# Learning in neural networks

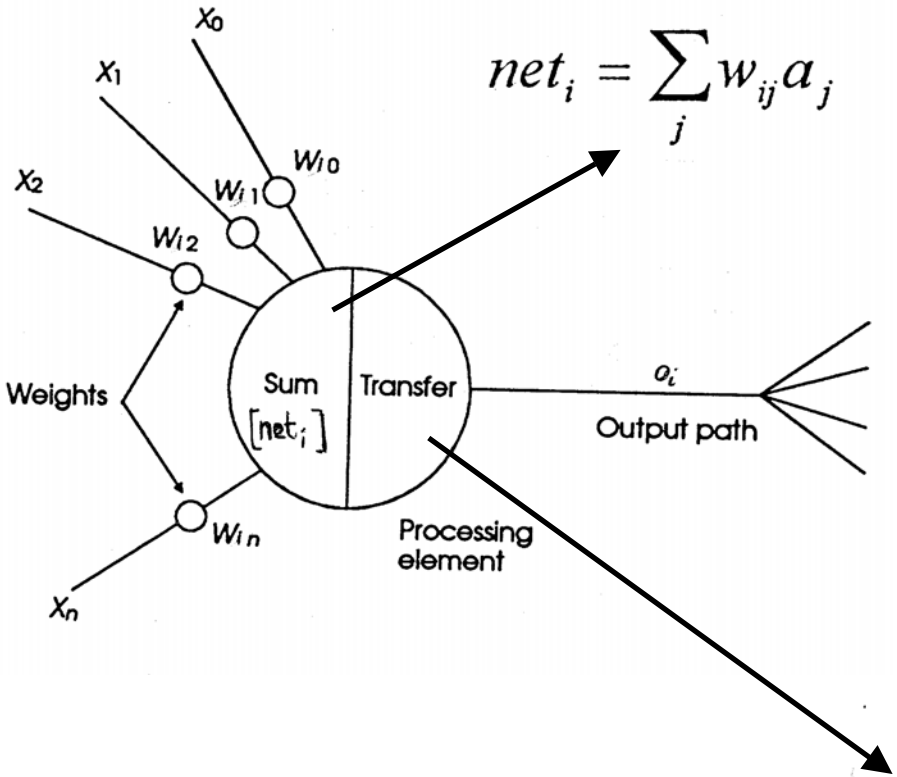
# Connectionist systems: Basic principles

**Neurons (processing units):** serve as detectors, signal with their activity

**Networks:** link, coordinate, amplify, and select patterns of activity

**Learning:** organizes networks to perform task and to develop internal models of the environment

# Artificial neuron




$$net_i = \sum_j w_{ij} a_j$$

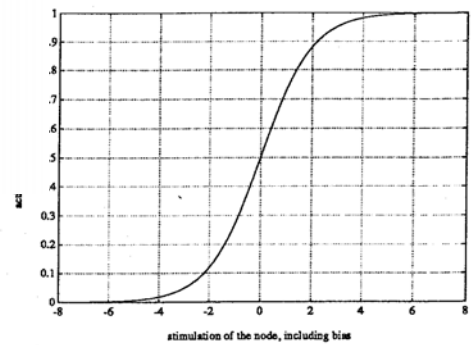



*Figure 4. The sigmoid function.*

$$o_i = \frac{1}{1 + e^{-net_i}}$$

# Networks

- The network *architecture* identifies:
  - groups or *layers* of different neurons
    - Units receiving input directly from the “environment” form the *input layer*
    - Units producing the final output of the network form the *output layer*
    - Units that do not make direct contact with input or output are called *hidden units*. Conversely, input and output units are sometimes named *visible units.*
  - the way in which neurons are connected to each other (*connectivity scheme*)
    - *Feed-forward network*: there are only unidirectional links from input to hidden to output units (bottom-up)
    - *Recurrent network*: there are bidirectional links, in which activation can spread backwards(top-down or feedback)
    - *Fully recurrent network*: as a recurrent network but there are also intra-layer (lateral) connections

