Neural networks

- Until now we have seen symbolic reasoning approaches based on symbols and syntactic rules for their manipulation.
- Connectionists believe that the symbolic manipulation is a very poor mechanism.
- The connectionist approach is based on simulation of the mechanisms in the human brain
- Models resembling neurological structures have therefore been developed.

The dilemma of IA

Up to some years ago, computers were excellent in computing, but failed when trying to play typically human activities:

- Sensor perception
- Sensor-motion coordination
- Image recognition
- Adaptability

With the advent of deep neural networks the situation has drastically changed but still…..

Child beats Computer 3 to 0

Although a computer can beat the world chess champion, it is not able to compete with a 3 year old child in

- Build a Lego car
- Recognize the face or the voice of a person

Problem

- These complex actions depend on many factors, which cannot be precisely predicted by a program.
- These factors have to be acquired with the experience, in a learning phase.

Examples

The success of gripping an object is determined by several factors:

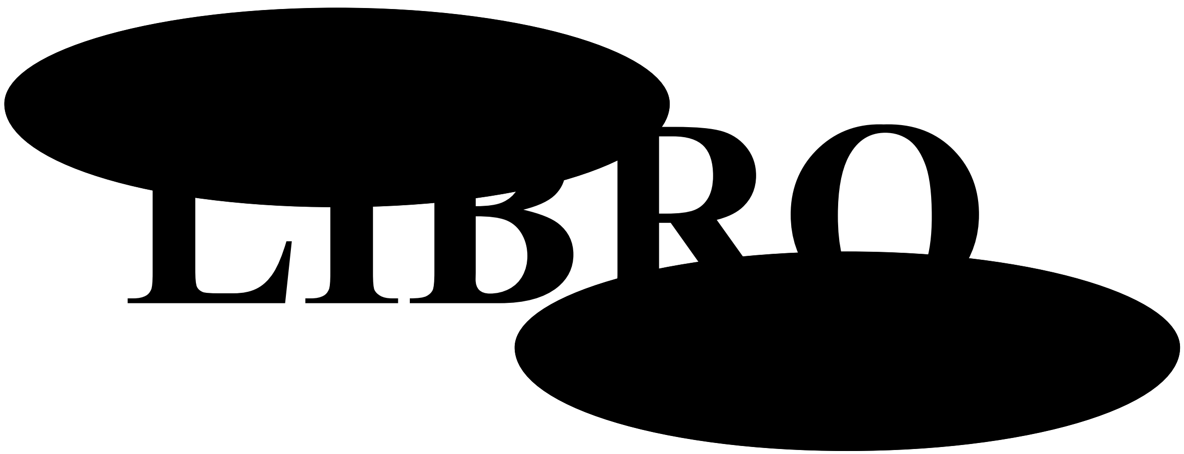
- the object position
- our posture
- the size and shape of the object
- the expected weight
- any interposed obstacles

Speech recognition

It requires a learning phase necessary to:

- adapt to the person who speaks
- filter out external noise
- separate any other items

Image recognition



How does the brain work?

- When we recognize a face or grasp an object we do not solve equations.
- The brain works in an **associative** fashion

Each sensory state evokes a brain state (electro-chemical activity) that is stored according to need.

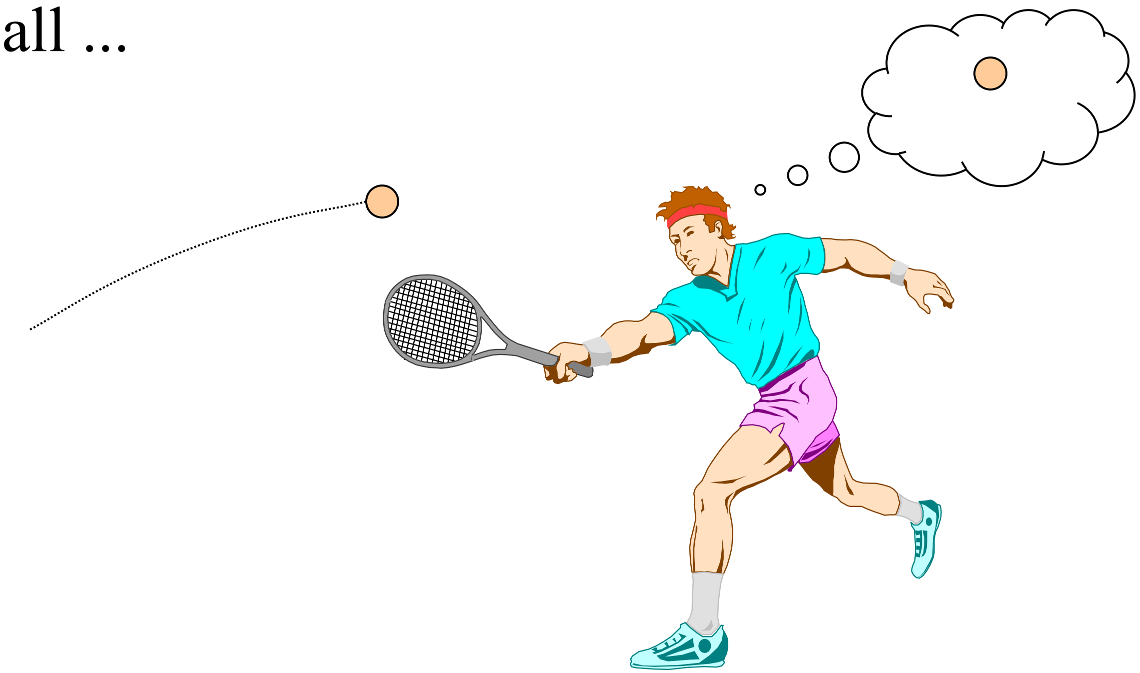
Hitting a tennis ball

- The trajectory depends on several factors:
  - stepping force, initial angle, effect, wind speed;
- Forecasting trajectory requires:
  - the precise measurement of the variables;
  - the simultaneous solution of complex equations, to be recomputed at each data acquisition.

How does a player do that?

