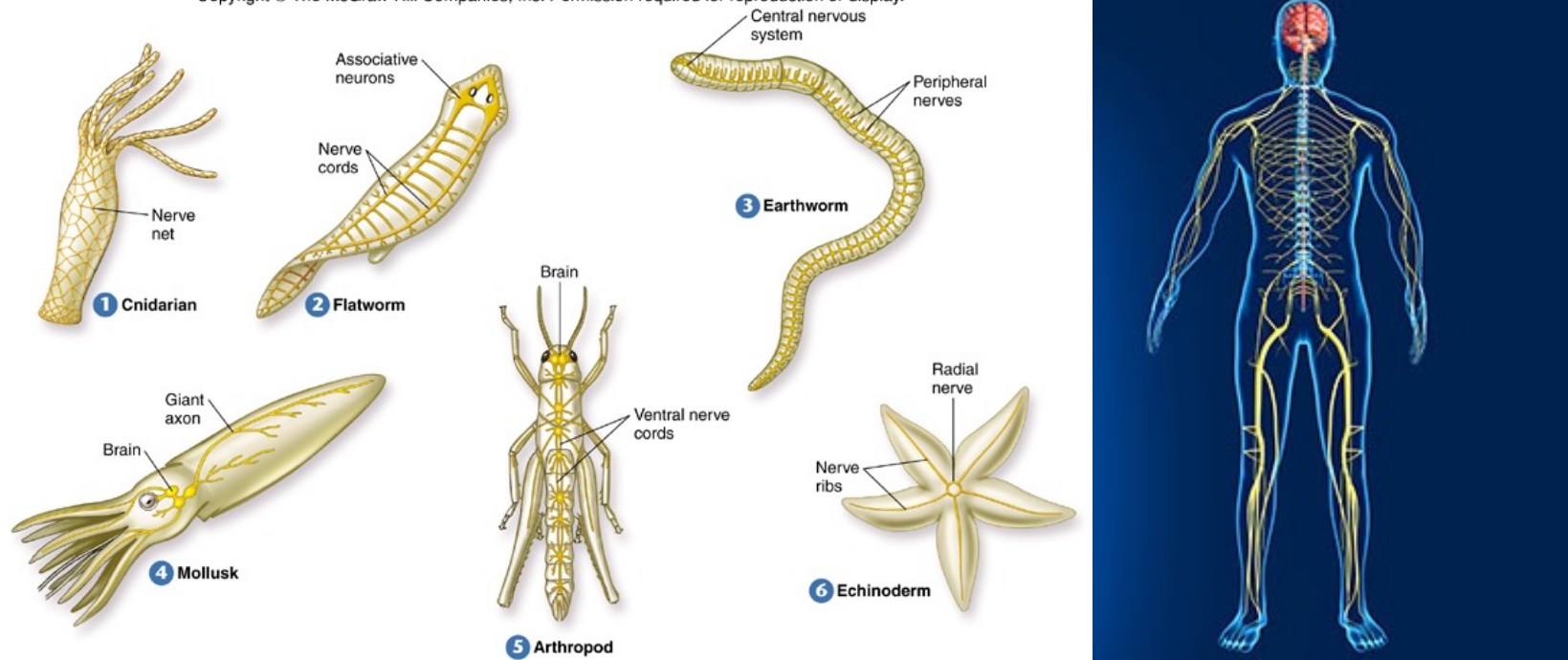


Not all animals have nervous systems; some use only chemical reactions
Paramecium and sponge move, eat, escape, display habituation

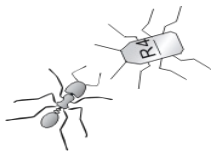


Why Nervous Systems?

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

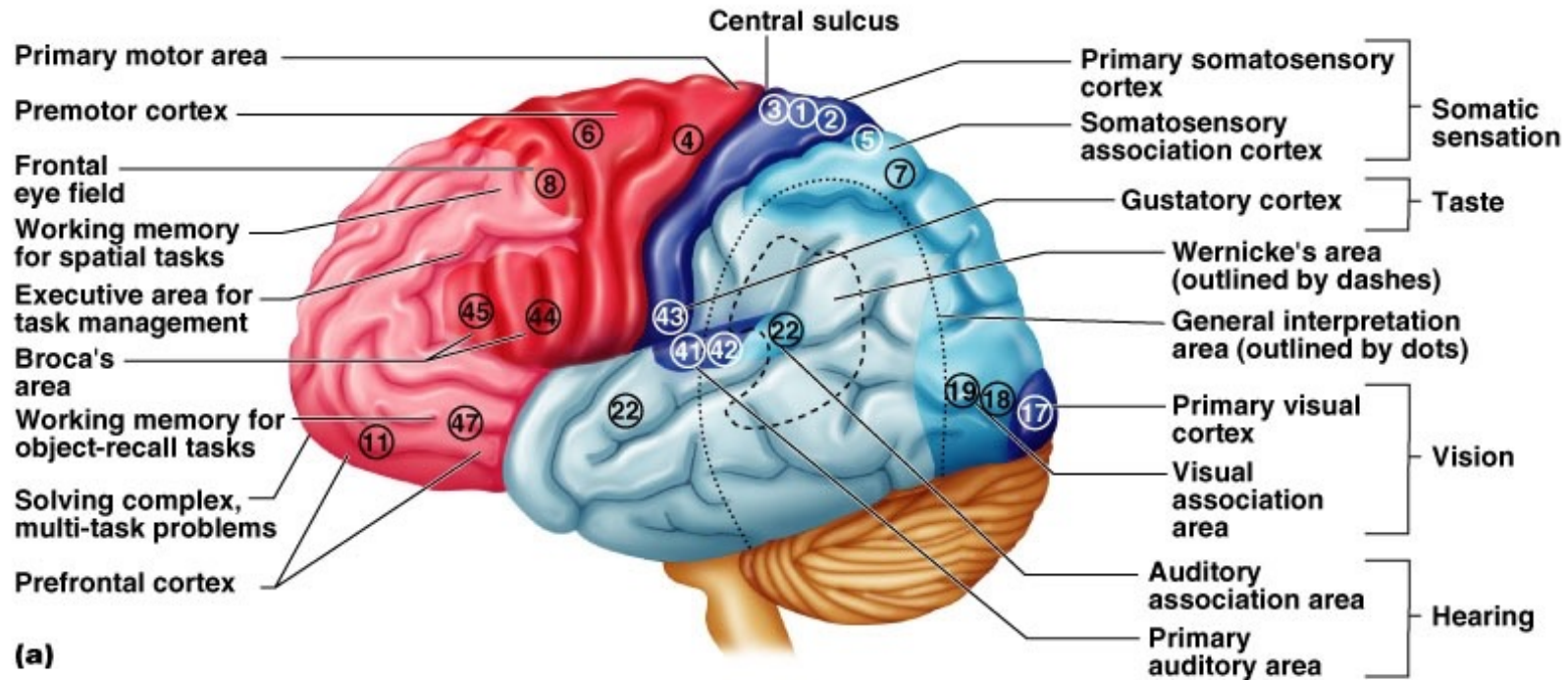


- 1) Faster reaction times = competitive advantage
- 2) Selective transmission of signals across distant areas = more complex bodies
- 3) Generation of non-reactive behaviors
- 4) Complex adaptation = survival in changing environments

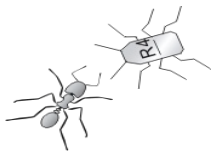


Companion slides for the book *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies* by Dario Floreano and Claudio Mattiussi, MIT Press

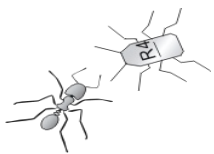
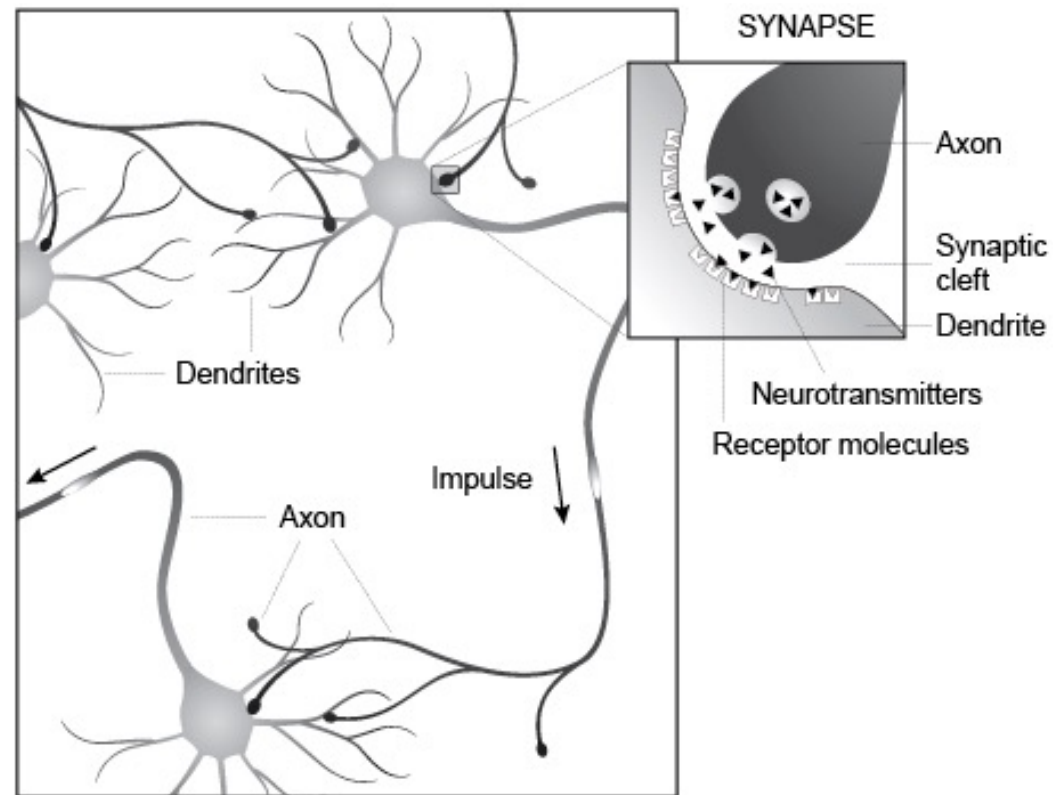
Central Nervous System with Cortex



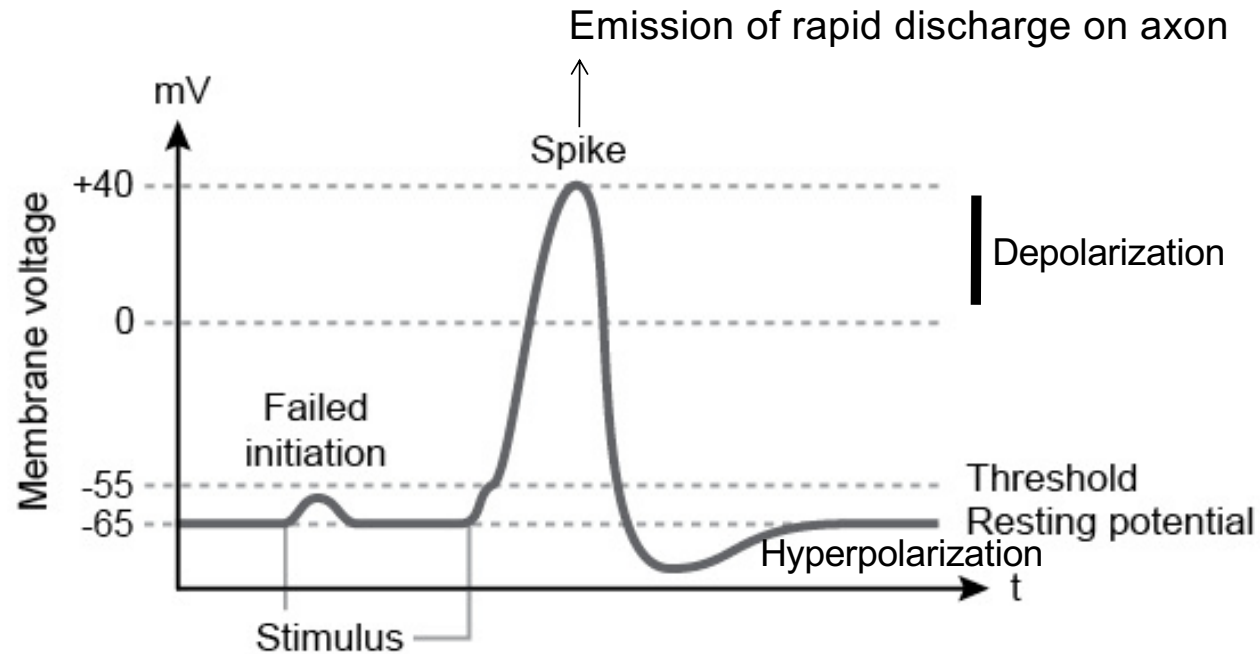
Copyright © 2004 Pearson Education, Inc., publishing as Benjamin Cummings.



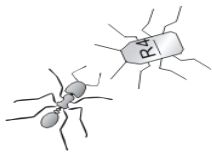
Biological Neurons



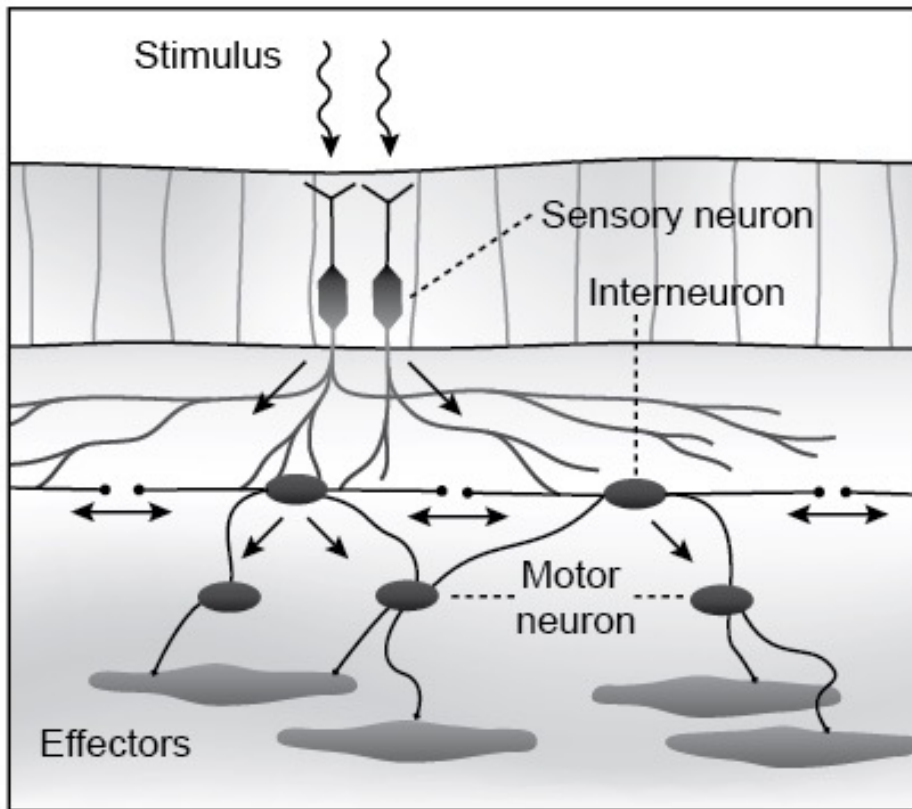
Membrane Dynamics



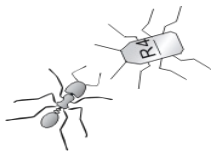
This cycle lasts approximately 3-50 ms, depending on type of ion channels involved (Hodgkin and Huxley, 1952)



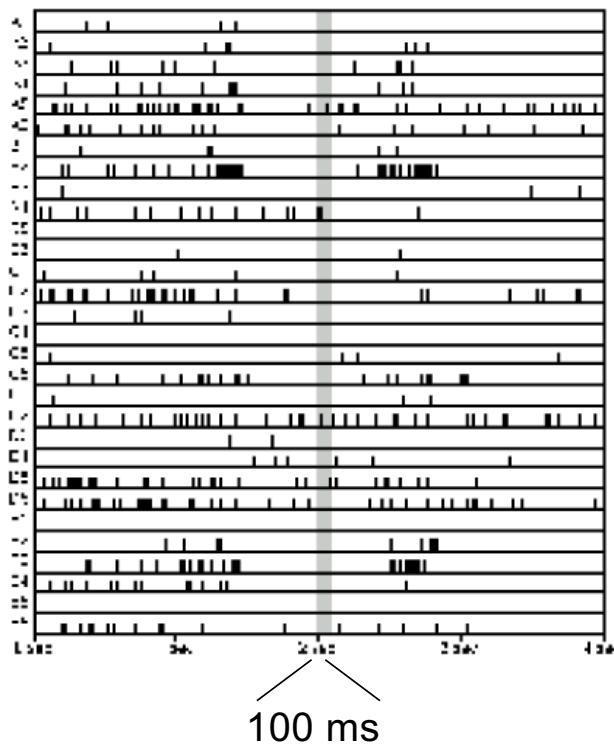
Types of Neurons



Interneurons can be
1- Excitatory
2- Inhibitory



How Do Neurons Communicate?