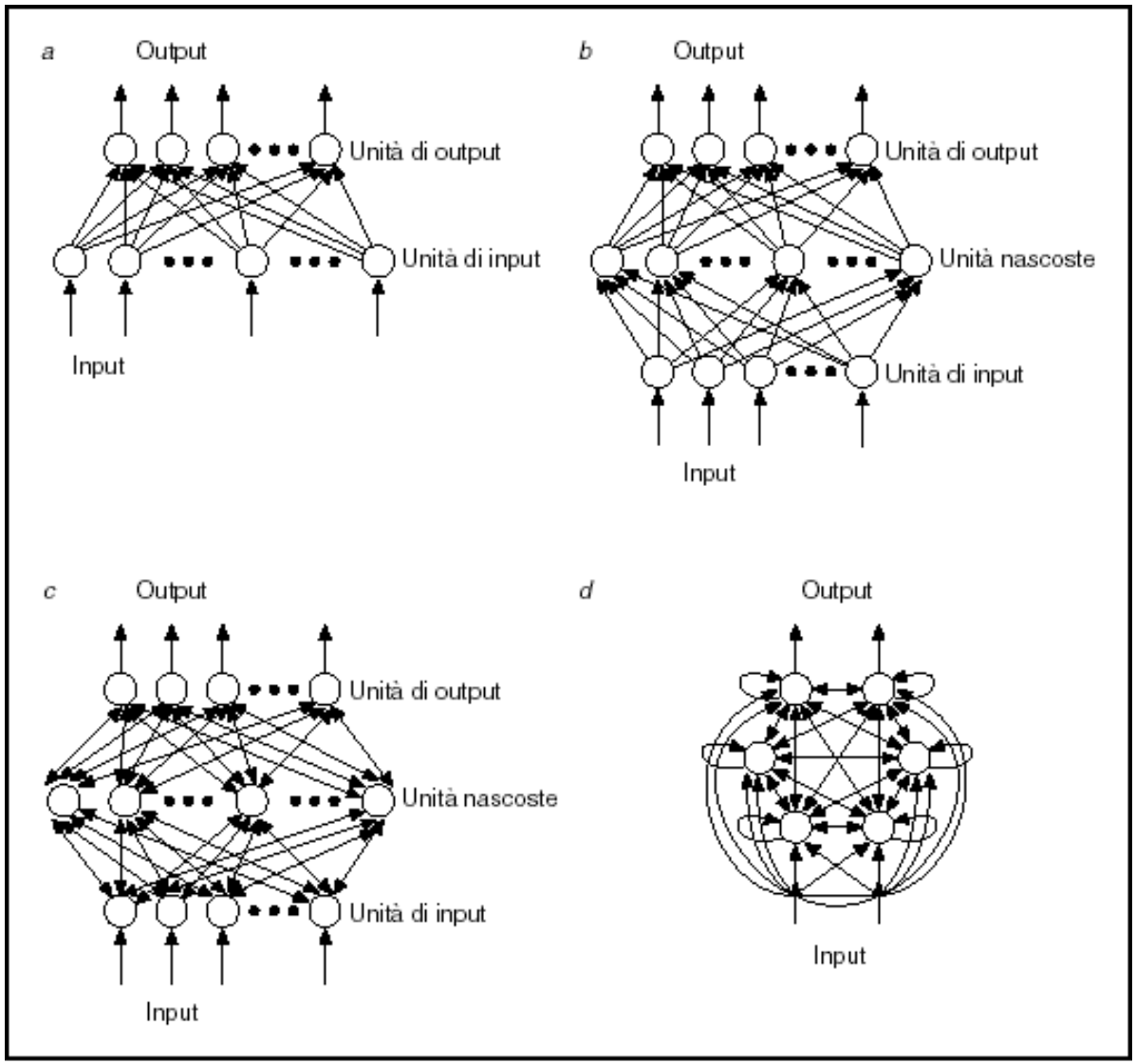



FIG. 11.3. Architetture di reti.

A. A *pattern associator*. There are only input and output units, with unidirectional links.

*B.* *Multi-layer feedforward* network. There are also one or more layers of hidden units. Unidirectional connections.

*C.* *Recurrent* network. Activation can spread backwards towards the hidden and input layers. Bidirectional links (feedback).

*D.* *Fully recurrent network* Like a recurrent network but there are also within-layer (lateral) connections.

# Biological networks vs. artificial networks

**Neurons**:

• organization of synaptic contacts

• distribution of excitatory vs. inhibitory synapses

• neuron dynamics (Hodgkin & Huxley, 1952)

**Networks**:

•number of neurons: few tens/thousands vs. 10^11; each neuron communicates on average with 10^4 other neurons

• laminar structure of the cortex

• types of neurons (e.g. excitatory vs. inhibitory)

• topographic maps

• columnar architecture

# Learning

•Learning in a network consists in finding the set of connection weights that allows the network to produce the appropriate response to a given input

– Hebb rule: if two linked neurons are simultaneously active, the synaptic efficacy (= connection weight) is strengthened:

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

– There are different types of learning and many different learning rules / learning algorithms

# Three Types of Learning

Imagine an organism (natural or artificial system) that experiences a series of sensory inputs: $x_1, x_2, x_3, x_4, \ldots$

**Supervised learning:** The system is also given desired outputs $y_1, y_2, , ..,$ and its goal is to learn to produce the correct output given a new input. Note that there is an external teacher.

**Unsupervised learning:** The goal of the system is to build representations from that can be used for reasoning, decision making, predicting things, communicating etc. Note that there is not a specific task.

**Reinforcement learning:** The system can also produce actions which affect the state of the world, and receives rewards (or punishments) $r_1, r_2, ..$ Its goal is to learn to act in a way that maximises rewards in the long term.

# Supervised learning:

During learning, a “teacher” provides information regarding the desired (correct) behaviour

- Supervised hebbian learning
- Delta rule
- Back-propagation
- Radial basis function

**Applications**: classification, function approximation, prediction

Note: Feedforward networks with nonlinear hidden units are universal approximators

**Relation to statistical methods**: general linear model (GLM), non-linear regression, discriminant analysis

# **Unsupervised learning**:

the network learns autonomously and discovers properties that are present in the training set. The type of property depends on the choice of the learning algorithm.

- Hebbian learning
- Competitive learning
- Kohonen networks (LVQ, SOM)
- Generative models

**Applications**: clustering, dimensionality reduction, data modeling

**Relation to statistical methods** : Cluster analysis, Principal Component Analysis (PCA)

# Common aspects of learning algorithms

•initial values of synaptic weights are *randomly* assigned (e.g. between -0.1 and +0.1) or set to *zero*

• repeated presentation of training pattern. *Supervised learning*: **input + target** (desired output). *Unsupervised learning*: **input** only

•learning consists in changing the synaptic weights, that is the computation of $\Delta \boldsymbol{w}$ with respect to the current value of $\boldsymbol{w}$. The update may take place after each pattern (*on-line learning*) or after each epoch (*batch learning*).