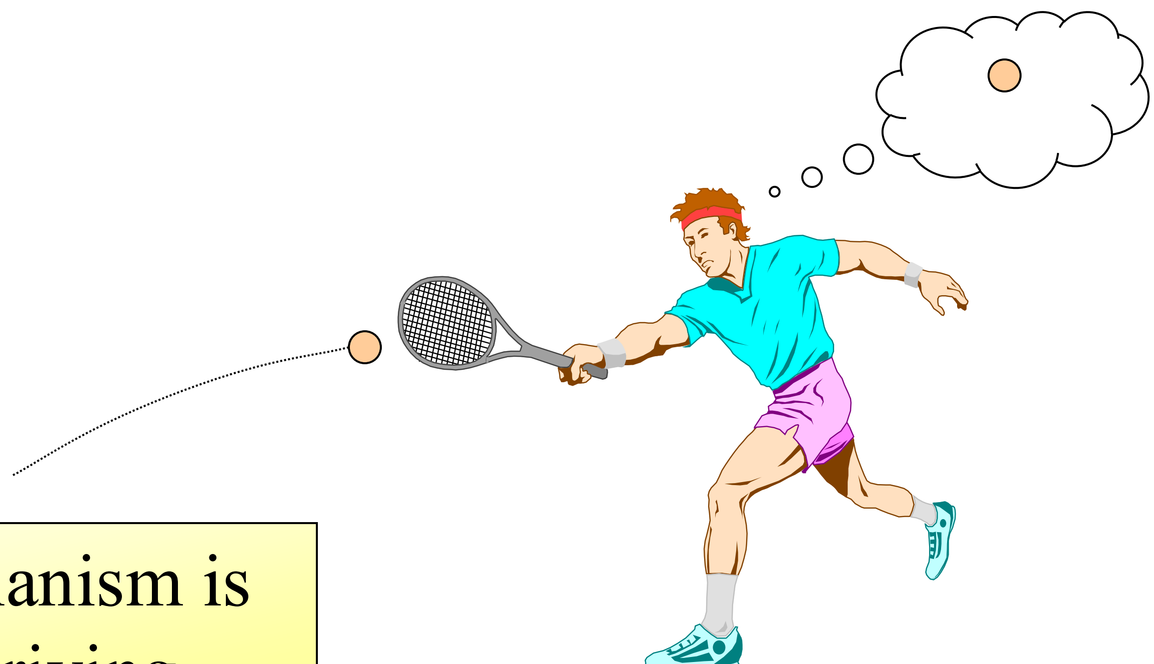
Learning phase

- In a learning phase the player tries actions and records the good ones:
  - If the ball is passed in this visual field area, take a step back;
  - if the ball ...



Operational phase

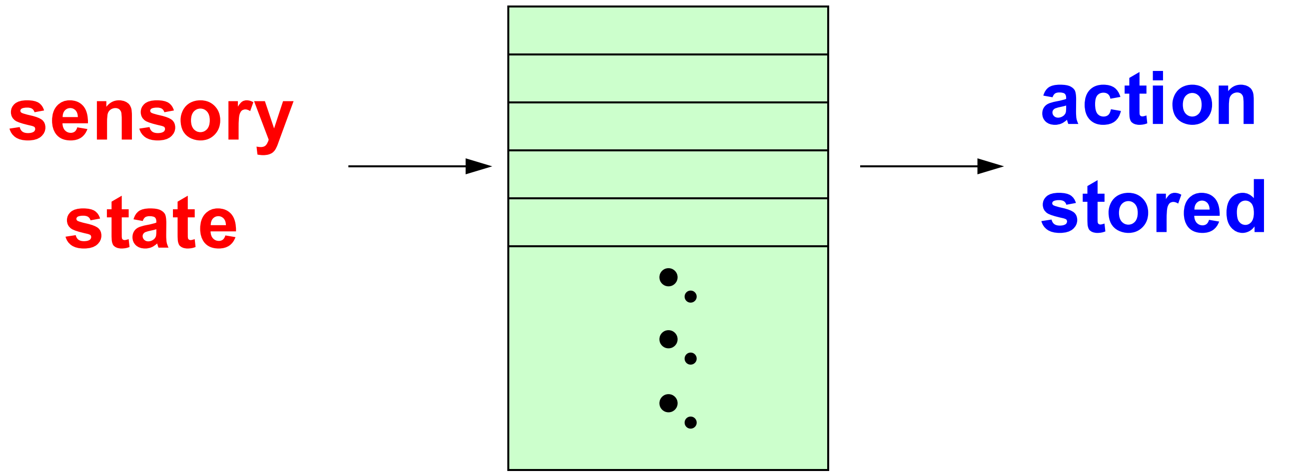
- Once trained, the brain performs actions *without thinking*, on the basis of learned associations.



A similar mechanism is used while driving

The associative calculation

- A set of complex equations are solved by means of a **look-up table**.
- It is built on the basis of experience and is refined during training.



The neural computing

It is extremely difficult to treat these problems with a computer. There is need to study new methods of computation, inspired by the neuronal networks.

Neurophysiologists → study the brain

Engineers → implement the code

Historical hints

- **1943 - McCulloch and Pitts:** defined the first binary threshold neuron model

- **1949 - Hebb:** from studies on the brain, he showed that learning is not a neuron property, but it is due to a modification of synapses.

- **1962 - Rosenblatt:** he proposes a neuron model that can learn by examples: the **perceptron**.

- **1969** - **Minsky and Papert** showed the limitations of the perceptron: diminished enthusiasm on neural networks.

Historical hints

- **1982 – Hopfield** proposed a network model to create associative memories.
- **1982 – Kohonen** proposed a type of self-organizing network (receptive maps).
- **1985 - Rumelhart, Hinton and Williams:** formalize supervised learning (Back-Propagation).
- **2006 - Yoshua Bengio** deep networks