Firing rate

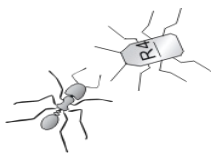
Firing time

↓
McCulloch-Pitts

↓
Spiking neurons

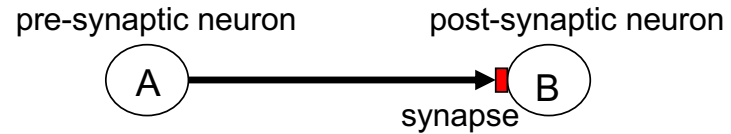
↓
Connectionism

↓
Computational
Biology



How Do Neurons Learn?

They learn by means of synaptic change



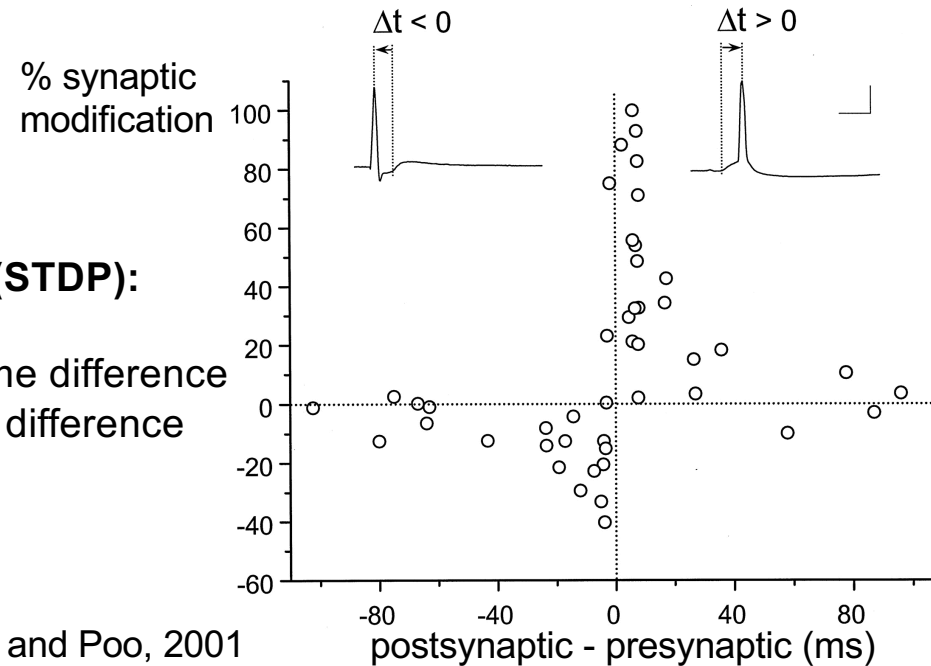
Hebb rule (1949):

Synaptic strength is increased if cell A consistently contributes to firing of cell B

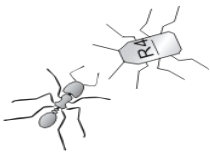
This implies a temporal relation: neuron A fires first, neuron B fires second

Spike Time Dependent Plasticity (STDP):

- Small time window
- Strengthening (LTP) for positive time difference
- Weakening (LTD) for negative time difference

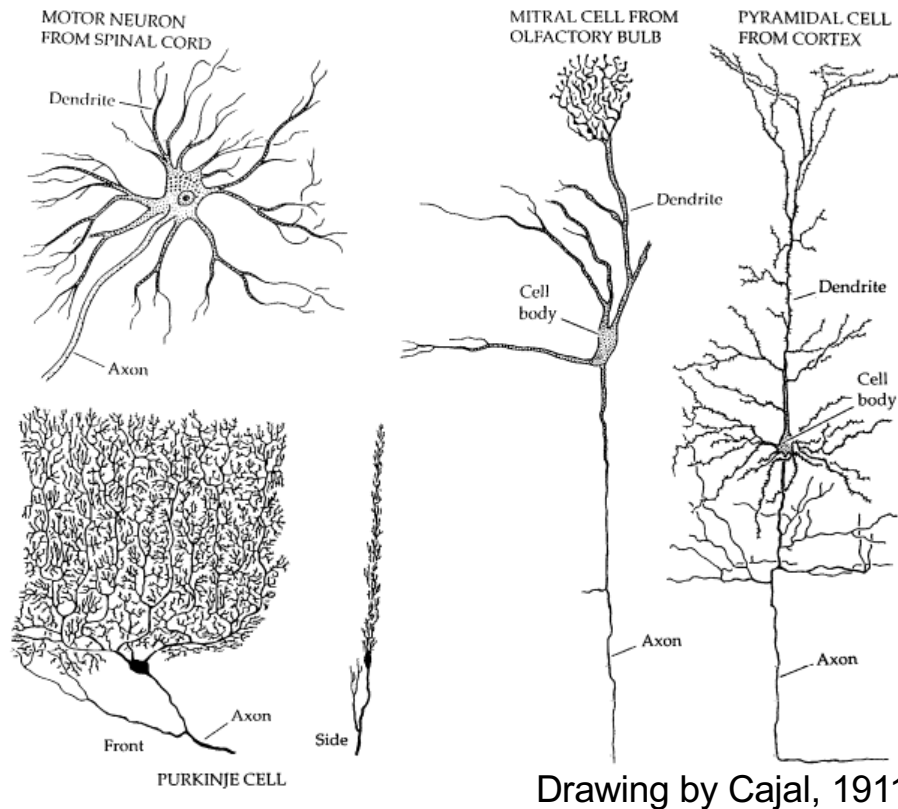


From Bi and Poo, 2001



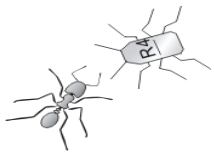
What Does Make Brains Different?

Components and behavior of individual neurons are very similar across animal species and, presumably, over evolutionary history (Parker, 1919)

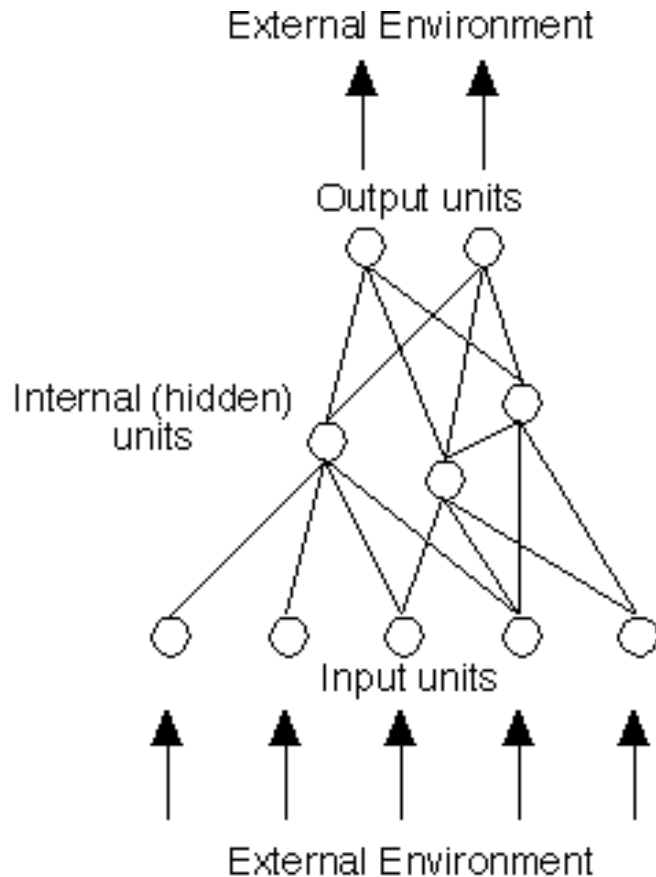


Evolution of the brain seems to occur mainly in the **architecture**, that is how neurons are interconnected.

First classification of neurons by Cajal in 1911 was made according to their connectivity patterns



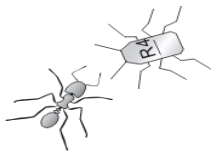
An Artificial Neural Network



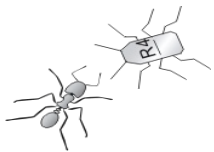
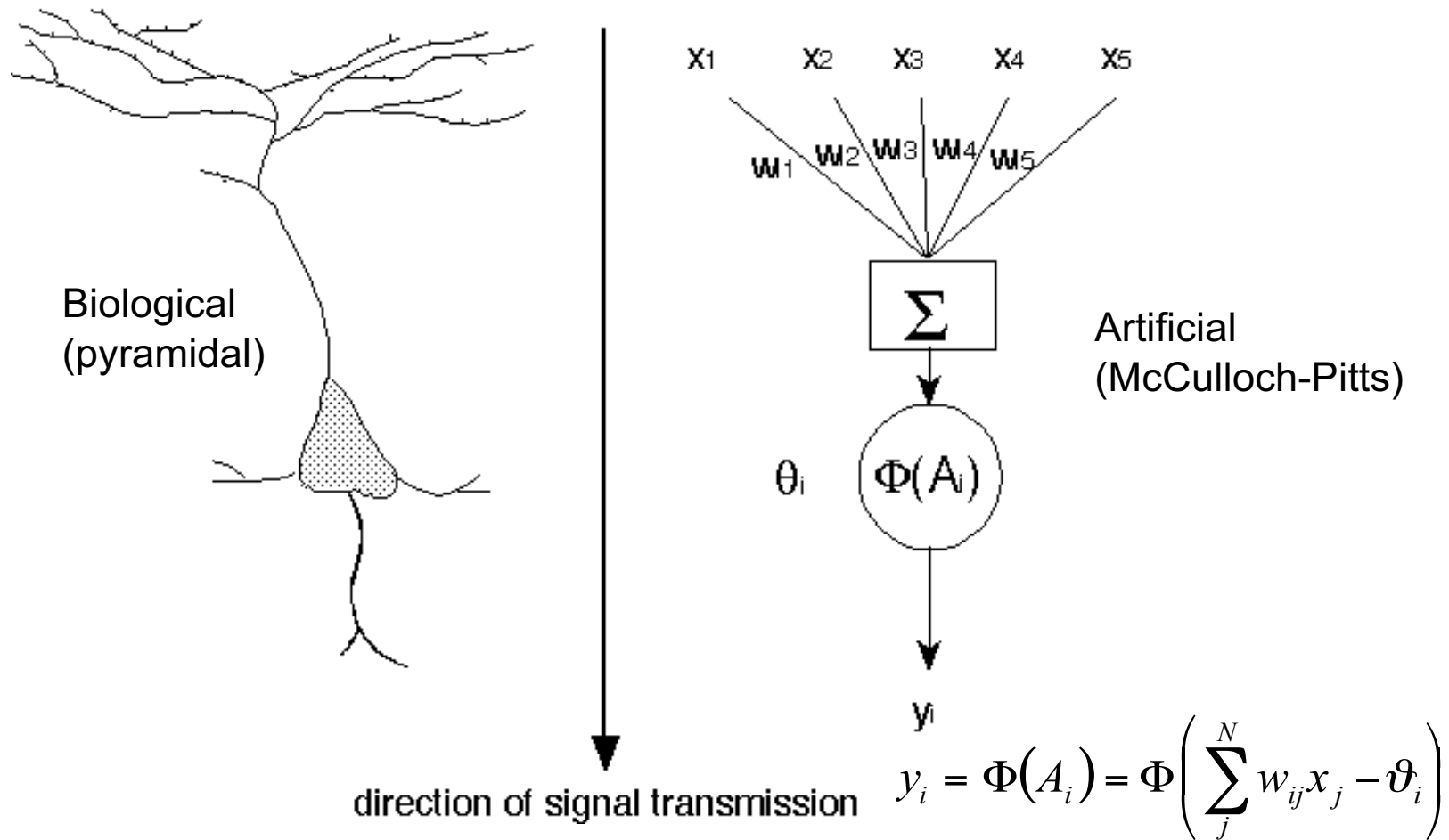
A neural network communicates with the environments through input units and output units. All other elements are called internal or hidden units.

Units are linked by uni-directional connections.

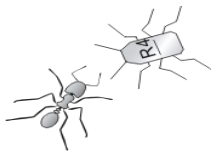
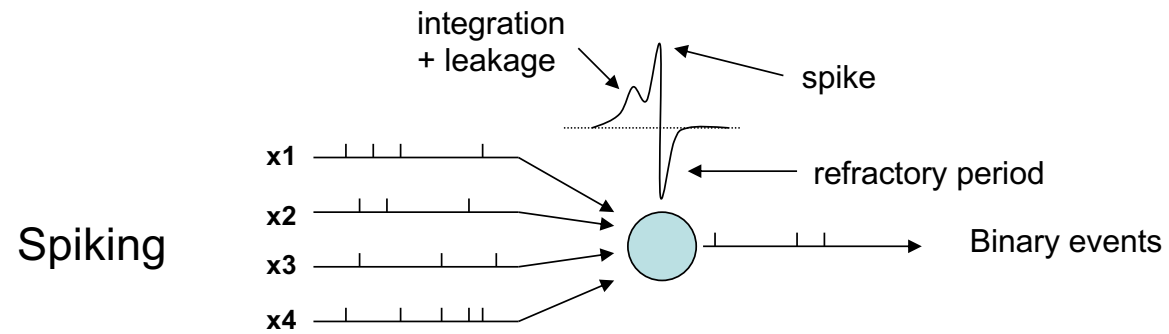
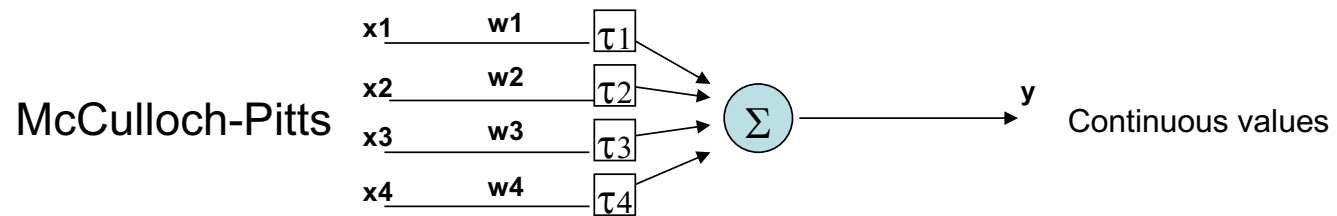
A connection is characterized by a weight and a sign that transforms the signal.