•To avoid the deletion of previously learned knowledge, only a fraction of the computed synaptic change must be used, which is defined through the constant $\eta$ - **learning rate**. Thus:

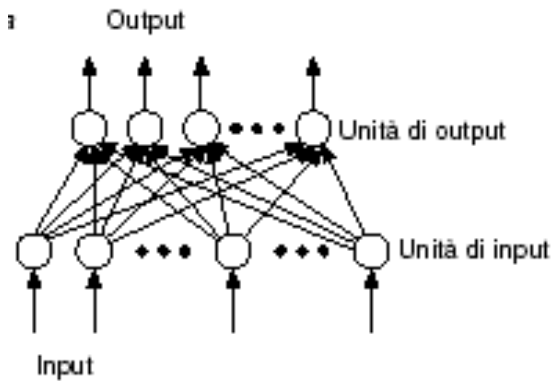
$$w_{ij}^{t} = w_{ij}^{t-1} + \eta \Delta w_{ij}^{t}$$

# Hebb rule

The most simple learning rule is Hebb's rule:

*If two connected neurons are simultaneously active, the synaptic efficacy is strengthened.* Formally, to associate an input patter **x** with some output **y**,

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




In the original formulation, Hebb's rule can only strengthen the connections. However, using bipolar neurons (rather than binary neurons) we obtain a more powerful learning rule, the **covariance rule** (Hopfield).

*Note: Hebbian rules are more often used in the context of unsupervised learning.*

# Perceptron rule

The network is trained using examples composed by an input vector **x** and a desired output **t** (target or “teaching input”). Both must use bipolar values (-1;1).

The network output for each input pattern is given by

$$y = \begin{cases} 1 \text{ se } \sum_{i=0}^{N} w_i x_i > 0 \\ -1 \text{ otherwise} \end{cases}$$

This value is compared with the desired output **t**. If there is error, the synaptic weights are changed according to the correct response:

$$\Delta w_i = \eta t x_i$$

# Delta rule

•similar to the perceptron rule, but can be used with output units that use a continuous and differentiable output function (e.g., sigmoid)

•allows to describe performance with a function that measures error – *error function or cost function* – which is based on the mean squared error between desired output and actual output:

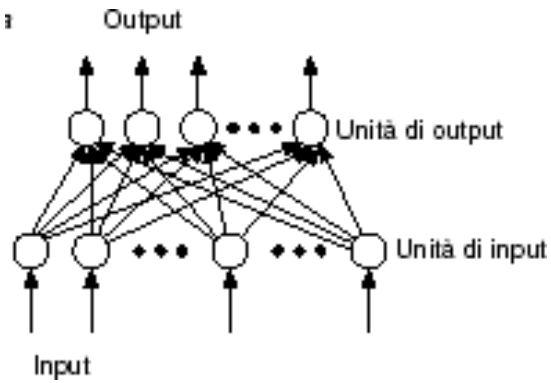
$$E_W = \frac{1}{2}\sum_{\mu}\sum_{i}(t_i^{\mu} - y_i^{\mu})^2$$

•learning consists in minimizing the cost function E, which depends uniquely on the values of the connection weights W. Thus, weights are modified in a direction opposite to that of the gradient of the cost function (*gradient descent*):

$$\Delta w_{ij} = -\frac{\partial E}{\partial w_{ij}}$$

•By solving the equation, we get that the weight changes are given by the difference between target and output multiplied by the presynaptic activity:

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



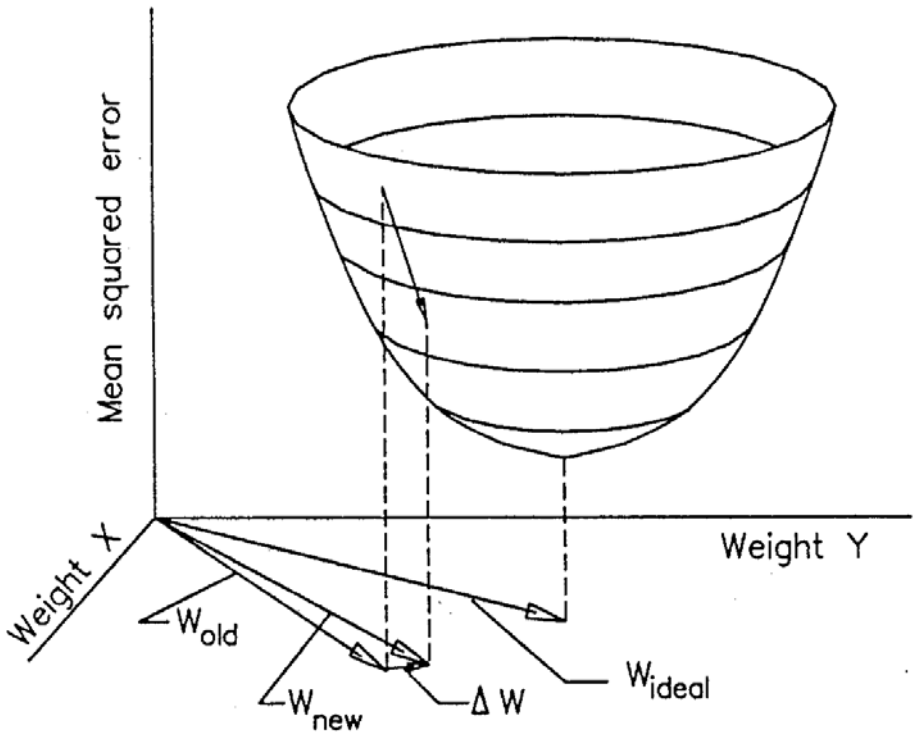




Figure 6.1 The delta rule moves the weight vector so that it follows the steepest descent of the hyperparaboloid.