Some properties of the brain

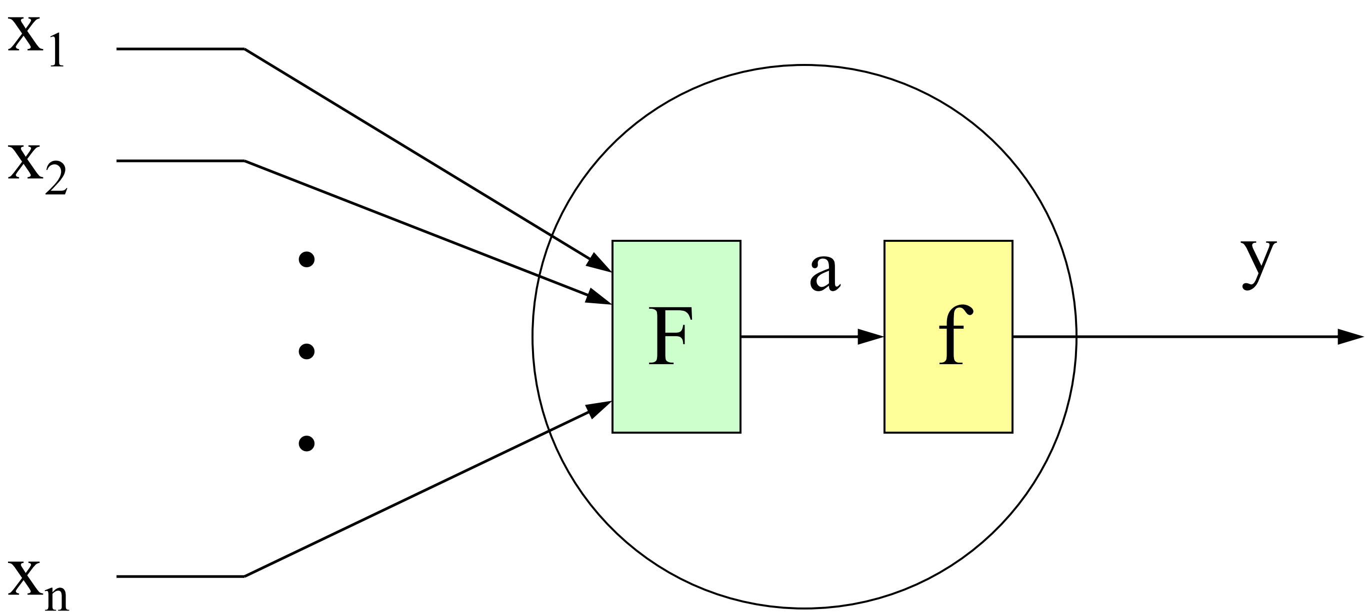
- **Speed of neurons:** few ms
- **Number of neurons:** $10^{11} \div 10^{12}$
- **Connections:** $10^{3} \div 10^{4}$ per neuron
- **Operations:** activation / inhibition
- **Distributed control:** lacks of a CPU
- **Fault tolerance:** graceful degradation

Neural model

To model a neuron we have to define:

- the number of input channels: **N**
- the type of input signals: $\mathbf{x_i}$
- the connection weights: $\mathbf{w_i}$
- the activation function: **F**
- the output function: **f**