Biological and Artificial Neurons



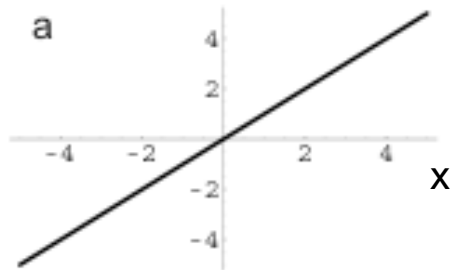
Neuron models



Some output functions

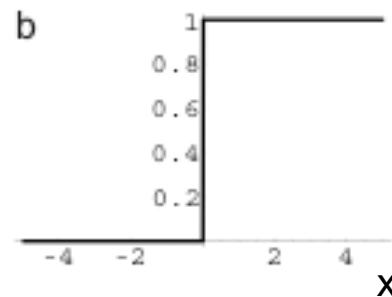
Linear

$\Phi(x)$



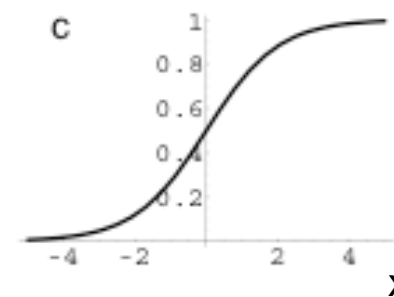
Step

$\Phi(x)$



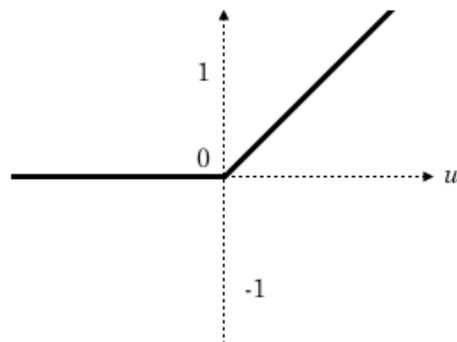
Sigmoid

$\Phi(x)$



Rectified Linear

$$f(u) = \max(0, u)$$

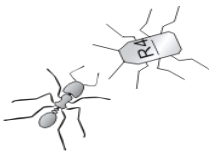


Sigmoid function:

- continuous
- non-linear
- monotonic
- bounded
- asymptotic

$$\Phi(x) = \frac{1}{1 + e^{-kx}}$$

$$\Phi(x) = \tanh(kx)$$

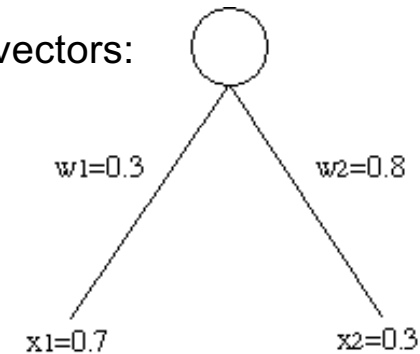


Neurons signal “familiarity”

The output of a neuron is a measure of similarity between its input pattern and its pattern of connection weights.

1. Output of a neuron is the dot product of the weight and input vectors:

$$y = a\left(\sum_i^N w_i x_i\right), \quad a = 1 \longrightarrow y = \mathbf{w} \cdot \mathbf{x}$$



2. Distance between two vectors is:

$$\cos \vartheta = \frac{\mathbf{w} \cdot \mathbf{x}}{\|\mathbf{w}\| \|\mathbf{x}\|}, \quad 0 \leq \vartheta \leq \pi$$

where the vector length is:

$$\|\mathbf{x}\| = \sqrt{\mathbf{x} \cdot \mathbf{x}} = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2}$$

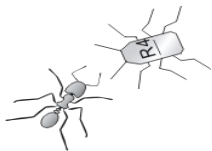
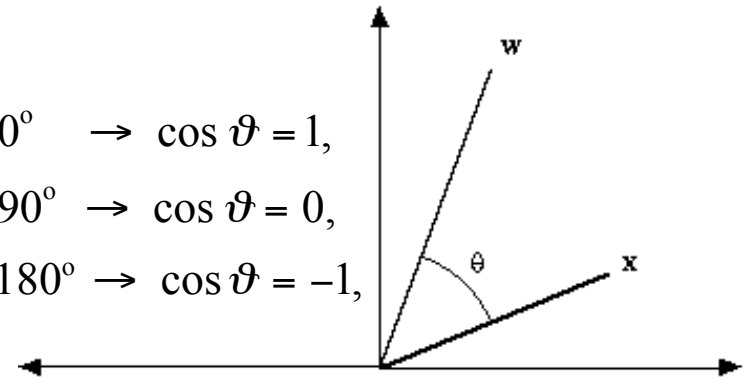
3. Output signals vector distance (familiarity)

$$\mathbf{w} \cdot \mathbf{x} = \|\mathbf{w}\| \|\mathbf{x}\| \cos \vartheta$$

$$\vartheta = 0^\circ \rightarrow \cos \vartheta = 1,$$

$$\vartheta = 90^\circ \rightarrow \cos \vartheta = 0,$$

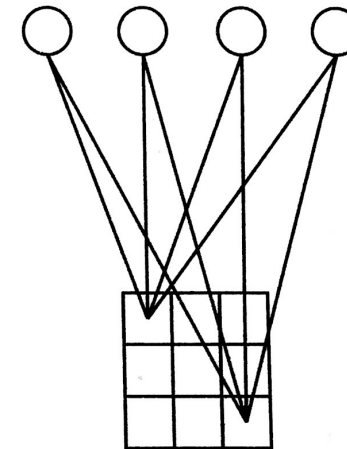
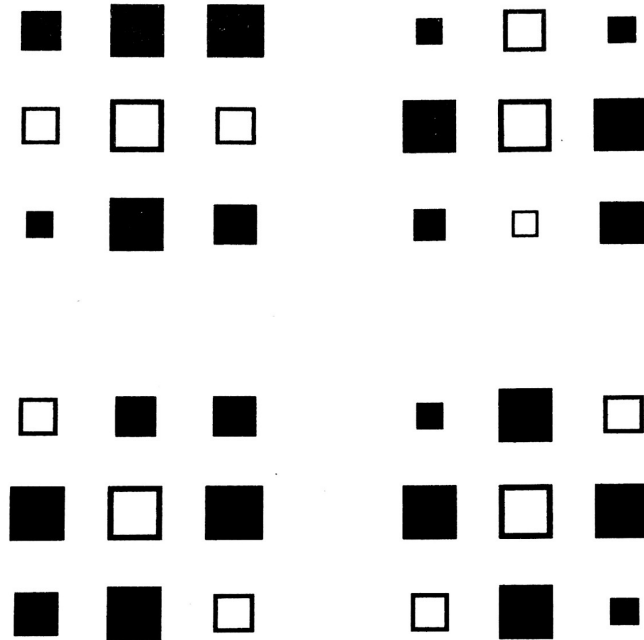
$$\vartheta = 180^\circ \rightarrow \cos \vartheta = -1,$$



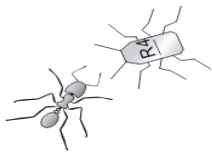
Neural Receptive Fields

The **Receptive Field** indicates the input area subtended by a neuron *and* the input pattern that generates the strongest activation.

RF can be visualized by plotting the weight pattern in the input space.



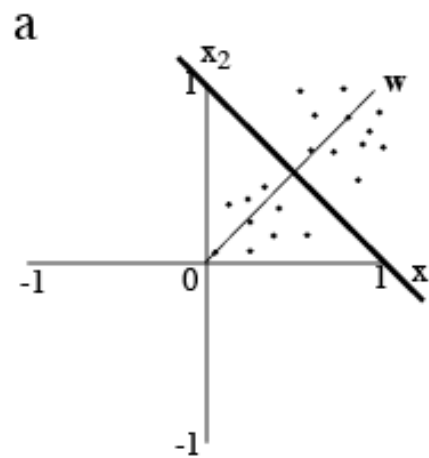
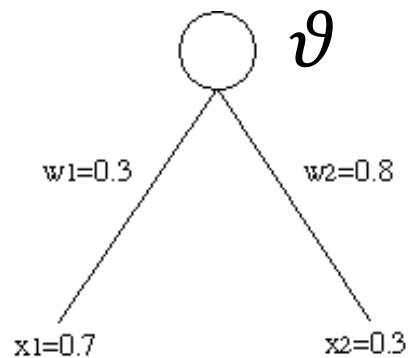
Fully connected
(only some
connections are
shown)



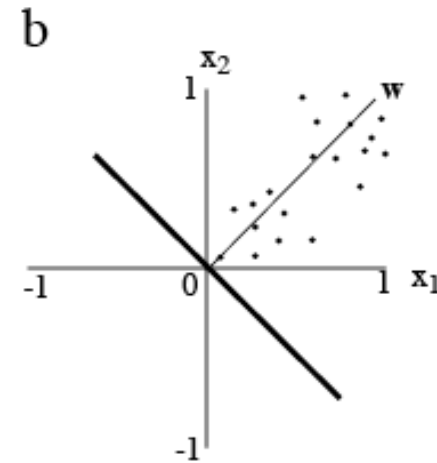
Neurons can act as classifiers

A neuron divides the input space in two regions, one where weighted input sum ≥ 0 and one where weighted input sum < 0 . The separation line is defined by the synaptic weights

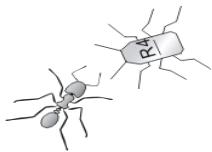
$$w_1x_1 + w_2x_2 - \vartheta = 0 \quad x_2 = \frac{\vartheta}{w_2} - \frac{w_1}{w_2}x_1$$



$$\vartheta > 0$$



$$\vartheta = 0$$

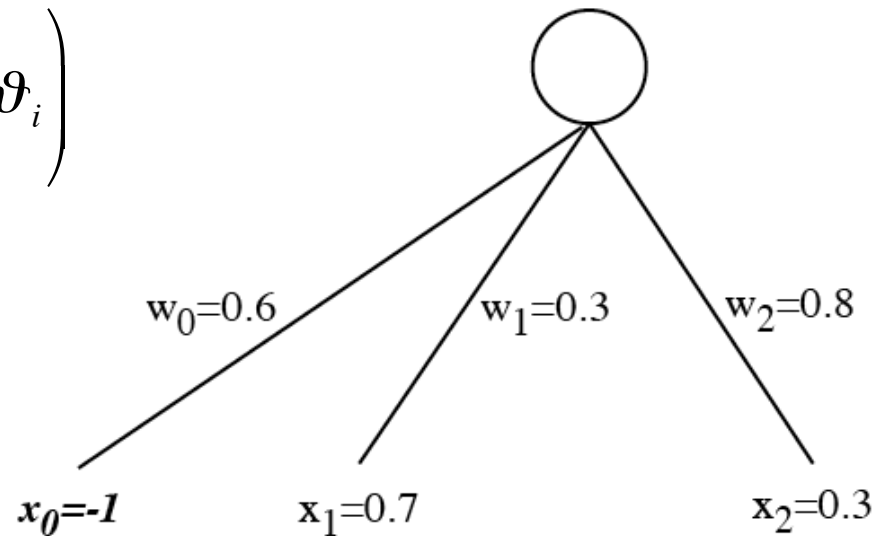


From Threshold to Bias unit

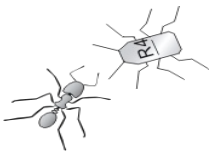
The threshold can be expressed as an additional weighted input from a special unit, known as bias unit, whose output is always -1.

$$y_i = \Phi(A_i) = \Phi\left(\sum_{j=1}^N w_{ij} x_j - \vartheta_i\right)$$

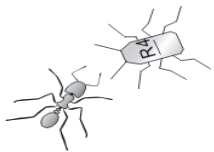
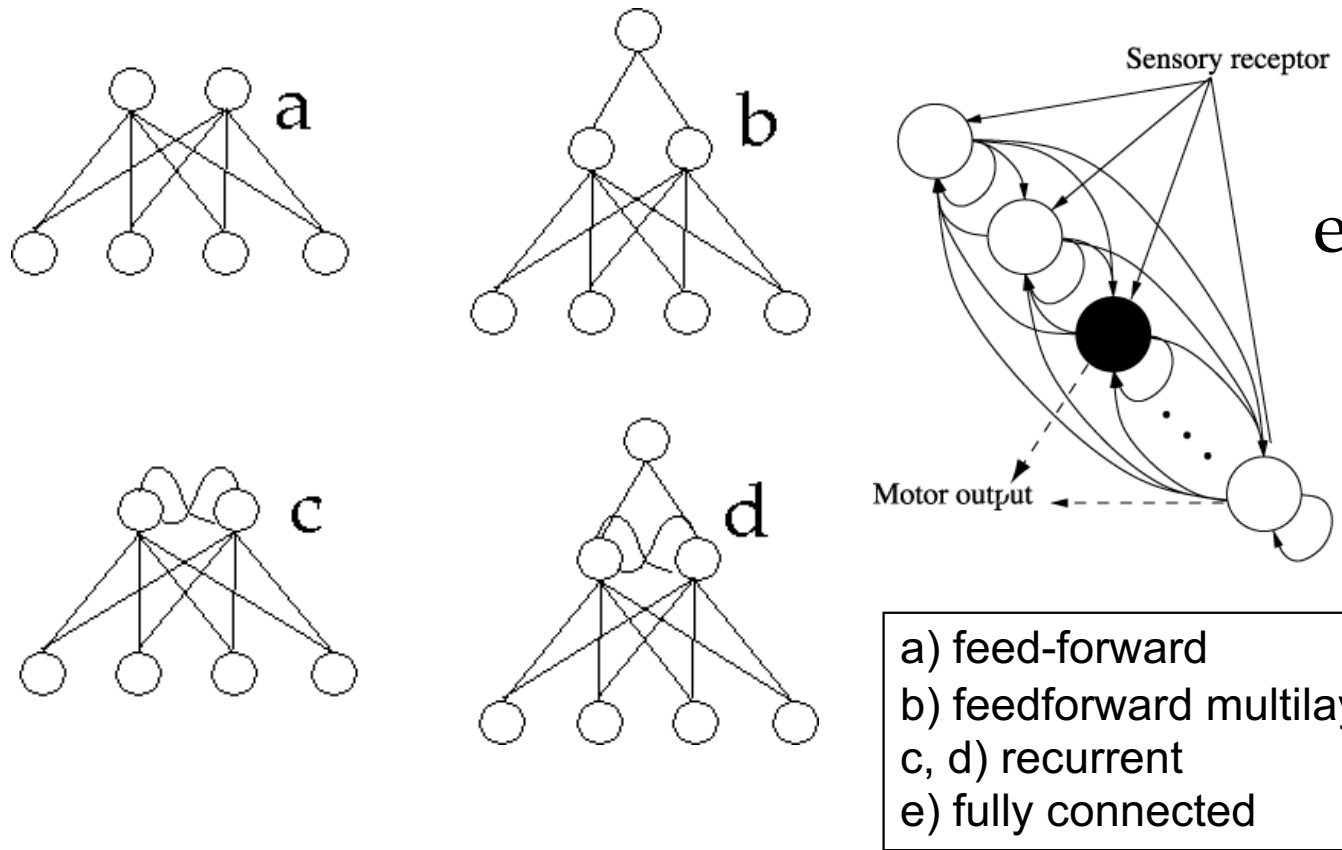
$$y_i = \Phi(A_i) = \Phi\left(\sum_{j=0}^N w_{ij} x_j\right)$$