*Note: delta rule is biologically plausible and it has been formally shown to correspond to the Rescorla-Wagner rule in classical conditioning.*

# Back-propagation

Multi-layer networks (one or more intermediate layers with nonlinear units) are necessary to solve problems that are not linearly separable (e.g. XOR)

•Back-propagation is an extension of the delta rule (generalized delta rule) that allows learning in multi-layer networks

•It solves the problem of how to compute errors for the hidden units: backward propagation of errors (*error back-propagation*) using the same connections that spread activation forward.

• Error back-propagation is biologically implausible.

# Generalization

- Ability to appropriately exploit existing knowledge of the domain when encountering new examples of the problem
- Necessary (but not sufficient) conditions to obtain good generalization performance in neural nets:
  - Input variables should contain sufficient information related to the target; that is, input and correct output should be linked by a (possibly unknown) mathematical function with some degree of accuracy
  - The training set should contain a sufficient number of training examples that constitute a representative sample of the population
- For any underlying mapping function, it is useful to distinguish between interpolation (usually possible) and extrapolation (difficult and sometimes impossible)

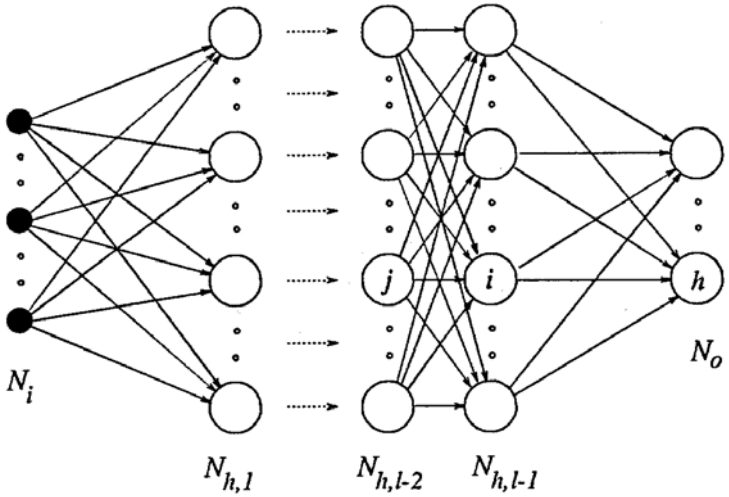



Figure 4.1: A multi-layer network with $l$ layers of units.

Multi-layer networks trained with error gradient descent can learn any arbitrary function (task)

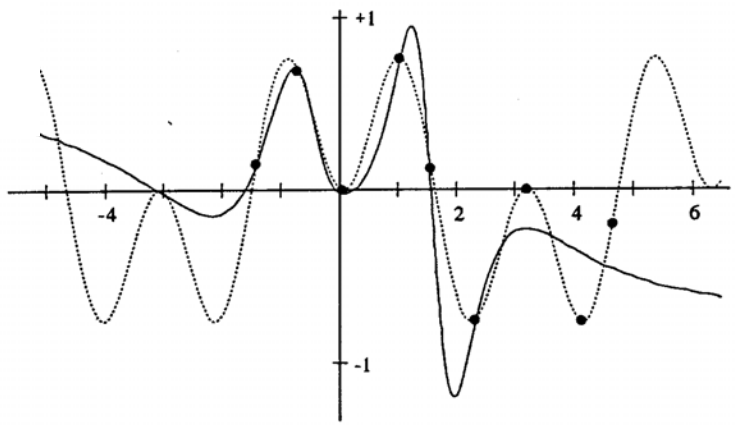



Figure 4.4: The periodic function $f(x) = \sin(2x)\sin(x)$ approximated with sigmoid activation functions.

The starting point is Hebb's rule:

$$\Delta w_j = \eta y x_j$$

Where activation of the single output unit is the weighted sum of the input units' activities. There is no desired output.

If a component of the input vector activates y, the strength of the synapse will be increased and on the next presentation it will be further strengthened (self-amplification). Thus, this simple network will learn to respond more strongly to the patterns that contain the most frequent components.

Problem: weights will grow unbounded towards infinite. We can therefore normalize the synaptic values – the length of the weight vector must be of 1

$$w_j = \frac{w_j}{\sqrt{w_1^2 + w_2^2 + .. + w_n^2}}$$

In this case, the network converges towards the solution in which the output expresses the direction of maximal variance in the distribution of the input. Thus, patterns that are aligned in this direction will produce a stronger response.

If the distribution of input patterns is centred on zero, the output represents the first principal component of the distribution.

Note: this is not a local rule! (we need to know all weights)

# Oja's rule and Sanger's rule

**Oja**: adds to Hebb's rule a "forgetting factor" that bounds growing of the synaptic value as a function of its current value:

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

The rule can be further improved to be used with N output units, by subtracting from the activation of the presynaptic unit the weighted activation of all N output units linked to the same input unit:

$$\Delta w_{ij} = \eta y_i \left( x_j - \sum_{k=1}^{N} w_{kj} y_k \right)$$

In this case the network will extract N principal components

**Sanger**: the component to be subtracted is computed on all units before unit i. Thus the sum over k goes from 1 to i instead of from 1 to N.

Therefore, extraction of the N principal components is ordered. Each unit is forced to develop a weight vector that is orthogonal to the vectors of the preceding units and at the same time to maximize the residual variance.