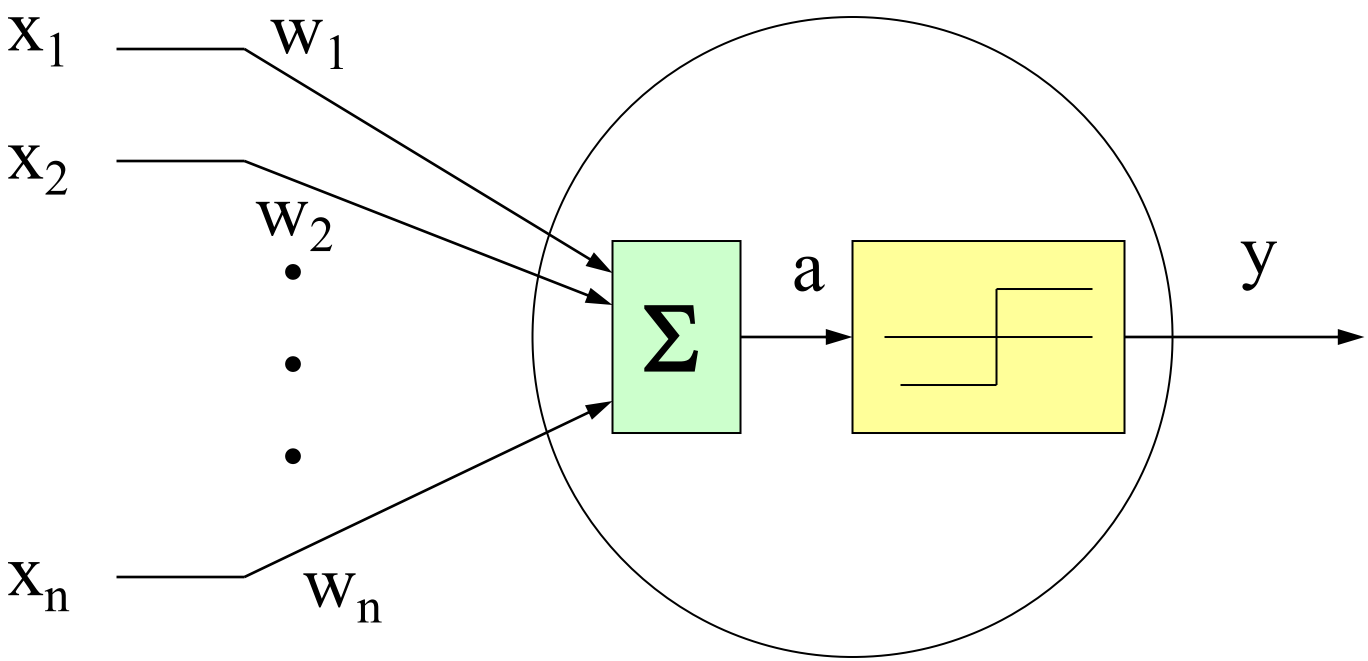
General neuron model



$$a(t) = F(x_1, x_2, \ldots, x_n)$$

$$y(t) = f(a)$$

The neuron threshold Binary

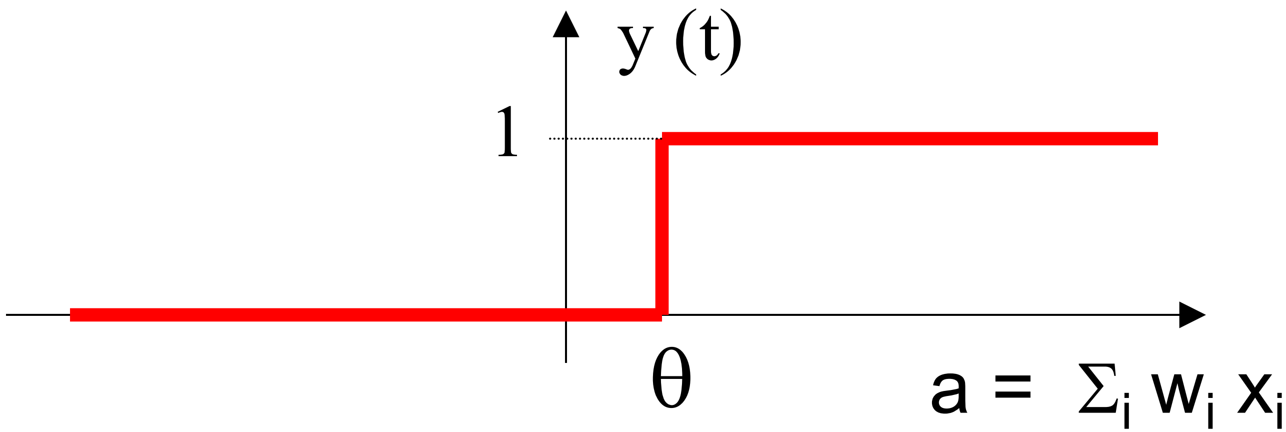


$$a = \Sigma_i \, w_i \, x_i$$

$$y = HS\,(a - \theta)$$

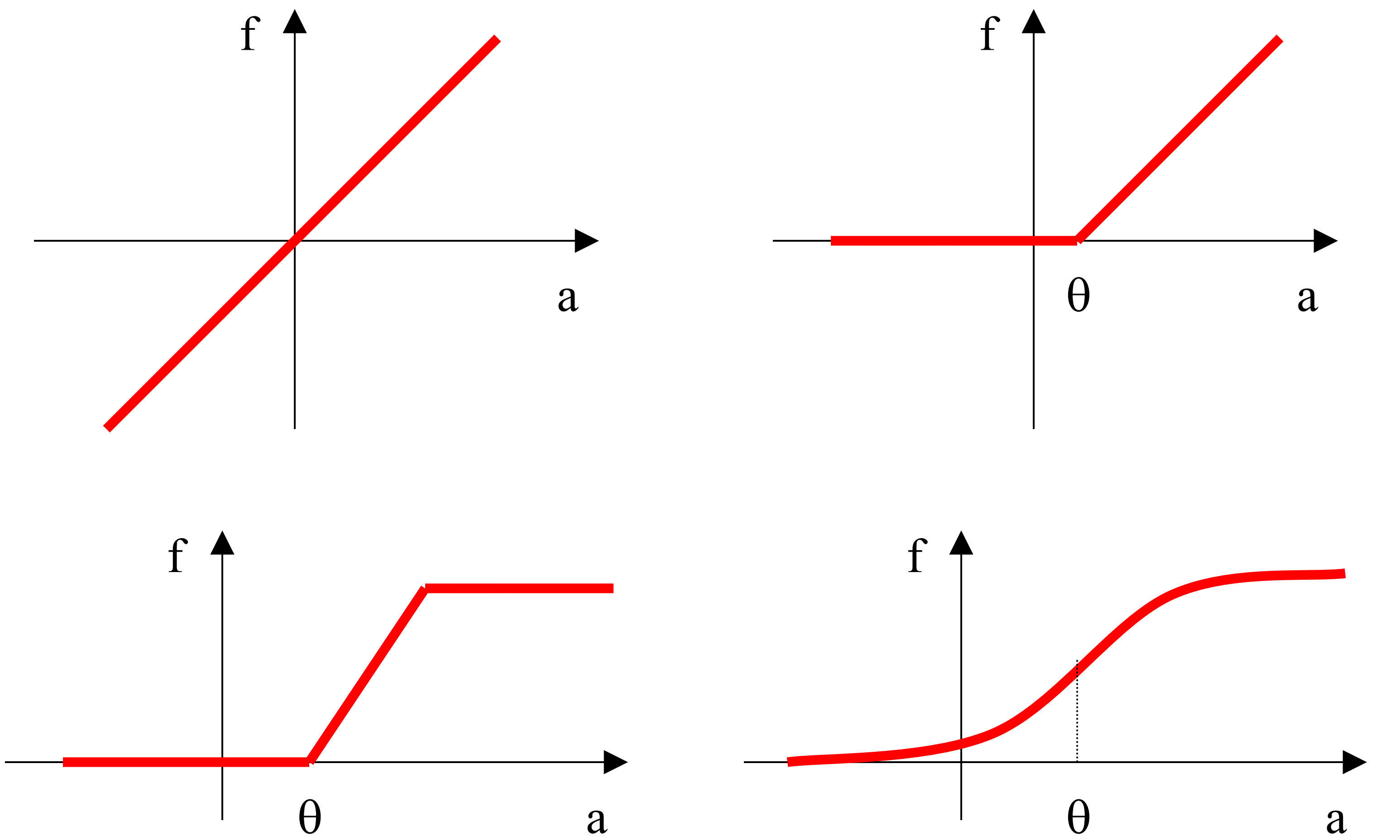
Neuronal functioning:
the pulses received from the dendrites increase the electric potential in the neuron up to a certain threshold

Heaviside function



$$y(t) = \begin{cases} 0 & \text{se } \sum_i w_i \, x_i < \theta \\ 1 & \text{otherwise} \end{cases}$$

Other output functions

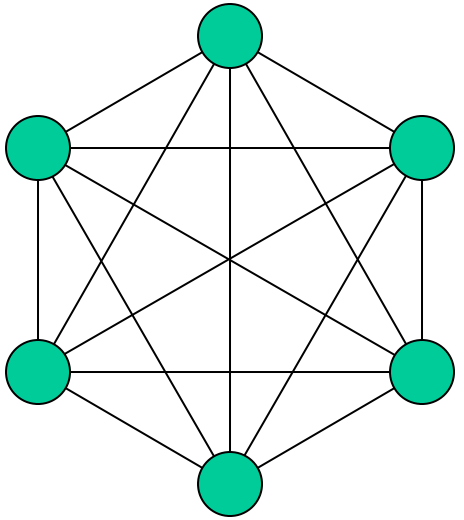


Neural networks

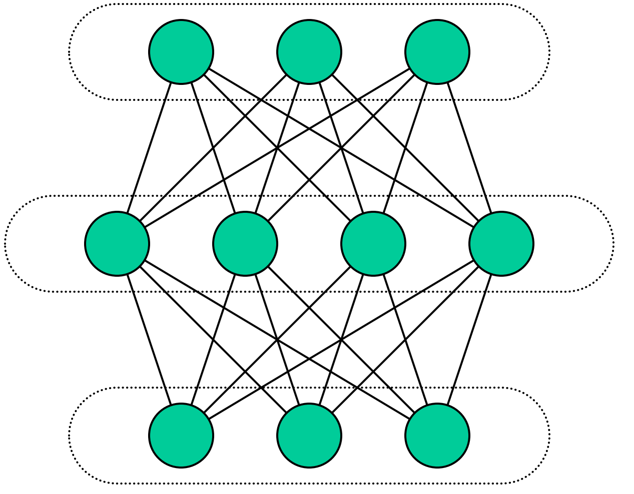
To build a neural network we have to define:

- The neuron model
- The network architecture
- The neuron activation mode
- The learning paradigm
- The learning law