- Easier to express/program
- Threshold is adaptable like other weights

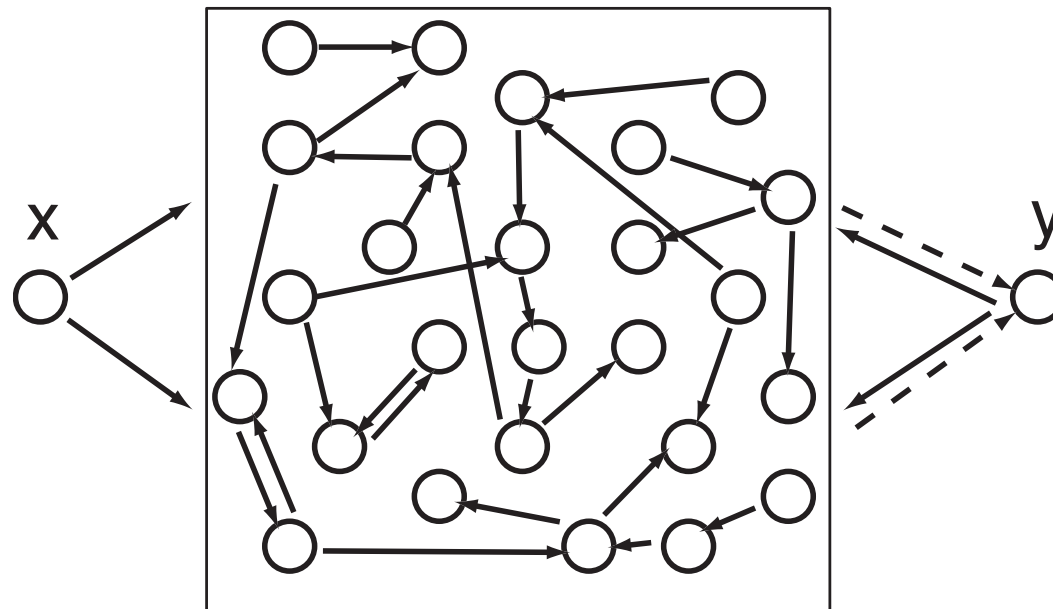


Architectures



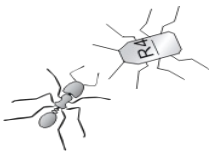
Reservoir Architectures

Exploit rich dynamics in the reservoir of hundreds of randomly interconnected neurons with low connectivity (0.01, e.g)

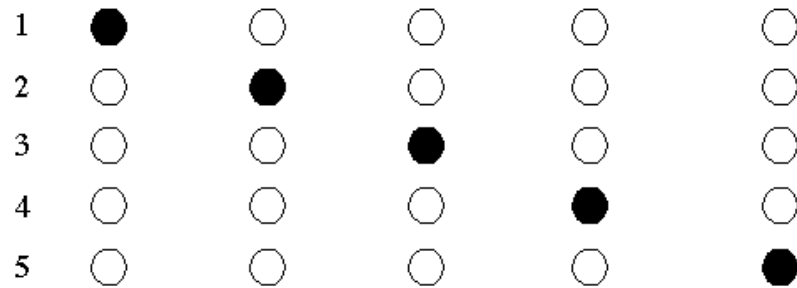


Liquid State Machines (Maas et al, 2002)

Echo State Networks (Jaeger et Haas, 2004)

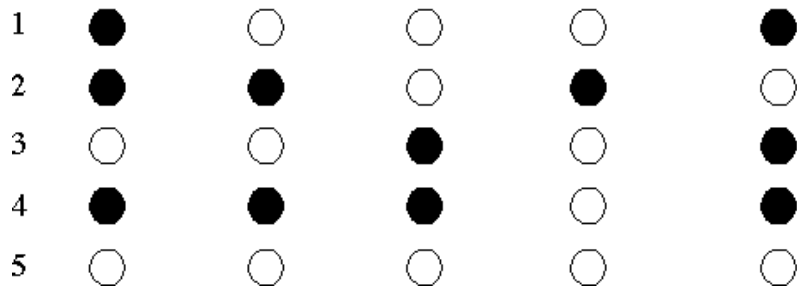


Local vs Distributed Input Encoding



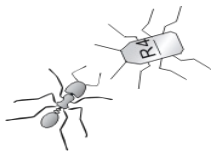
LOCAL

One neuron stands for one item
a.k.a. «Grandmother neurons»
Scalability problem



DISTRIBUTED

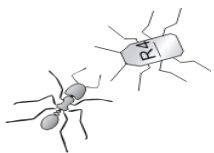
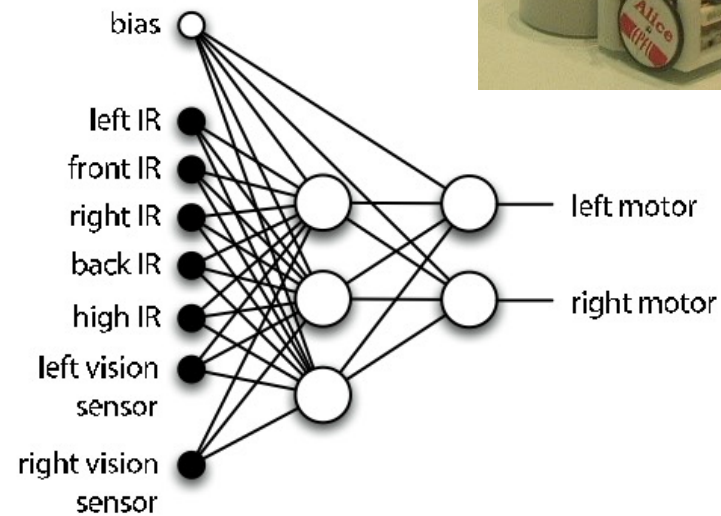
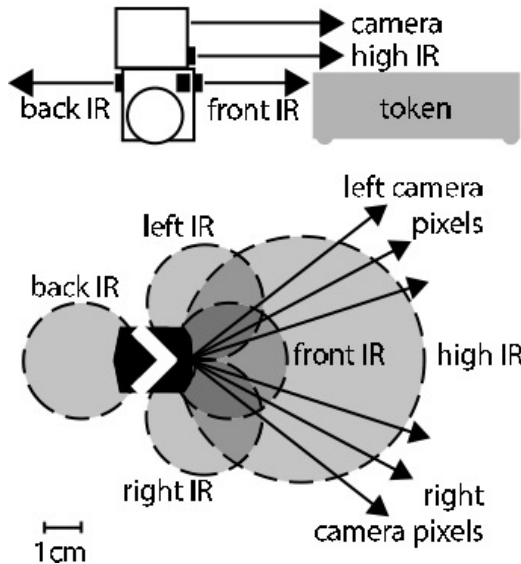
Neurons encode features
One neuron may stand for >1 item
One item may activate >1 neuron
Robust to damage



Normalisation of sensory input

Input signals from different sensory sources can have different amplitudes that must be normalised to enable comparisons by receiving neurons

$$x'_i = \frac{x_i}{\sqrt{\sum_{j=1}^N x_j^2}}$$

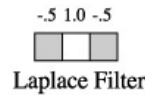


Convolution to capture spatial relationships

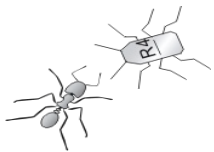
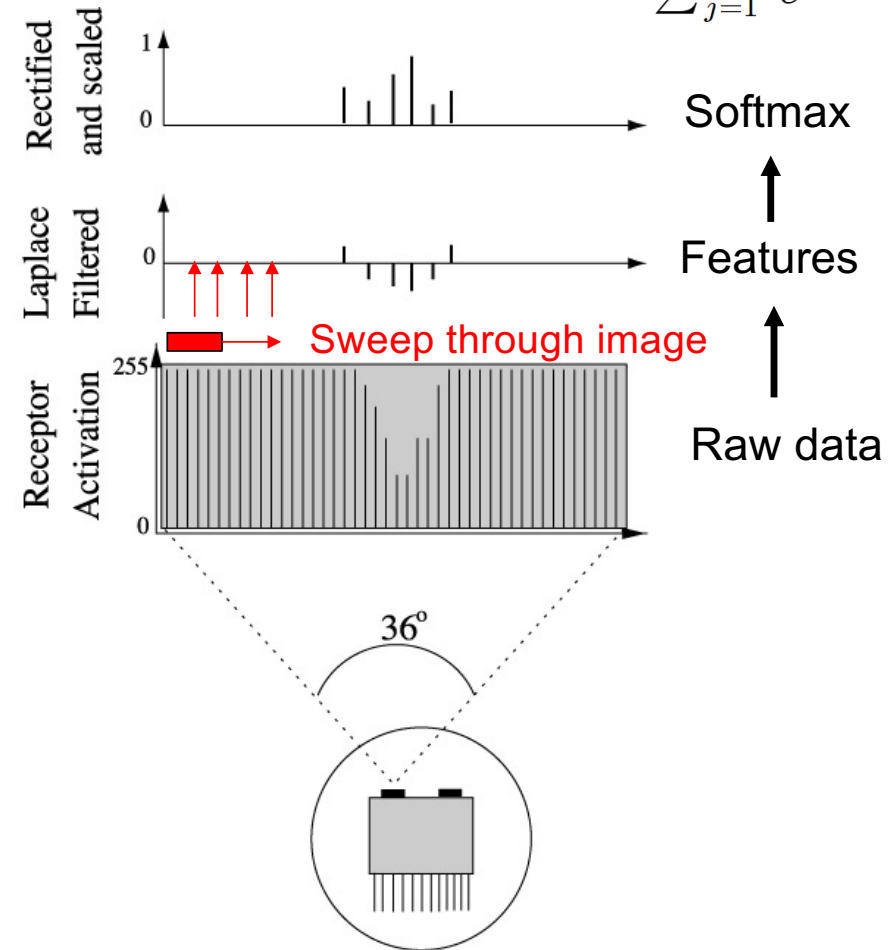
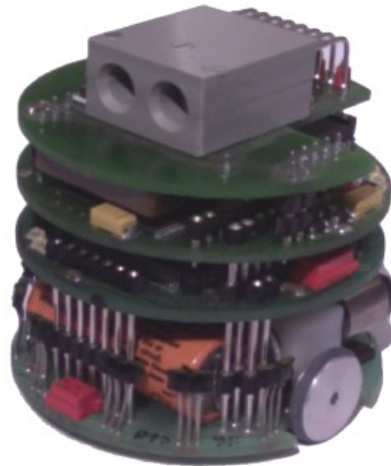
$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

We want to find in the image locations with features of interest, for example a brightness contrast

Design a filter (vector) that captures the feature, for example a Laplace filter

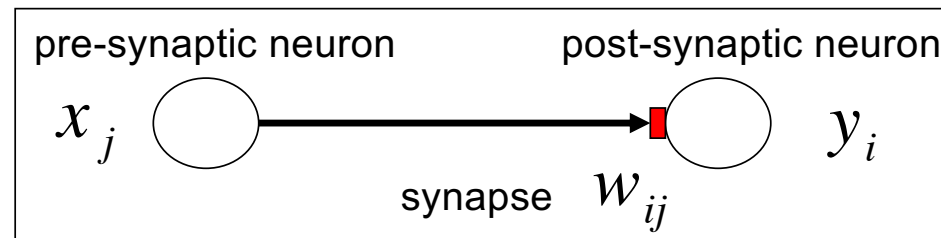


Convolve the image with the filter



Learning

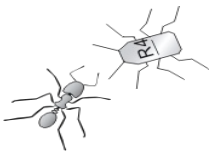
Learning is experience-dependent modification of connection weights



$$\Delta w_{ij} = x_j y_i$$

Hebb's rule (1949)

Learning is a gradual process and requires many input-output comparisons



Learning cycle

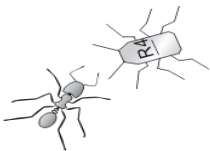
1. Initialize weights (e.g., random values from normal distribution)
2. Present randomly selected input pattern to network
3. Compute values of output units
4. Compute weight modifications
5. Update weights

Standard weight update

$$w_{ij}^t = w_{ij}^{t-1} + \eta \Delta w_{ij}$$

η learning rate $[0,1]$

6. Repeat from 2. until weights do not change anymore



Learning modalities

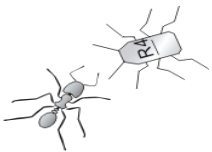
Unsupervised learning

Supervised learning

Reinforcement learning

Evolution

Evolution and learning



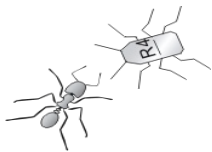
Unsupervised learning: what for?



Input: x (images, signals, text, etc.)

Categories (labels): none

Goal: learn compact structure (features) that describes input



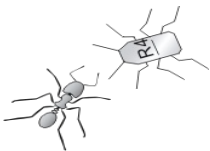
Unsupervised learning

The weight change depends on the activity of the pre-synaptic and of the postsynaptic neurons:

$$\Delta w_{ij} = x_j y_i$$

Unsupervised learning is used for

- Detecting statistical features of the input distribution
- Data compression and reconstruction
- Detect topological relationships in the input data
- Memorization

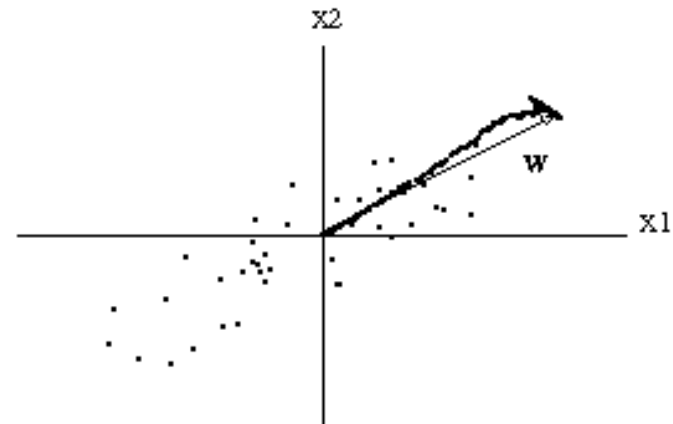
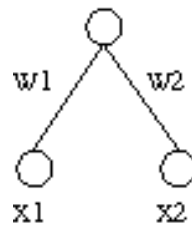


Oja's learning rule

Hebb's rule suffers from **self-amplification** (unbounded growth of weights), **but** biological synapses cannot grow indefinitely

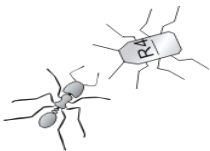
Oja (1982) introduced self-limiting growth factor in Hebb rule

$$\Delta w_j = \eta y (x_j - w_j y)$$



As a result, the weight vector develops along the direction of maximal variance of the input distribution.