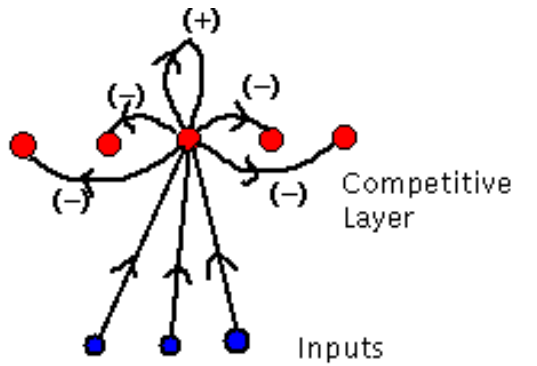
Pluses: interpretation of the output is easy.

Minuses: introduces distortions if N is large.

# Competitive networks

A competitive network is formed by a group of units in which each neuron receives activation from the input layer and there are also lateral (intra-layer) connections. Each neuron is connected with itself with an excitatory weight (self-feedback) and inhibits all the other neurons in the layer with inhibitory weights.

Activation is therefore given by the combination of input signals from the preceding layer and the weighted sum of activity of the surrounding units.




(+/–) indicates negative/positive weight

For a given input I, some neuron k will have higher activation than any other neuron in the layer. If activation is allowed to dynamically change over time (competitive dynamics), k will develop a maximum activation values whereas the other neurons will be “shut down” (winner-takes-all).

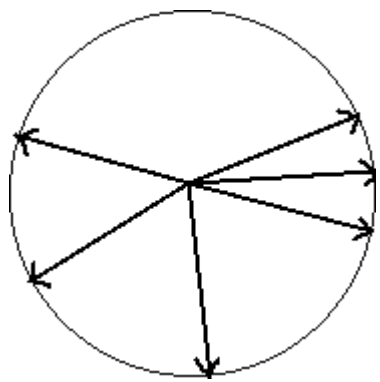
# Competitive learning

Consider a training set in which all vectors have the same length, e.g., 1 (that is, they are normalized).

Note: length of a vector: $\|x\| = \sum_i x_i^2$

If the components are all positive or zero, this is approximately equivalent to the condition: $\sum_i x_i = 1$

Given that all vectors have length 1, they can be represented by arrows from the origin towards the surface of a (hyper)sphere of radius 1.



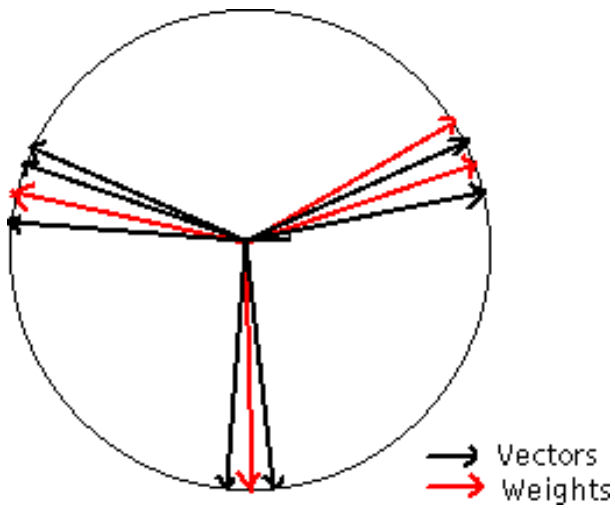
Suppose that the competitive layer has normalized weight vectors. Thus, these vectors can be represented on the same sphere.

To encode the training set, it is necessary that the weight vectors in the networks become aligned with any cluster of input vectors in the training set and that every cluster will be represented by at least one neuron.

Thus, when a input vector is presented to the network, there will be one node (or one group on units) maximally (and selectively) responding to that input.

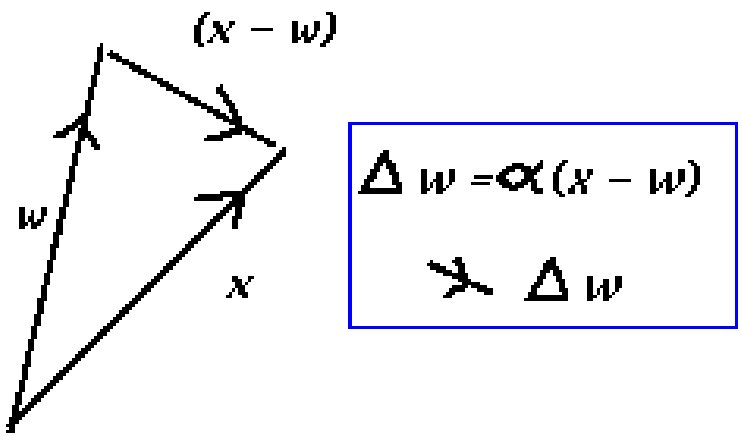
**Alignment of weights**

**and input vectors**



To achieve this result, weight vectors must be rotated around the sphere so that they will align with the training vectors. This can be obtained in a gradual and efficient way by moving the weight vector that is closest (in angular distance) to the current input vector towards the input vector itself. The neuron with the closest weight vector is simply the *k* neuron that receives the strongest excitation from the input. To align the weight vector of k, we can use the following learning rule:

$$\Delta w = \eta(x - w)$$



whole network:

$$\Delta w = \eta(x - w)y$$

Thus the synaptic change will take place only for the winner (y=1), whereas for all other neurons there will be no weight change (y=0).