Network Architectures

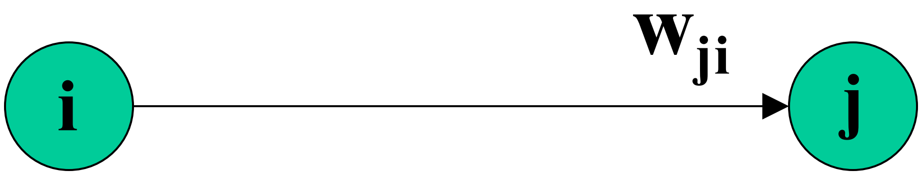


Fully connected



Multi-layer

Connections Representation

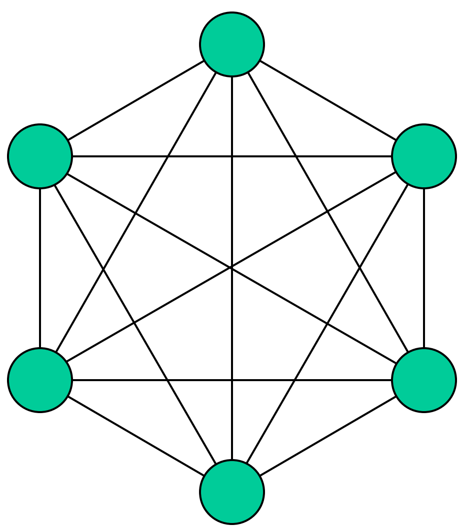


weight on neuron j on the connection coming from neuron i

Fully connected networks

They represent states that evolve over time

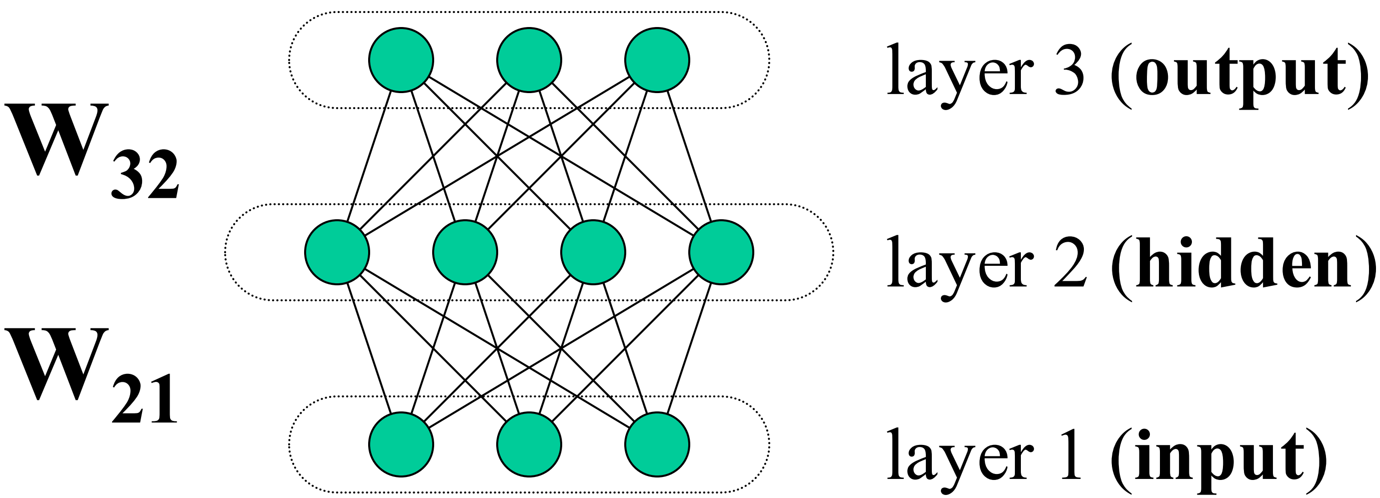
The weights of the network can be specified through a **connection matrix**



$$\mathbf{W} = \begin{pmatrix} \text{neuron 1 weights} \\ \text{neuron 2 weights} \\ \vdots \\ \text{neuron n weights} \end{pmatrix}$$

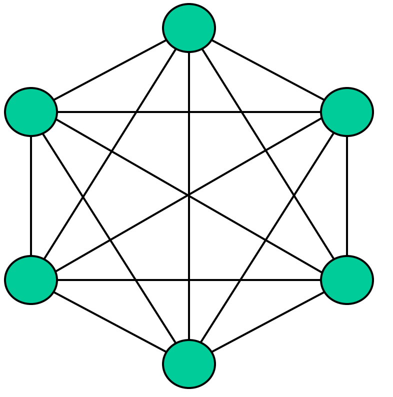
Layered networks

The weights of a network composed by **n** layers can be specified through **n–1 connection matrices:**



Fully connected network

- Binary neurons with threshold
- Parallel activation



State transition

$$\mathbf{x_i(t + 1) = HS\ [\Sigma_i\ w_i\ x_i(t)]}$$

$$x_i(t+1) = \begin{cases} 1 \text{ if } \ \Sigma_i w_i x_i(t) \geq 0 \\ 0 \text{ otherwise} \end{cases}$$

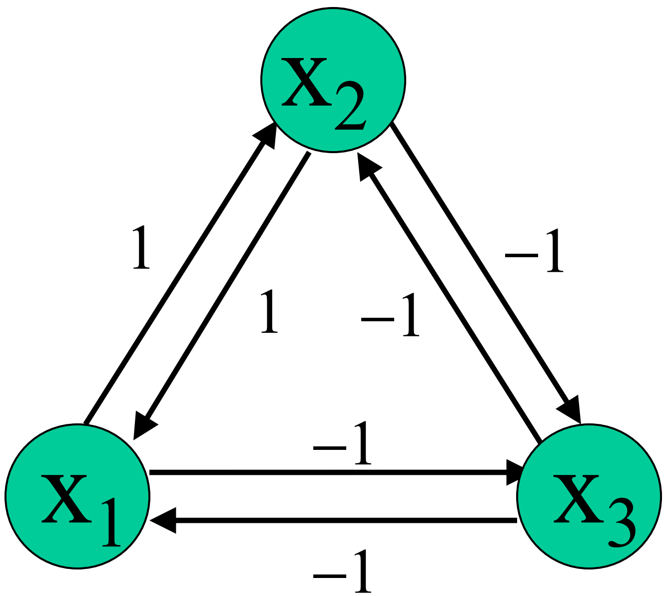
Evolution equation

In matrix form:

$$\mathbf{X\ (t + 1) = HS\ [W\ X\ (t)]}$$

- X (t) is the state of the network at time t
- W is the weight matrix

Example



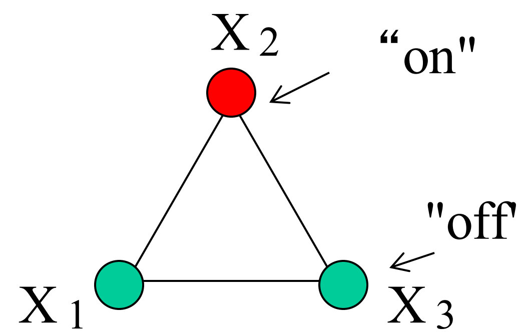
symmetric matrix

$$\mathbf{W} = \begin{pmatrix} 0 & 1 & -1 \\ 1 & 0 & -1 \\ -1 & -1 & 0 \end{pmatrix}$$

no self connections: 0 on the diagonal

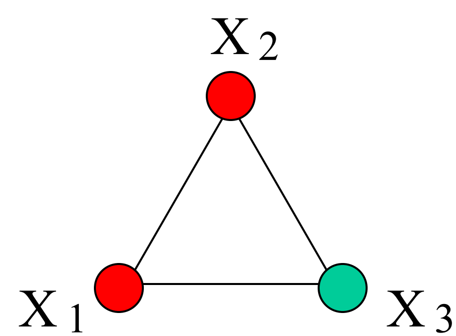
State Transition

Initial state: $$\mathbf{X\,(t)} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