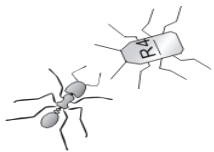
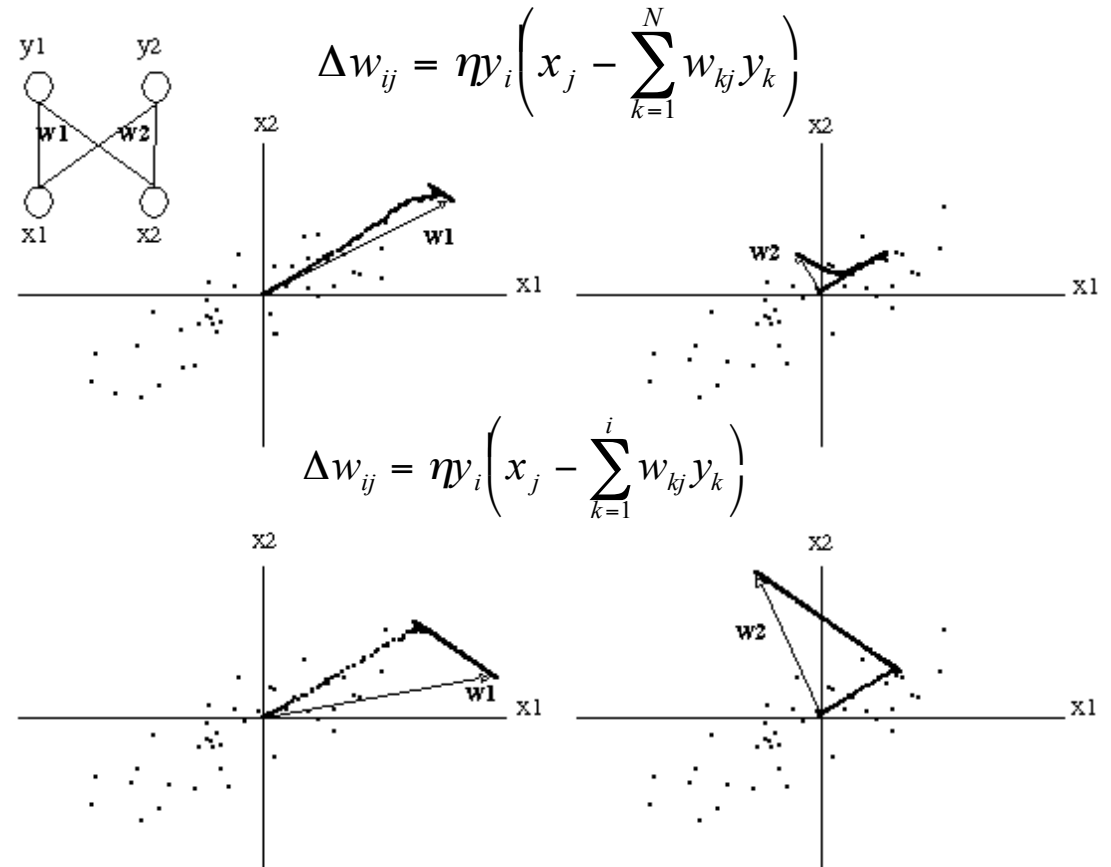
Neuron learns **how familiar** a new pattern is: input patterns that are closer to this vector elicit stronger response than patterns that are far away.



Principal Component Analysis

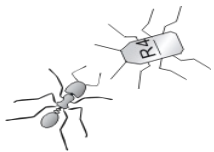
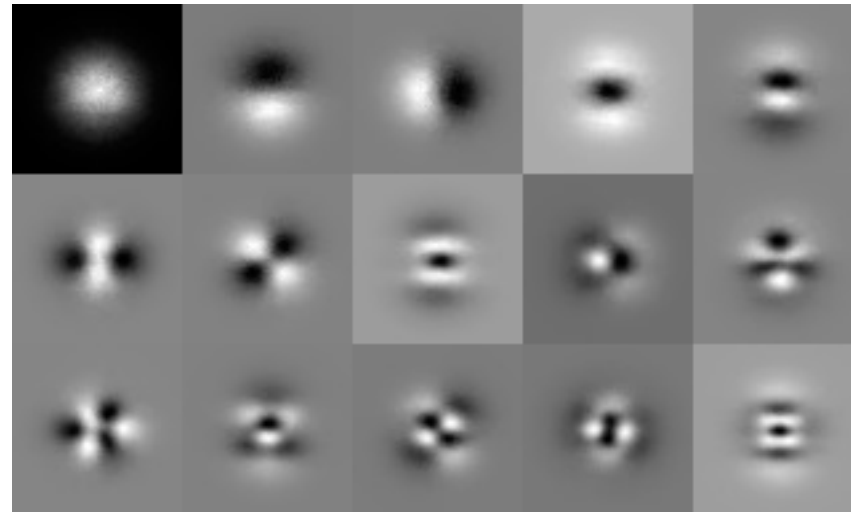
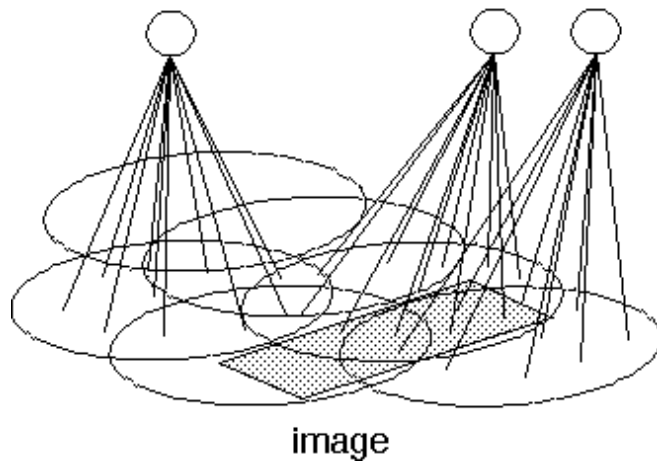
Oja rule for N output units develops weights that span the sub-space of the N principal components of the input distribution.

Sanger rule for N output units develops weights that correspond to the N principal components of the input distribution.



Do brains compute PCA?

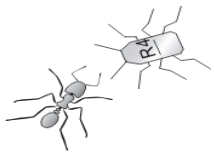
An Oja network with multiple output units exposed to a large set of natural images develops *receptive fields* similar to those found in the visual cortex of all mammals [Hancock et al., 1992]



Mammals are born with pre-formed hierarchically-organized feature detectors. But they never saw anything in the womb: how can it be?

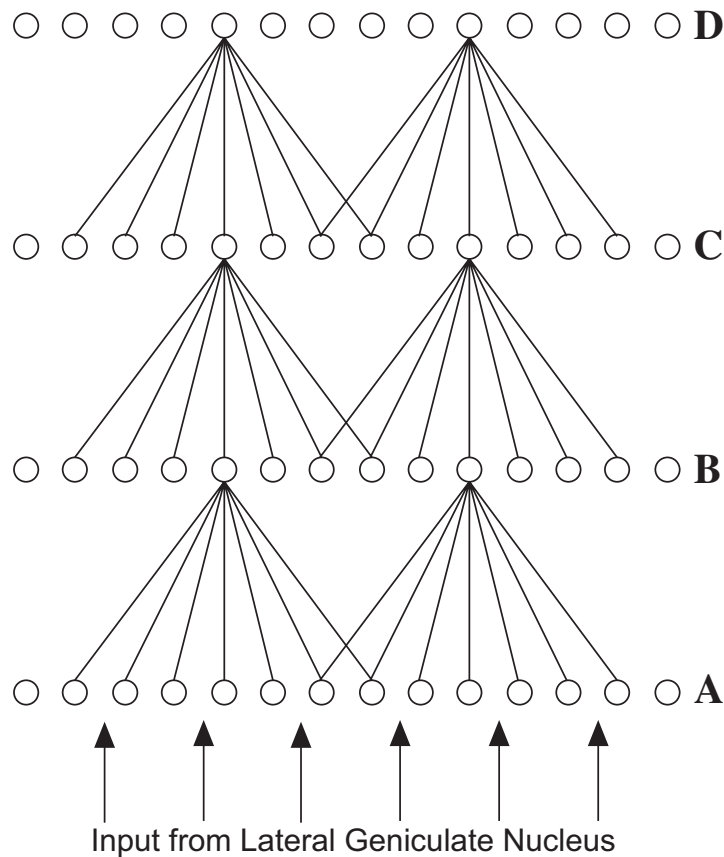


Image: Kelly Frankenburg



Multilayer Feature Detection

Linsker (1986)

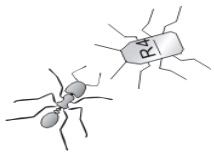


Topologically restricted connectivity

Linear activation function

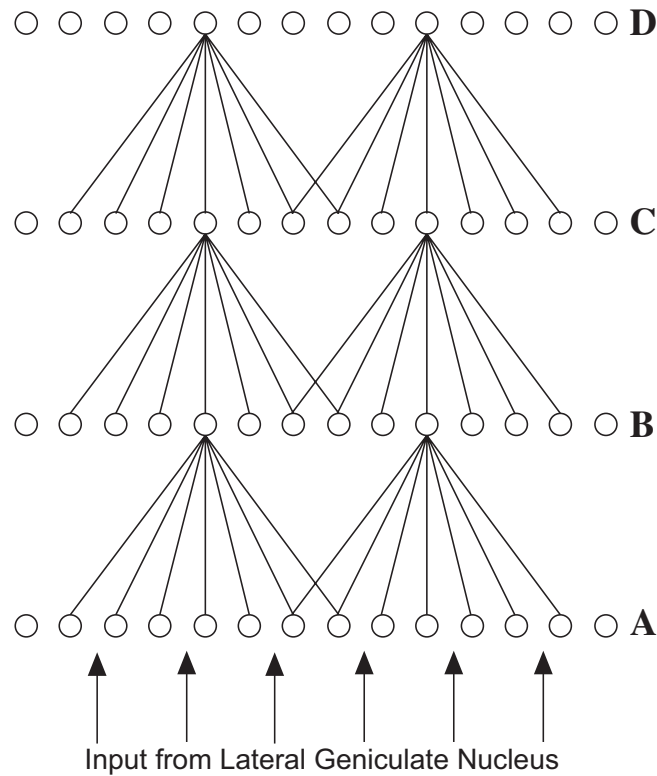
Plain Hebbian learning with weight clipping at w_+ and w_-

Learn one layer at a time, starting from lower layer

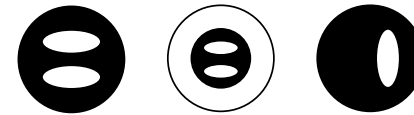


Emerging Receptive Fields

Linsker (1986)



Complex feature detectors

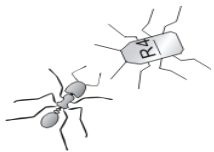
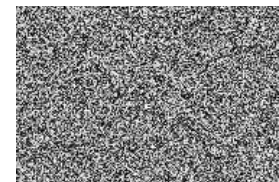


Simple feature detectors



Average luminosity in RF

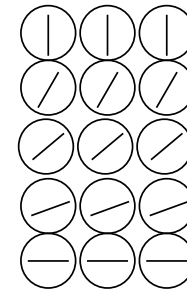
+++++



Sensory maps

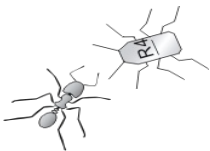
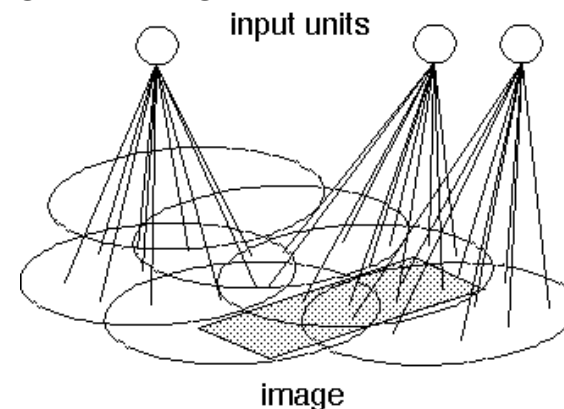
Neighbouring neurons respond to similar patterns with gradual transitions

The visual cortex is organized in specialized modules. Each module is composed by a series of columns of neurons. For example, neurons in early modules respond to bars at different orientation

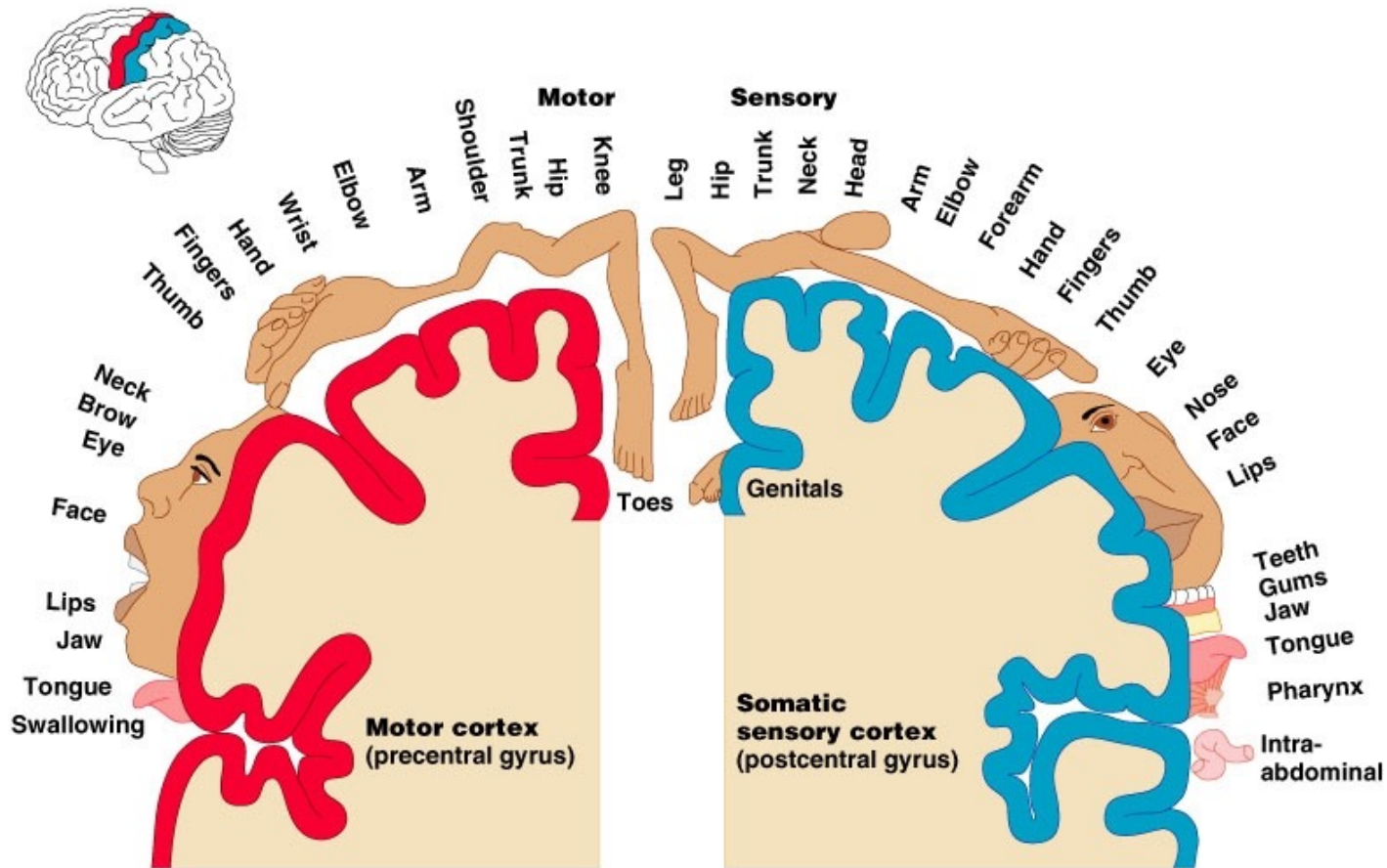


1. The bar orientation gradually varies along the column.
2. Neighbouring columns correspond to neighbouring areas of the retina (**retinotopic maps**).