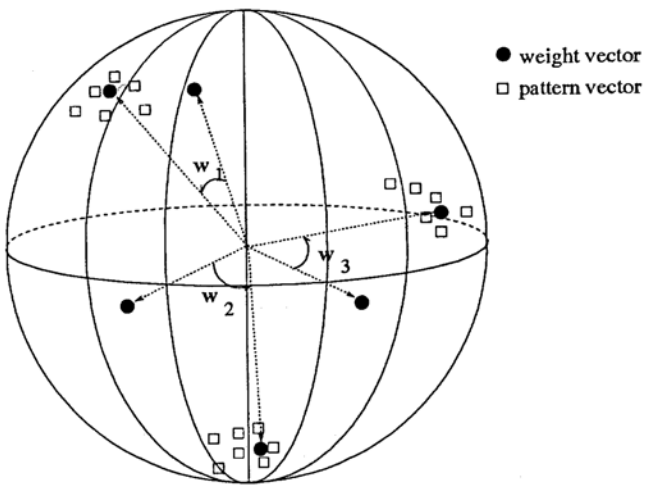
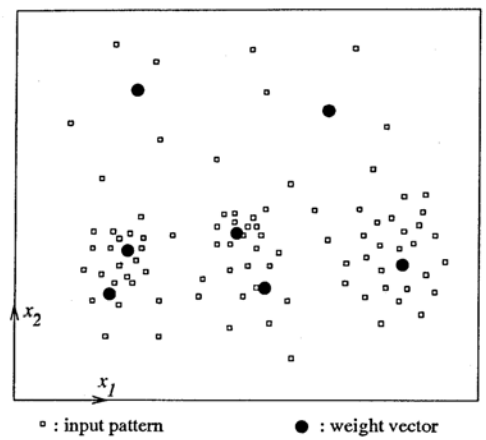
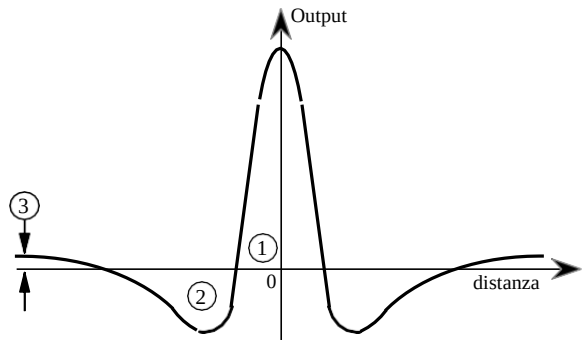



Figure 5.2: Example of clustering in 3D with normalised vectors, which all lie on the unity sphere. The three weight vectors are rotated towards the centres of gravity of the three different input clusters.




.4: This figure visualises the tracking of the input density. The input patterns are om $\mathbb{R}^2$; the weight vectors also lie in $\mathbb{R}^2$. In the areas where inputs are scarce, the rt of the figure, only few (in this case two) neurons are used to discretise the input Thus, the upper part of the input space is divided into two large separate regions. er part, however, where many more inputs have occurred, five neurons discretise t space into five smaller subspaces.

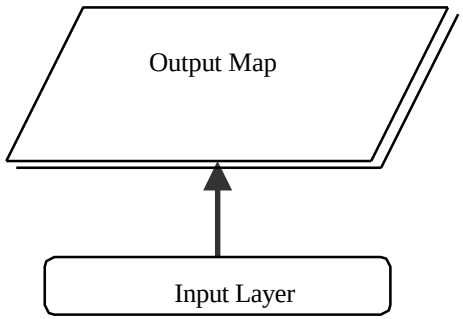
# Self-organizing maps (SOM)

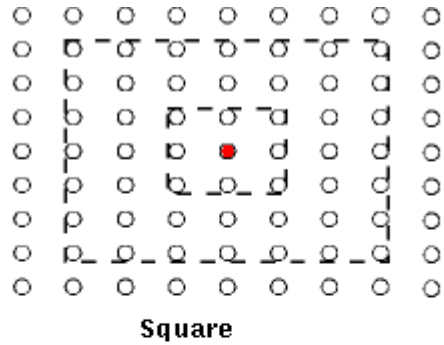



In biological systems, lateral connections between neurons can be both excitatory and inhibitory, in relation to the lateral distance.

- (1) narrow excitatory region
- (2) inhibitory region
- (3) far region of weak excitation (typically ignored)

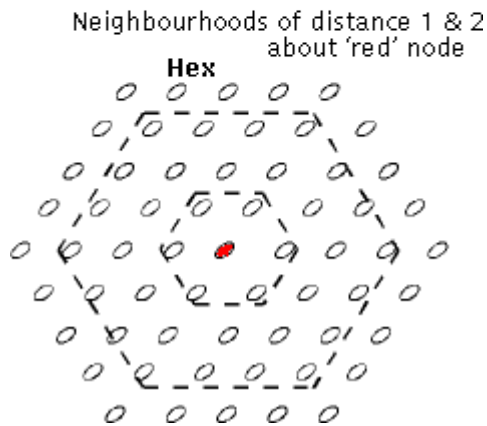
# SOM networks




SOM networks (Kohonen), consist of a bidimensional layer (“sheet”) of neurons. Every unit is connected with all input units and with the other neurons of the same layer according to a neighbourhood function that defines its “neighbours”.

This is a computational shortcut to avoid using time-expensive competitive dynamics.