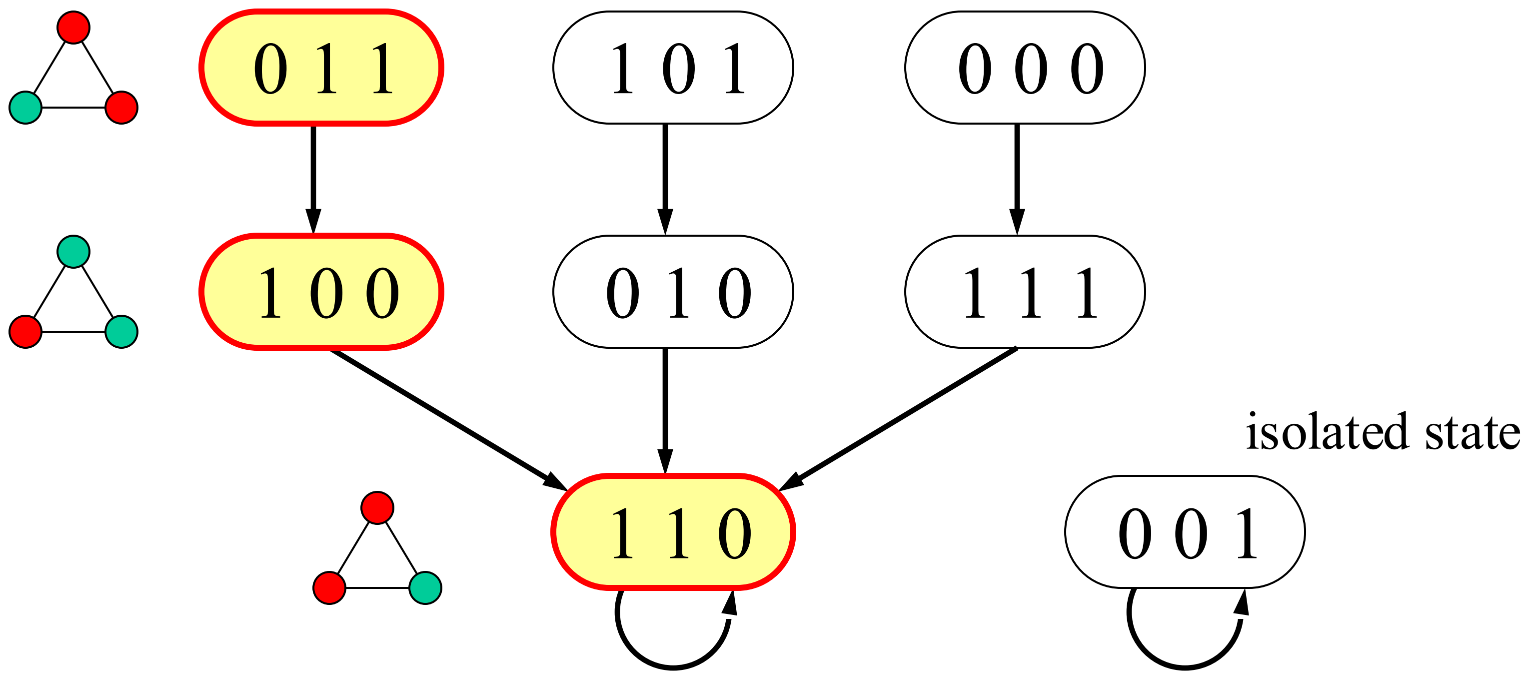


next state: $$\mathbf{X\,(t+1) = HS\,[W\,X\,(t)] =}$$

$$= \mathbf{HS} \left[\begin{pmatrix} 0 & 1 & -1 \\ 1 & 0 & -1 \\ -1 & -1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \right] = \begin{pmatrix} \mathbf{1} \\ \mathbf{1} \\ \mathbf{0} \end{pmatrix}$$