A similar structure exists in the auditory cortex (**tonotopic maps**).

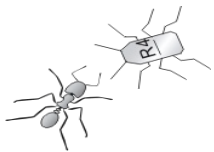


Sensory-Motor Body Map



Copyright © 2004 Pearson Education, Inc., publishing as Benjamin Cummings.

Companion slides for the book *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies* by Dario Floreano and Claudio Mattiussi, MIT Press



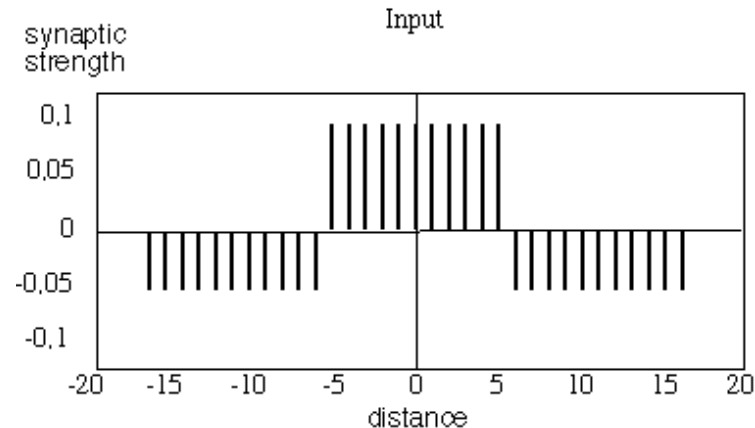
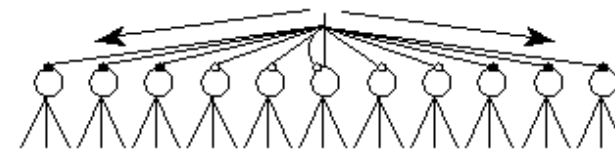
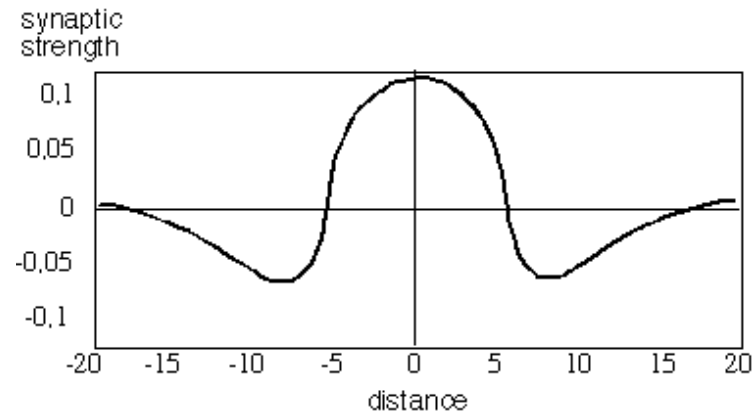
Lateral connections to neighbouring neurons

Cortical neurons display the following pattern of *projective* connectivity:

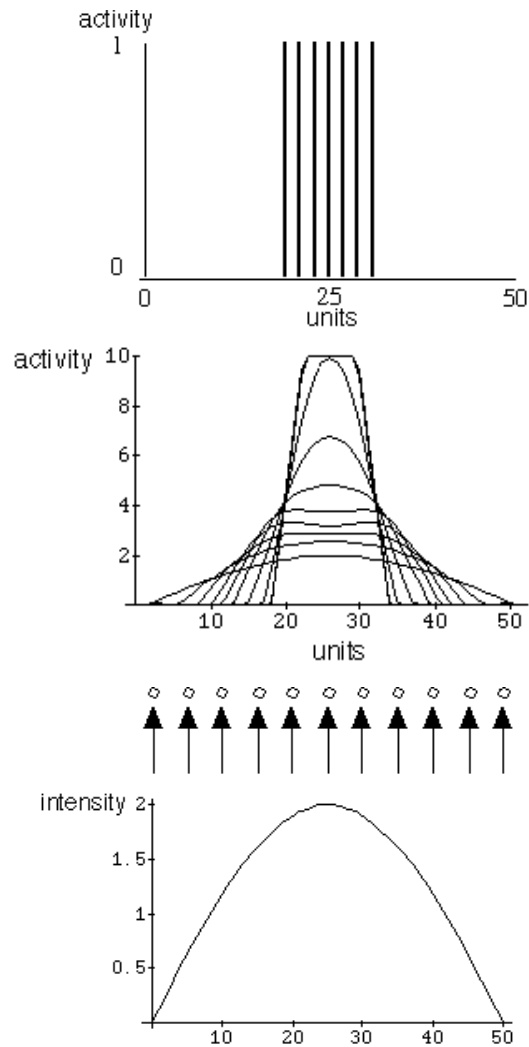
- up to 50-100 μm radius = excitatory
- up to 200-500 μm radius = inhibitory
- up to few cm radius = slightly excitatory

Also known as *Mexican Hat distribution*

In a neural network, we can approximate the Mexican hat with a bipolar weight distribution.



Formation of neural bubbles around strongest input



Simplification: set output of unit with highest activation and its n neighbors to 1, and all other units to 0



Gradual emergence of bubble centered around unit with strongest activation



Laterally connected neurons



Input pattern

Self-Organizing Topological Maps

Kohonen (1982)

Let's apply Hebb rule to a layer of laterally connected neurons

$$y_i = \Phi(A_i) = \begin{cases} 1 & \text{if within neighbourhood } \Psi(y) \text{ of neuron } y \text{ with highest activation} \\ 0 & \text{otherwise} \end{cases}$$

$$\Delta w_{ij} = \eta y_i (x_j - \Psi(y_i) w_{ij}) \quad \Psi(y_i) = \begin{cases} \psi & \text{if } y_i = 1 \\ 0 & \text{if } y_i = 0 \end{cases}$$

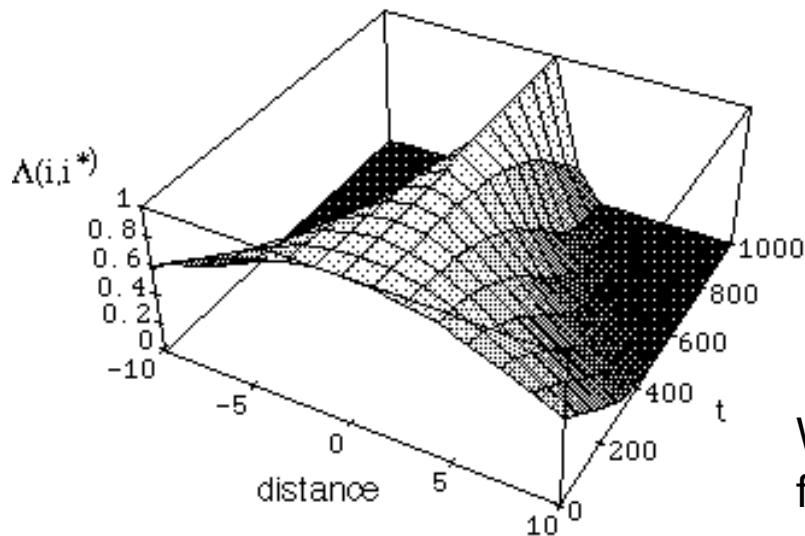
If we set $\Psi(y_i)$ equal to η , then the learning rule becomes:

$$\Delta w_{ij} = \begin{cases} \eta(x_j - w_{ij}) & \text{if } y_i = 1 \\ 0 & \text{if } y_i = 0 \end{cases} \quad \text{and} \quad \mathbf{w}_i^{t+1} = \begin{cases} \mathbf{w}_i^t + \eta(\mathbf{x} - \mathbf{w}_i^t) & \text{if } y_i = 1 \\ \mathbf{w}_i^t & \text{if } y_i = 0 \end{cases}$$

1. The weights are changed only for the neurons that are geographically near the neuron with the highest activity,
2. The change moves the weight vector towards the input pattern.

Neighborhood function

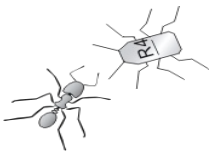
The neighbourhood size $\Psi(y)$ is a critical aspect of map self-organization. It should be large at the beginning of training to give a chance to all neurons to change weights and gradually shrink



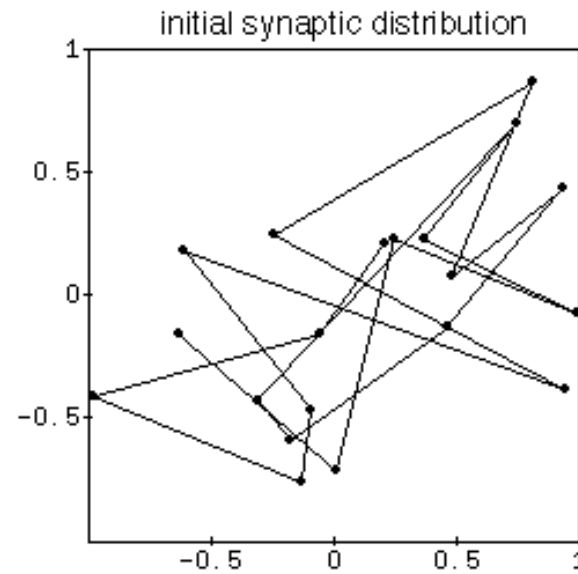
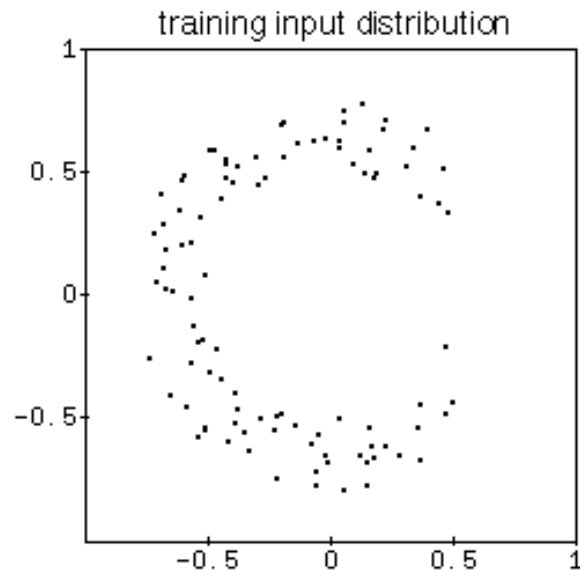
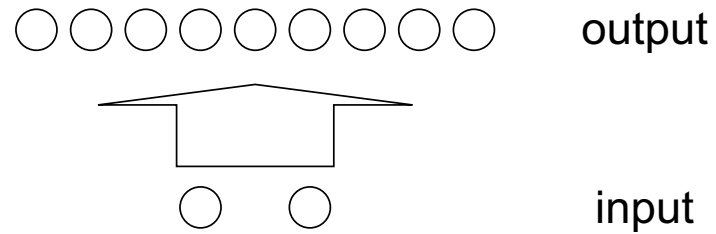
$$\Lambda(i, i^*) = \begin{cases} 1 & \text{if } \|\mathbf{c}_i - \mathbf{c}_{i^*}\| \leq r \\ 0 & \text{otherwise} \end{cases}$$

We can incorporate the neighborhood function in the learning rule

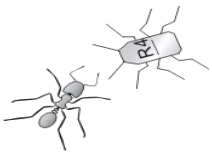
$$\Delta w_{ij} = \eta \Lambda(i, i^*) (x_j - w_{ij})$$



Example of self-organizing map

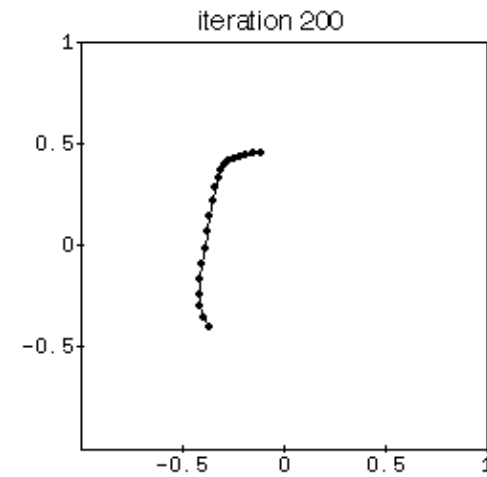
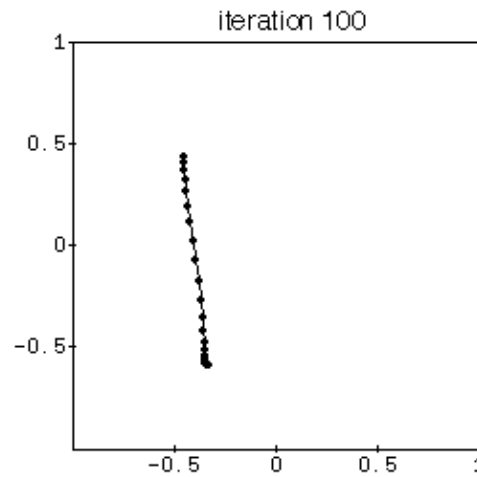


Companion slides for the book *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies* by Dario Floreano and Claudio Mattiussi, MIT Press

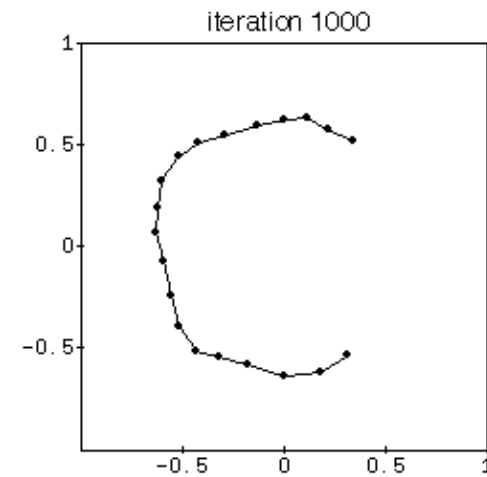
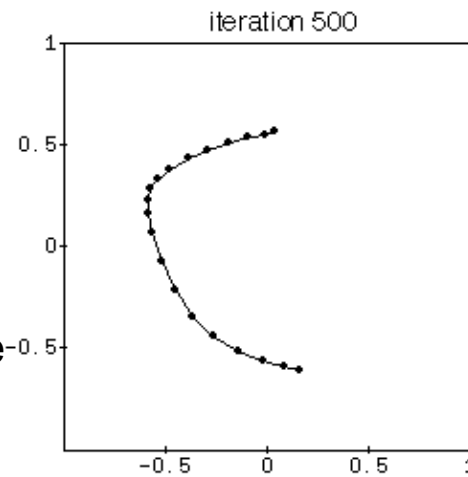


Self-organization phases

Ordering phase:
Fast
Neighborhood change



Convergence phase:
Slow
No neighborhood change



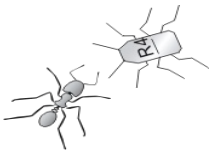
Supervised learning: what for



Input: x (images, signals, text, etc.)