Weight update is given by:

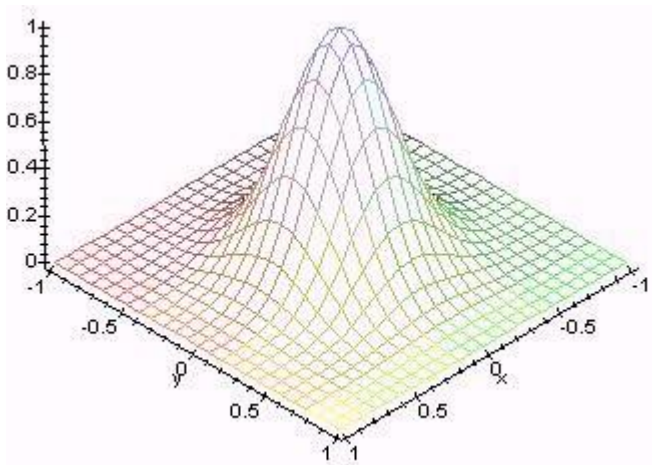
$$\Delta w_j = \eta(x - w_j) \qquad j \in \Lambda_{i(x)}$$

$\Lambda_{I(x)}$ is the neighbourhood function centred on the winning neuron *i(x)*. Both $\Lambda_{I(x)}$ and $\eta$ are dynamically changed during learning. $\eta$ usually varies according to an exponential function.

The neighbourhood function has a bubble shape that includes all neighbours in a region around the winner. Using a Gaussian, the neighbourhood function decreases with increasing lateral distance:

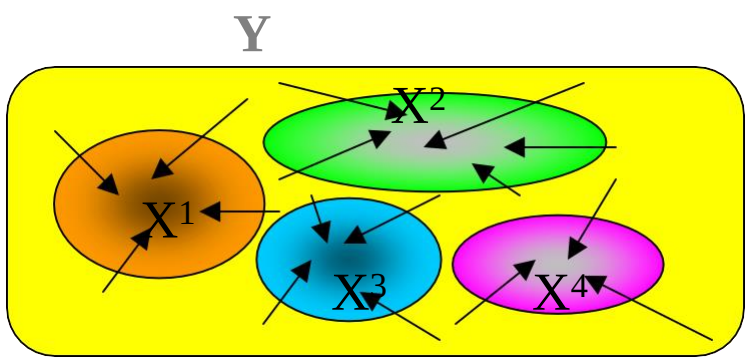
$$\pi_{ij} = e^{-\frac{d_{ij}}{2\sigma}}$$

Where $d_{ij}$ is the lateral distance of neuron *j* from the winner *i* and $\sigma$ is the parameter that controls the width of the Gaussian. The function controlling $\sigma$ is exponential



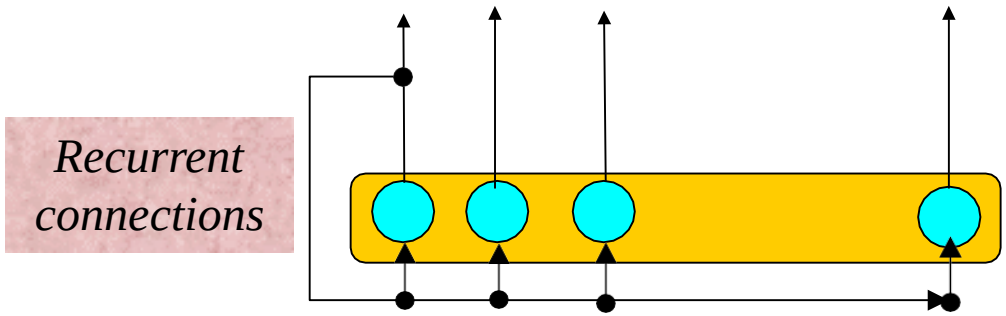
# Neural networks for associative memory

- Hopfield developed a fully recurrent neural network to solve the problem of storing a set of data $\{X^i\}$ in a way that the nework, if presented with a new pattern Y, will retrieve the pattern $X^i$ which is most similar to Y (**associative memory**);
- In ***associative memory networks***, data are encoded, on the basis of their content, as **attractors** that will draw any new pattern towards their "**attractor basins**". Note: these models are inspired by statistical mechanics



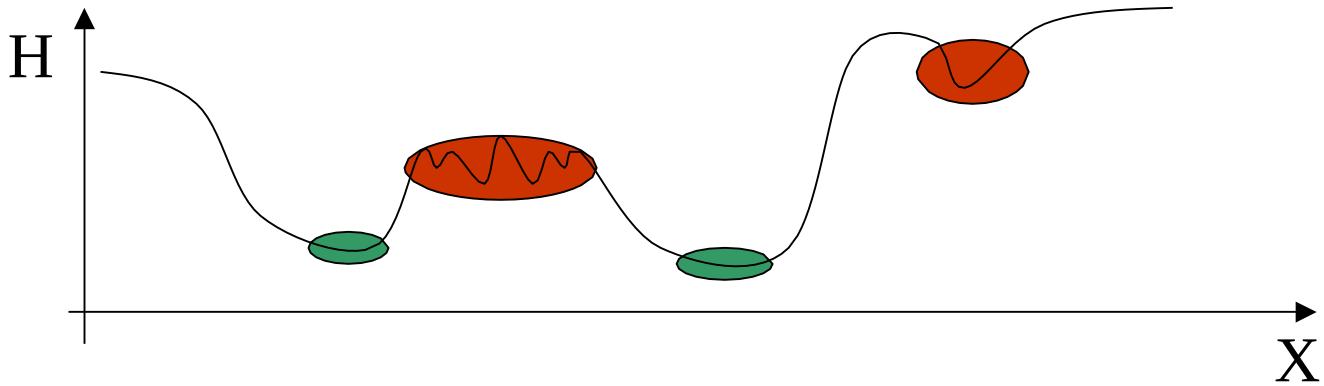

**State space X**

# Hopfield network




- The network is allowed to “cycle” until equilibrium after presenting a single pattern
- If the weights have appropriate values, the network state will proceed towards points of stability (attractors), where neurons stop changing their activation states (*equilibrium*).
- Learning is based on Hebb’s rule (covariance rule)