Category (label): y (eat, wear, wear, eat, wear, eat)

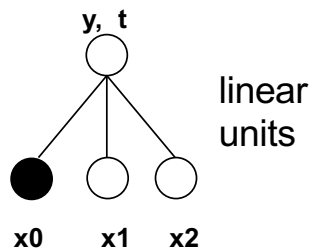
Goal: learn mapping between input data and labels



Supervised Learning

- **Teacher** provides desired responses for a set of training patterns
- Synaptic weights are modified in order to reduce the **error** between the output y and its desired output t (a.k.a. teaching input)

Widrow-Hoff defined the error with the symbol delta: $\delta_i = t_i - y_i$
(a.k.a. delta rule)



repeat
for every
input/output
pair until
error is 0

$$w_{ij} = rnd(\pm 0.1)$$

initialize weights to random values

$$y_i = \sum_{j=0} w_{ij} x_j$$

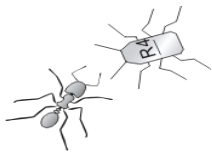
present input pattern and
compute neuron output

$$\Delta w_{ij} = \eta (t_i - y_i) x_j$$

compute weight change using
difference between desired
output and neuron output

$$w_{ij} = w_{ij}^{t-1} + \Delta w_{ij}$$

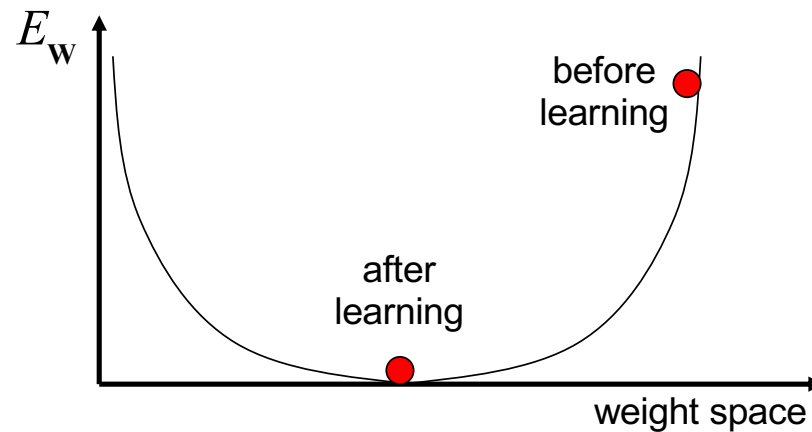
get new weights by adding
computed change to previous
weight values



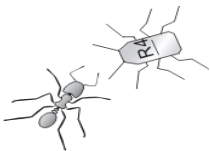
Error (loss) function

The delta rule modifies the weights to descend the gradient of the error function

$$E_{\mathbf{w}} = \frac{1}{2} \sum_{\mu} \sum_i \left(t_i^{\mu} - \sum_{j=0} w_{ij} x_j^{\mu} \right)^2$$



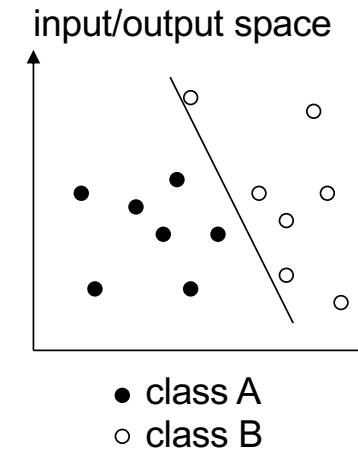
*Error space for a network with a single layer of synaptic weights
(perceptron, Rosenblatt, 1962)*



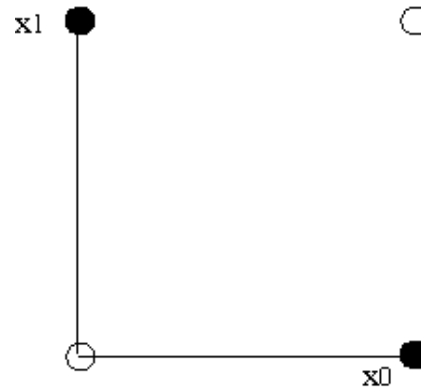
Linear Separability

Perceptrons can solve only problems whose input/output space is **linearly separable**.

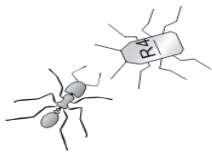
Several real world problems are not linearly separable.



Example of XOR problem

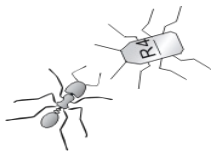
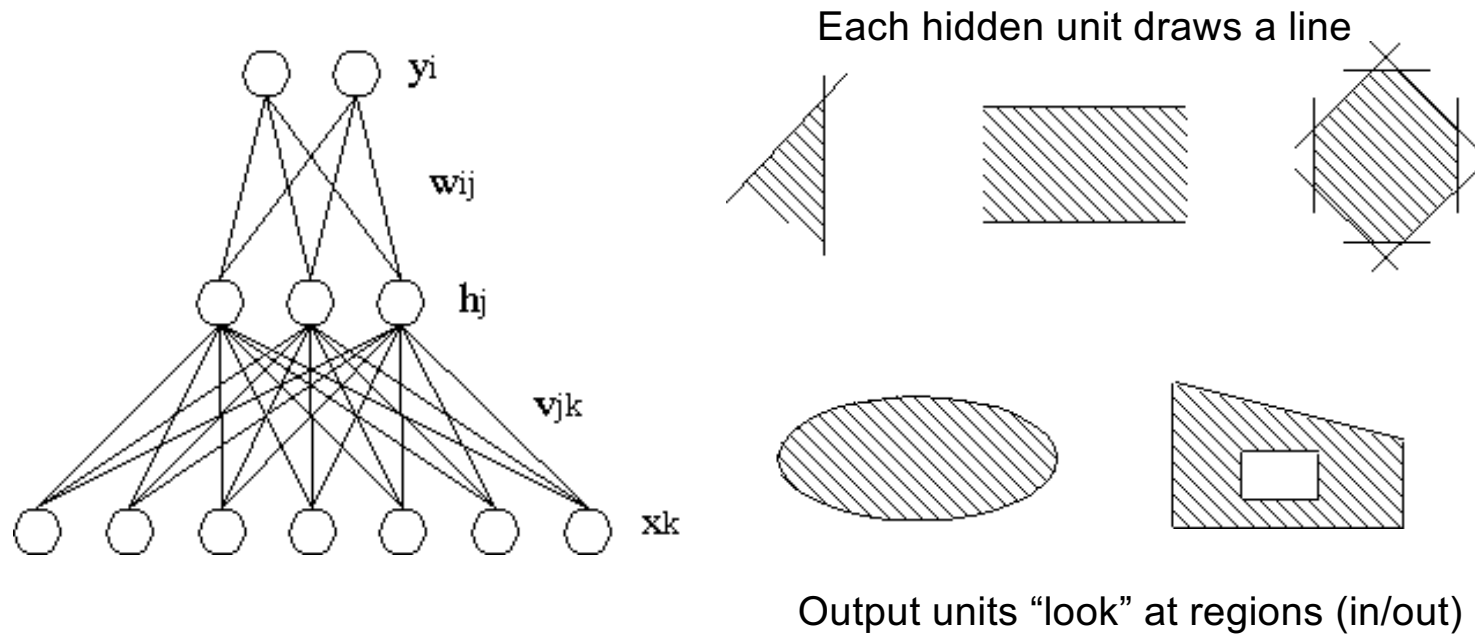


x_0	x_1	t	
0	0	0	○
1	1	0	○
1	0	1	●
0	1	1	●



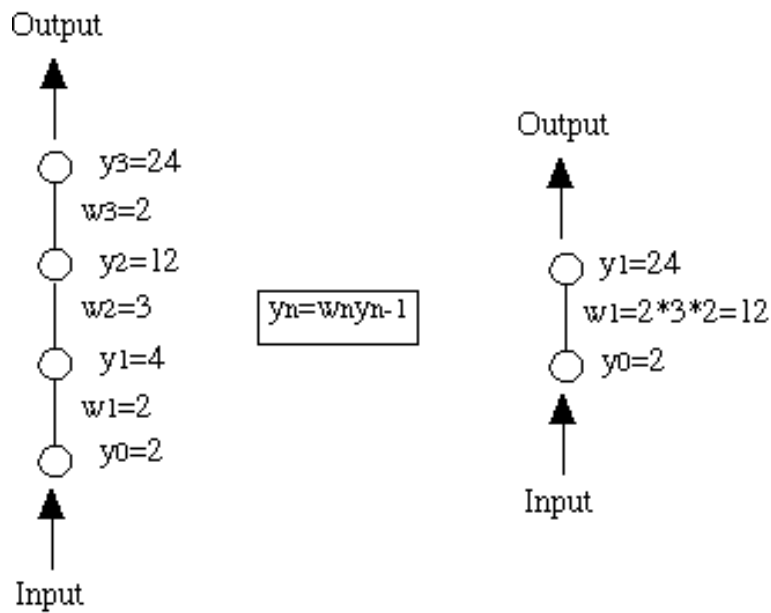
Multi-layer Perceptron (MLP)

- Multi-layer neural networks can solve problems that are not linearly separable
- Hidden units re-map input space into a space which can be linearly separated by output units.

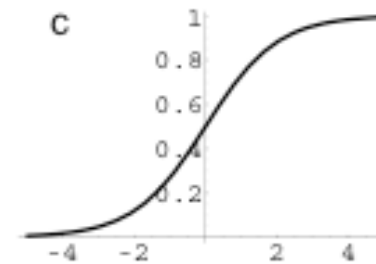


Output Function in MLP

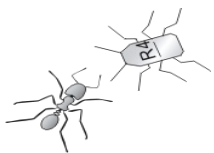
- Multi-layer networks should not use linear output functions because a linear transformation of a linear transformation remains a linear transformation.
- Therefore, such a network would be equivalent to a network with a single layer



For example, sigmoid function

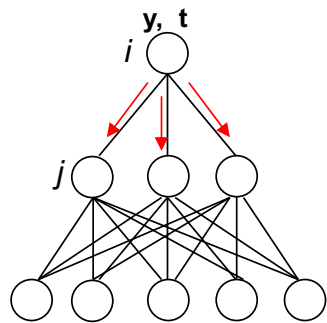


$$\Phi(x) = \frac{1}{1 + e^{-kx}}$$



Back-propagation of Error

In a simple perceptron, it is easy to change the weights to minimize the error between output of the network and desired output.



$$\delta_i = t_i - y_i$$

$$\Delta w_{ij} = \eta \delta_i x_j$$

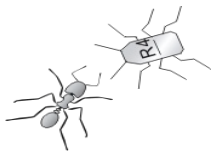
$$\delta_i = (t_i - y_i) \dot{\Phi}(A_i) \quad \text{in the case of non-linear output functions, add derivative of output}$$

In a multilayer network, what is the error of the hidden units? This information is needed to change the weights between input units and hidden units.

The idea suggested by Rumelhart et al. in 1986 is to propagate the error of the output units backward to the hidden units through the connection weights:

$$\delta_j = \dot{\Phi}(A_j) \sum_i w_{ij} \delta_i$$

Once we have the error for the hidden units, we can change the lower layer of connection weights with the same formula used for the upper layer.



Algorithm

1. Initialize weights (random, around 0)

2. Present pattern $x_k^\mu = s_k^\mu$

3. Compute hidden $h_j^\mu = \Phi\left(\sum_k v_{jk} x_k^\mu\right)$

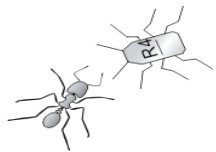
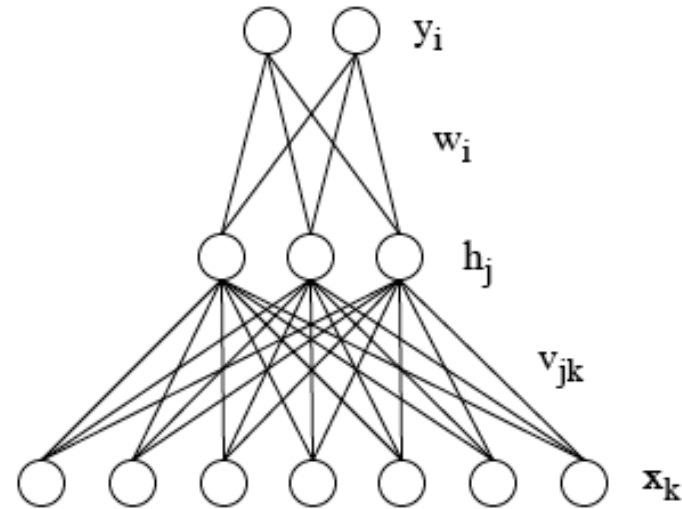
4. Compute output $y_i^\mu = \Phi\left(\sum_j w_{ij} h_j^\mu\right)$

5. Compute delta output $\delta_i^\mu = \dot{\Phi}\left(\sum_j w_{ij} h_j^\mu\right)(t_i^\mu - y_i^\mu)$ $\delta_i^\mu = y_i^\mu(1 - y_i^\mu)(t_i^\mu - y_i^\mu)$

6. Compute delta hidden $\delta_j^\mu = h_j^\mu(1 - h_j^\mu) \sum_i w_{ij} \delta_i^\mu$

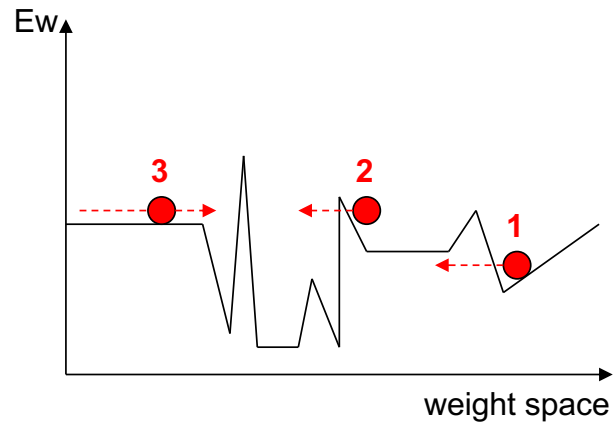
7. Compute weight change $\Delta w_{ij}^\mu = \delta_i^\mu h_j^\mu$, $\Delta v_{jk}^\mu = \delta_j^\mu x_k^\mu$

8. Update weights $w_{ij}^t = w_{ij}^{t-1} + \eta \Delta w_{ij}^\mu$, $v_{jk}^t = v_{jk}^{t-1} + \eta \Delta v_{jk}^\mu$

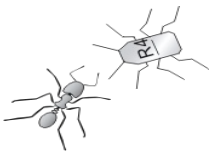


Using Back-Propagation

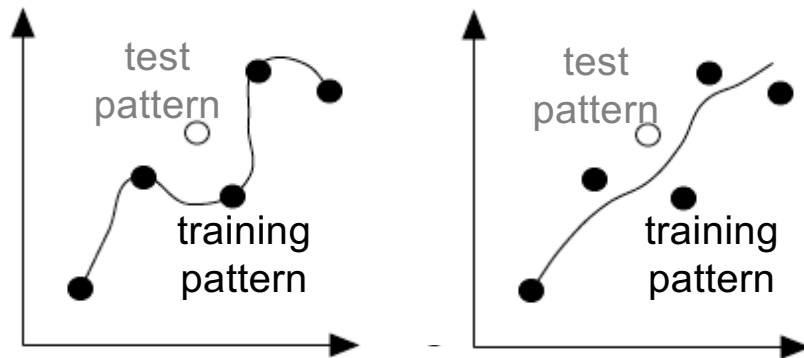
Error space can be complex in multilayer networks: local minima and flat areas