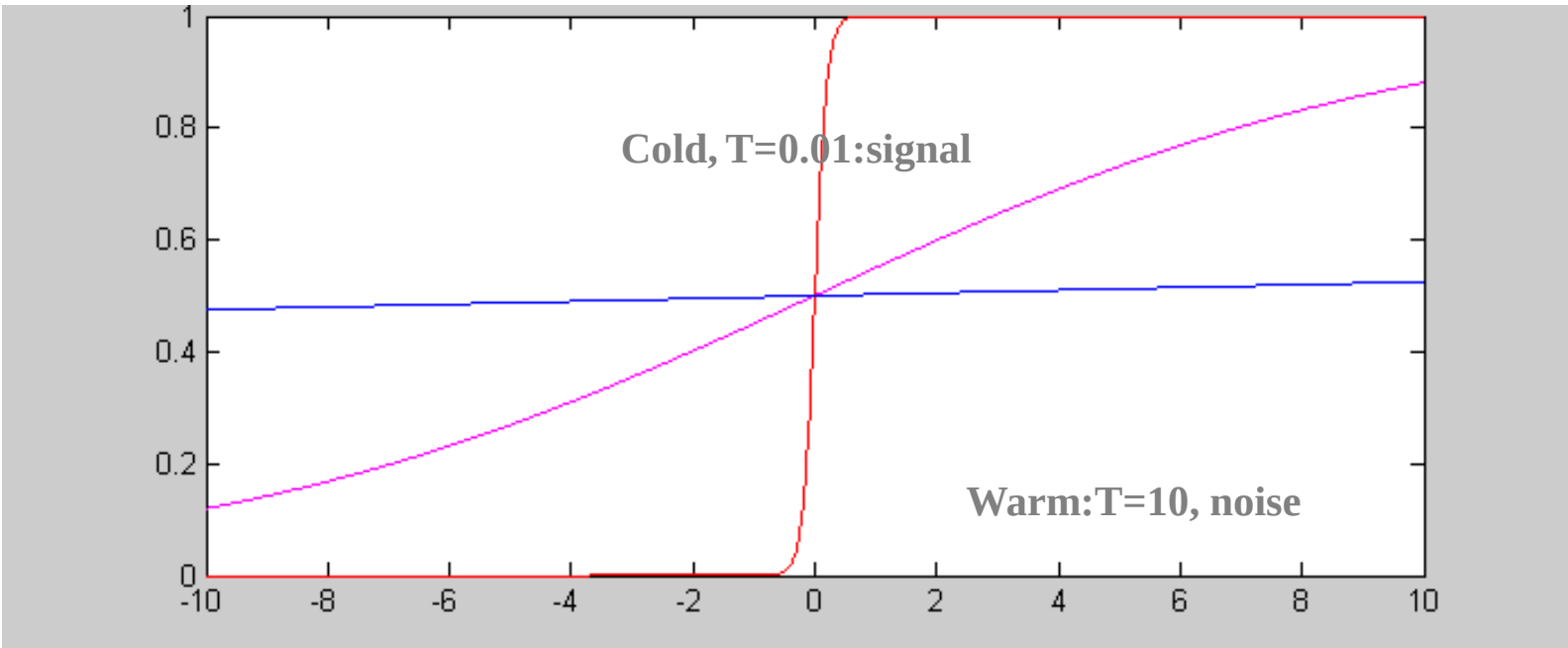
- In Hopfield nets, spurious attractors have low energy states, that is **local minima**.




- Local minima do not allow to reach **global minima** when the network is at equilibrium
(global minima represent real patterns).

## Stochastic dynamics and simulated annealing

- If the temperature il low (β large) is very likely that the neuron is active if the input signal is positive
- If the temperature is high (β small), the neuron will produce noise, instead of being driven by the input signal




$$P(x_i = 1) = \frac{1}{1 + e^{-\beta \sum_j w_{ij} x_j}} \qquad \beta = 1/T$$

# Generative models

- probabilistic models of how the underlying physical properties of the “world” cause the sensory data (input)
- provide one “objective function” for unsupervised learning
- learning can be viewed as maximazing the likelihood of the observed data under the generative model, which is equivalent of discovering efficient ways of coding sensory data (Gharahami, Korenberg, & Hinton, 1999)
- plausible models of cortical learning, offer a strong hypothesis of the role of top-down (feedback) and lateral connections in the cortex

- breakthrough in unsupervised learning: generative probabilistic models that discover representations both non-linear and distributed (Hinton & Gharahami, 1997, RGBN; Hinton, 1999, PoE). Previously considered as computationally intractable.

These techniques can be applied to fully recurrent networks (e.g., *Boltzmann Machines*)

→ contrastive divergence learning (Hinton, 2000)
(Mean Field version: Welling & Hinton, 2001)

- Visible layer: data (can be divided into input and output groups)
- Hidden layer: internal distributed representations
- Recurrent architecture: bidirectional connections between layers (bottom-up and top-down) and within layers (lateral connections). Many variants are allowed (restricted BMs).