1. Large learning rate: take large steps in the direction of the gradient descent
2. Momentum: add direction component from last update $\Delta w_{ij}^t = \eta \delta_i + \alpha \Delta w_{ij}^{t-1}$
3. Additive constant: keep moving when no gradient $\delta_i^\mu = (\dot{\Phi} + k)(t_i^\mu - y_i^\mu)$

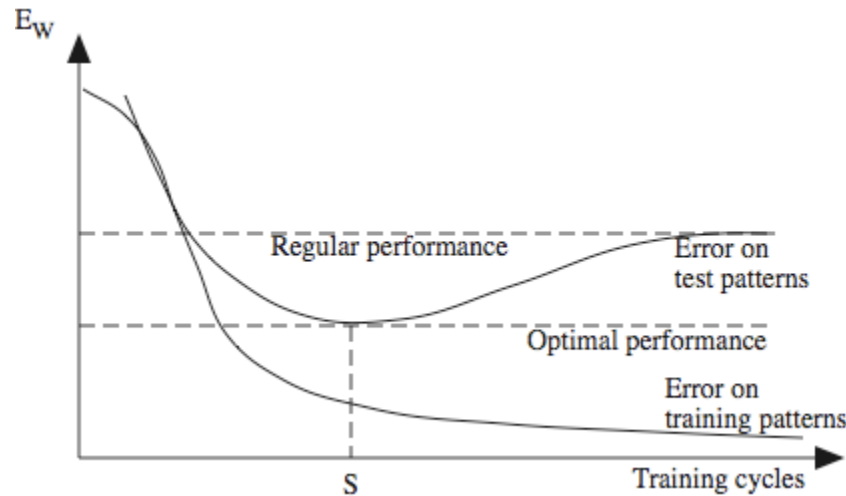


Over-fitting



Overfitting training data leads to poor generalisation

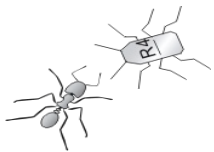
Overfitting can derive from too many weights and/or too long learning of training patterns



Solution: Use a Validation Set

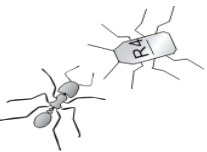
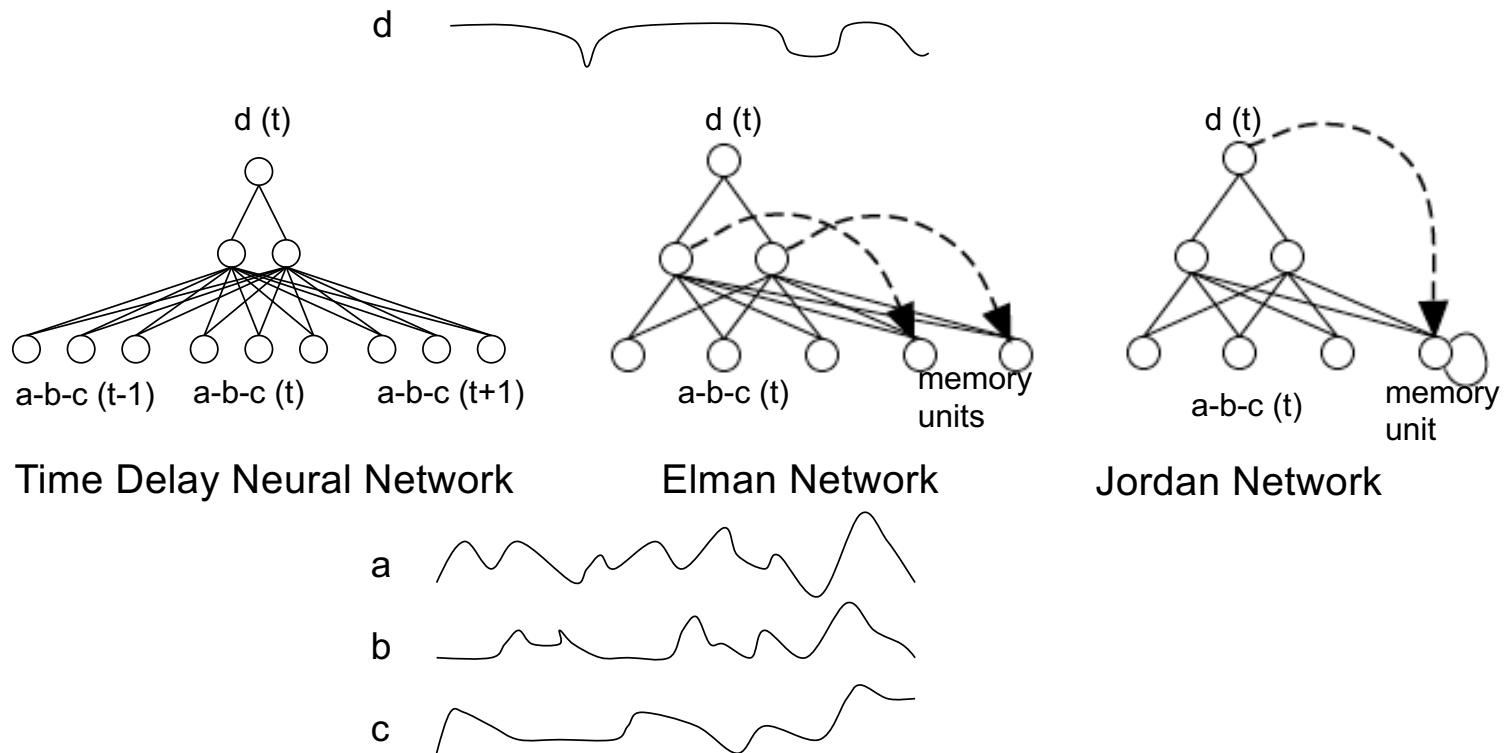
Divide available data into:

- training set (for weight update)
 - validation set (for error monitoring)
- Stop training when error for validation set starts growing



Time Series

Extraction of time-dependent features is necessary for time-series analysis



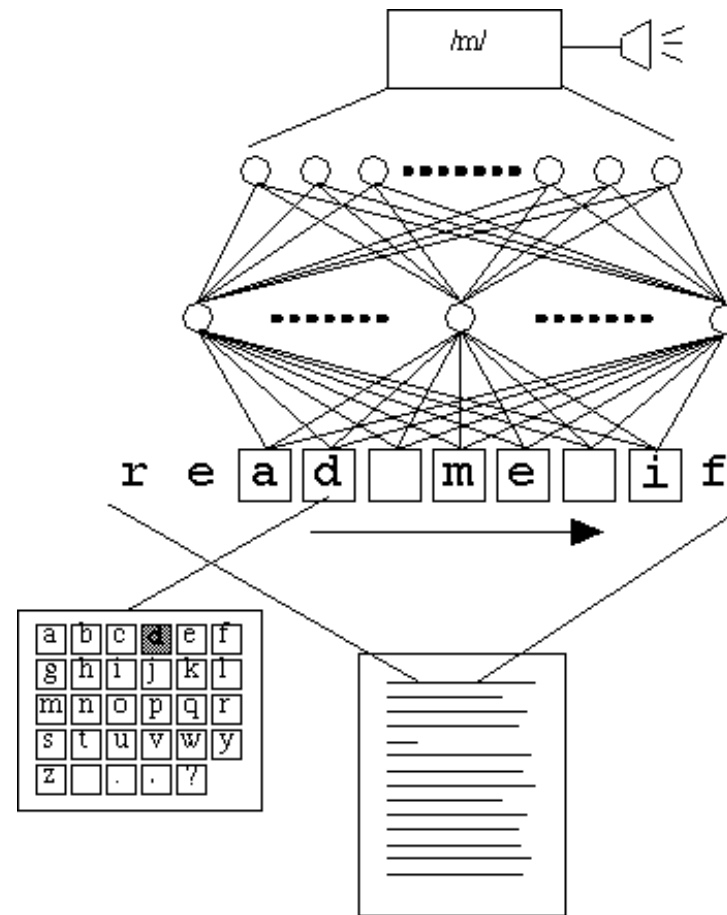
NETtalk

A neural network that learns to read aloud written text:

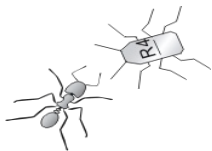
- 7 x 29 input units encode characters within a 7-position window(TDNN)
- 26 output units encode english phonemes
- approx. 80 hidden units

Training on 1000-word text, reads any text with 95% accuracy

Learns like humans: segmentation, bla-bla, short words, long words

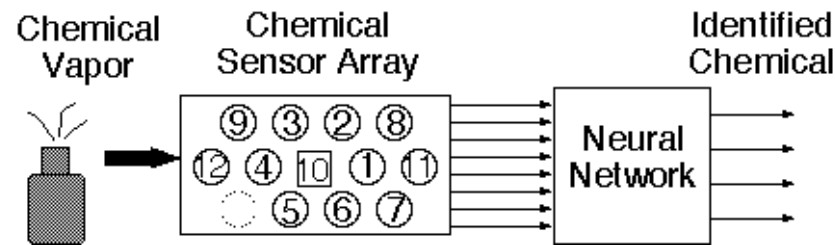


[Sejnowski & Rosenberg, 1987]

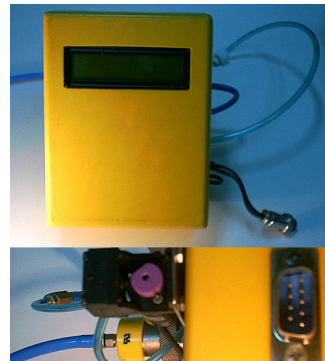


Chemical sensors of volatile molecules

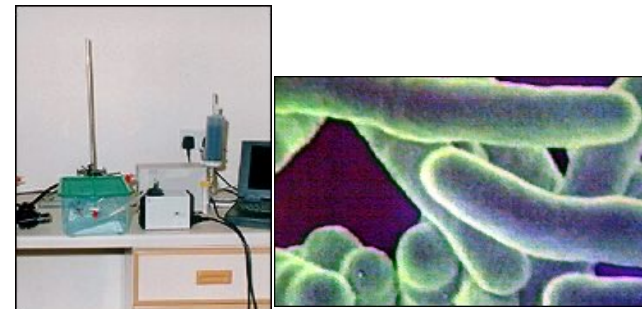
The human brain recognizes millions of smell types by combining responses of only 10,000 receptors. Smell detection is a multi-billion industry (food, cosmetics, medicine, environment monitoring...). Human detection: costly, fatigue, history, aging, subjective.



landmine detection
Tufts University



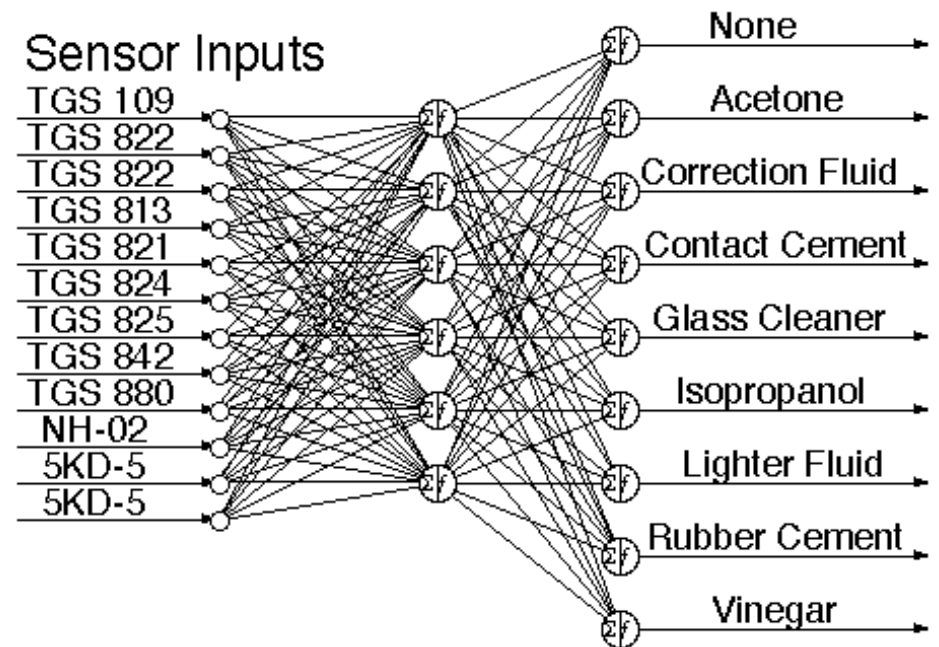
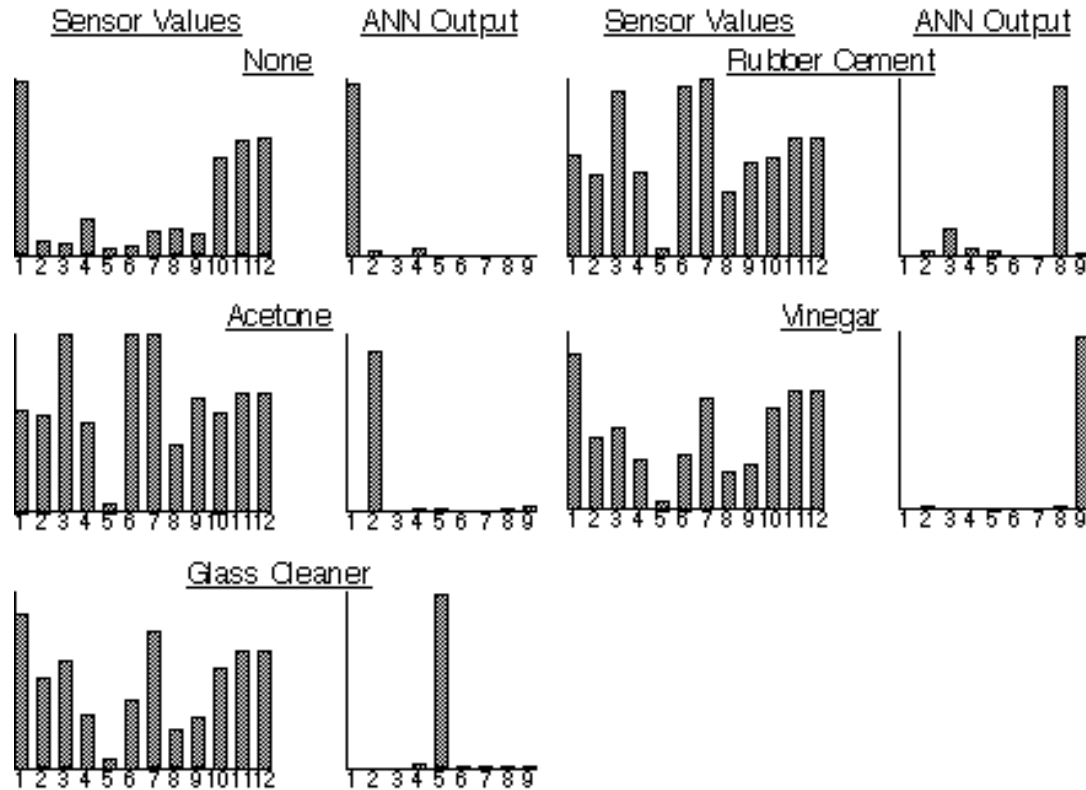
food quality
Pampa Inc.



tuberculosis diagnosis
Cranfield University



Substance recognition from chemical sensors



[Keller et al., 1994]