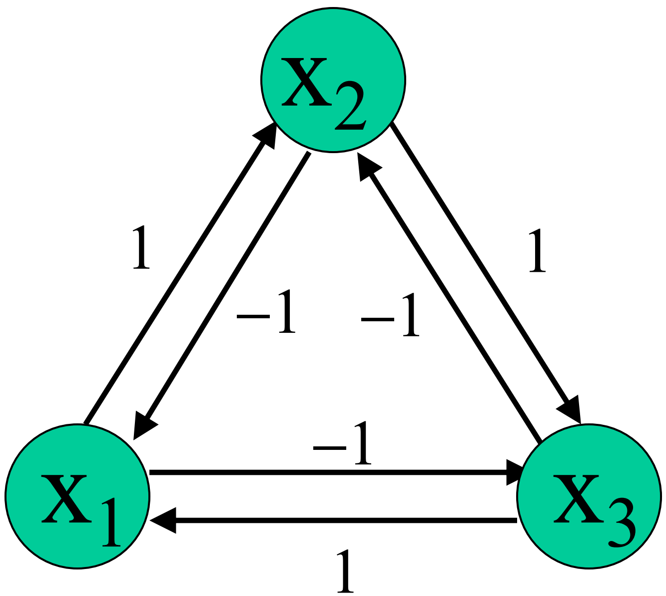


Transition Diagram



The network follows a trajectory up to stable states

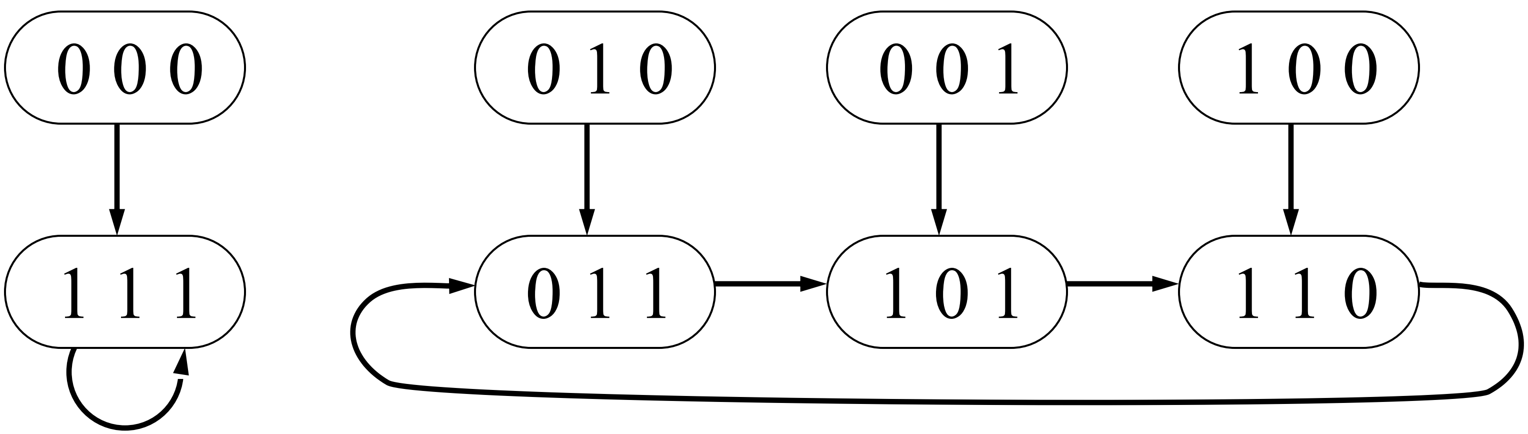
Example



antisymmetric matrix

$$\mathbf{W} = \begin{pmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{pmatrix}$$

Transition Diagram



Definitions

- Transformation

  Function T: $S \rightarrow S$, which transforms a state X(t) in the following X(t + 1).

- Trajectory

  Sequence of states traversed by the network, starting from an initial state X_0:

  $$X(0) = X_0$$

  $$X(t+1) = T[X(t)]$$

Definitions

- Limit cycle of order k

  Closed trajectory in phase space traversing X_i every k steps.

- Stable state

  State that generates a constant trajectory:

  $$X(t + 1) = X(t) = X_s$$

Definitions

- Reachable state

  A state X_F is said to be reachable from X_i if there exists a trajectory from X_i to X_F.

- Global stability

  A network is said globally stable if for every initial state X, the trajectory from X reaches a stable state.

Stability Properties

(Hopfield '82)

A fully connected neural network is globally stable if:

- the matrix of weights is <u>symmetric</u>
- the activation is <u>asynchronous</u>

Activation mode

- Synchronous (parallel)

  The neurons change their state all together, synchronized by a clock.

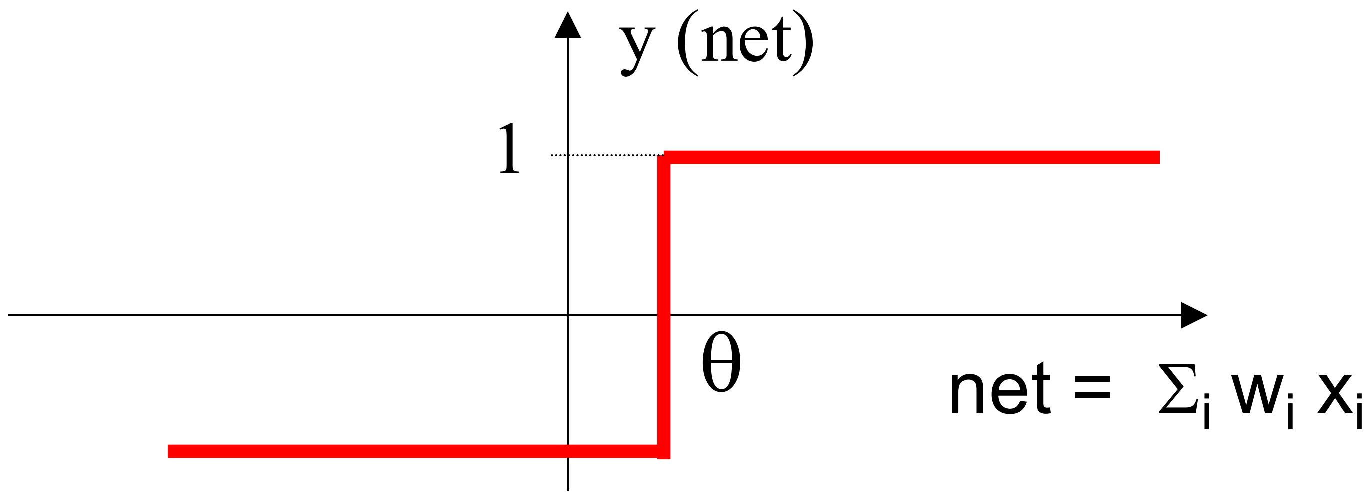
- Asynchronous (sequential)

  The neurons change state one at a time.

  We must define a selection criterion.

Only fully-connected networks have both types of activation

Hopfield model



$$y = \mathrm{sgn}\left(\sum_{i=1}^{n} w_i x_i - \theta\right)$$

The Energy Function

- Each state is characterized by an energy:

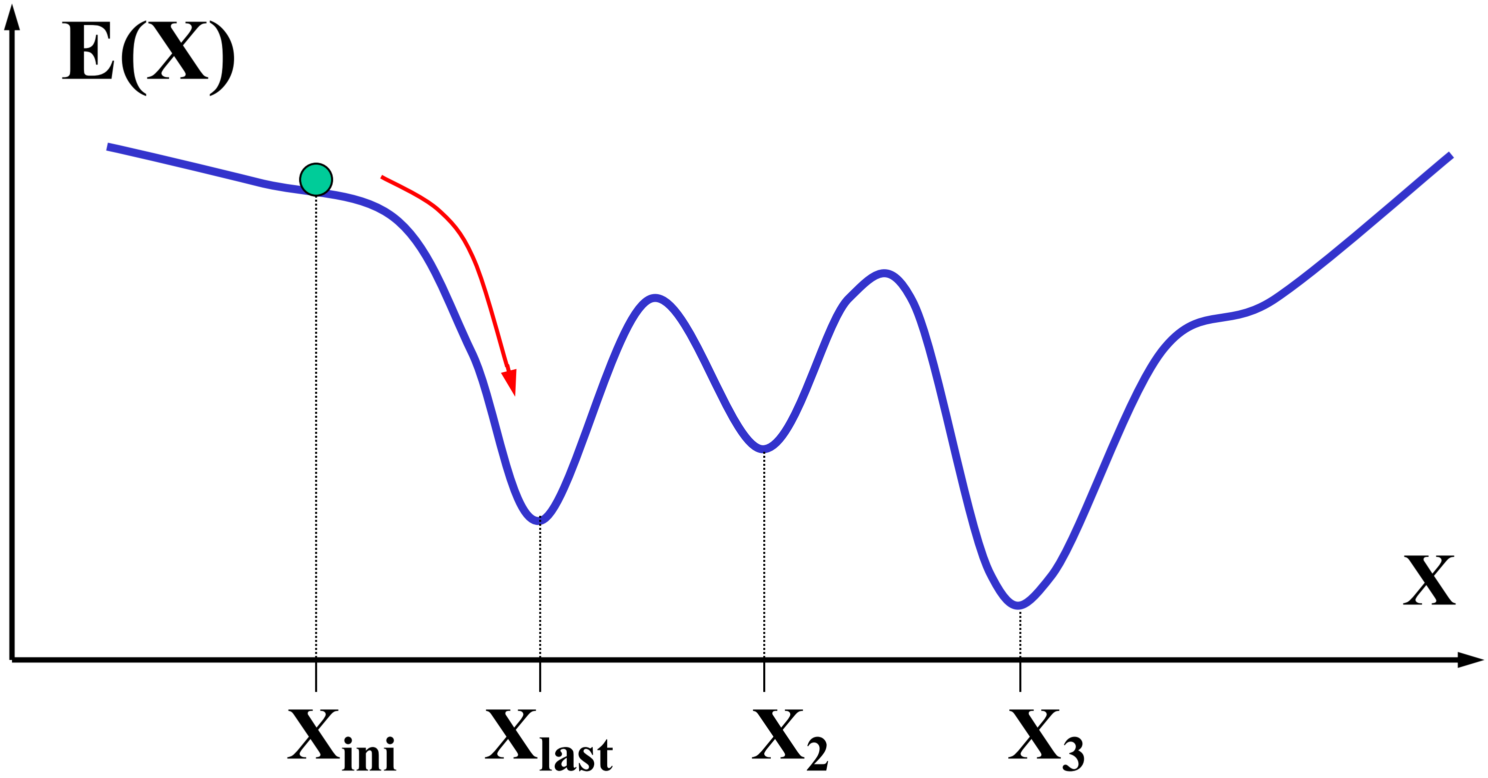
$$\mathbf{E(X)} = -\frac{1}{2}\mathbf{X}^{T}\mathbf{WX}$$

- If the matrix of weights is symmetric and the activation is asynchronous then

E (X) is non-increasing monotonic
in the state evolution

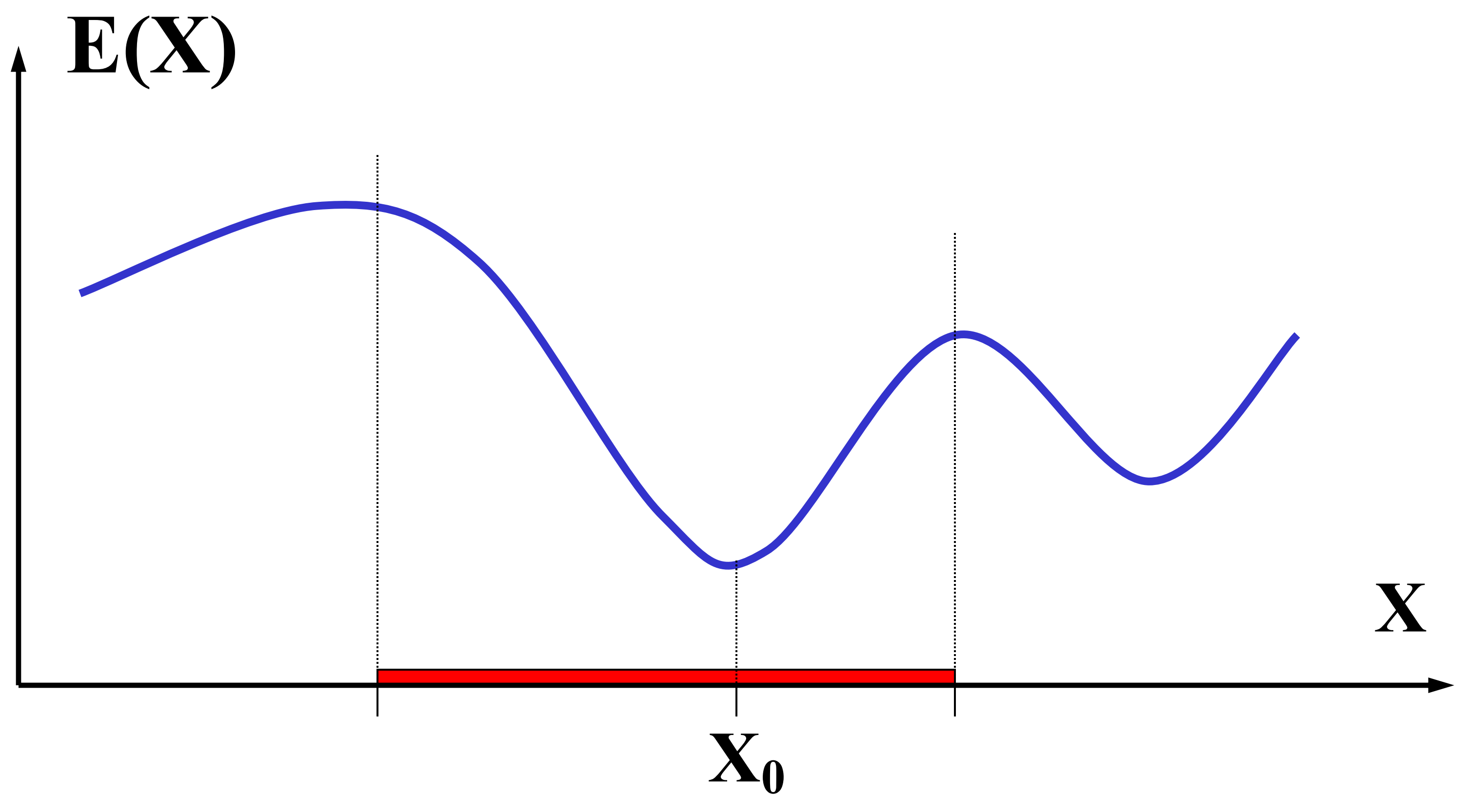
$$E[X(t+1)] \leq E[X(t)]$$

The network evolves towards a stable state

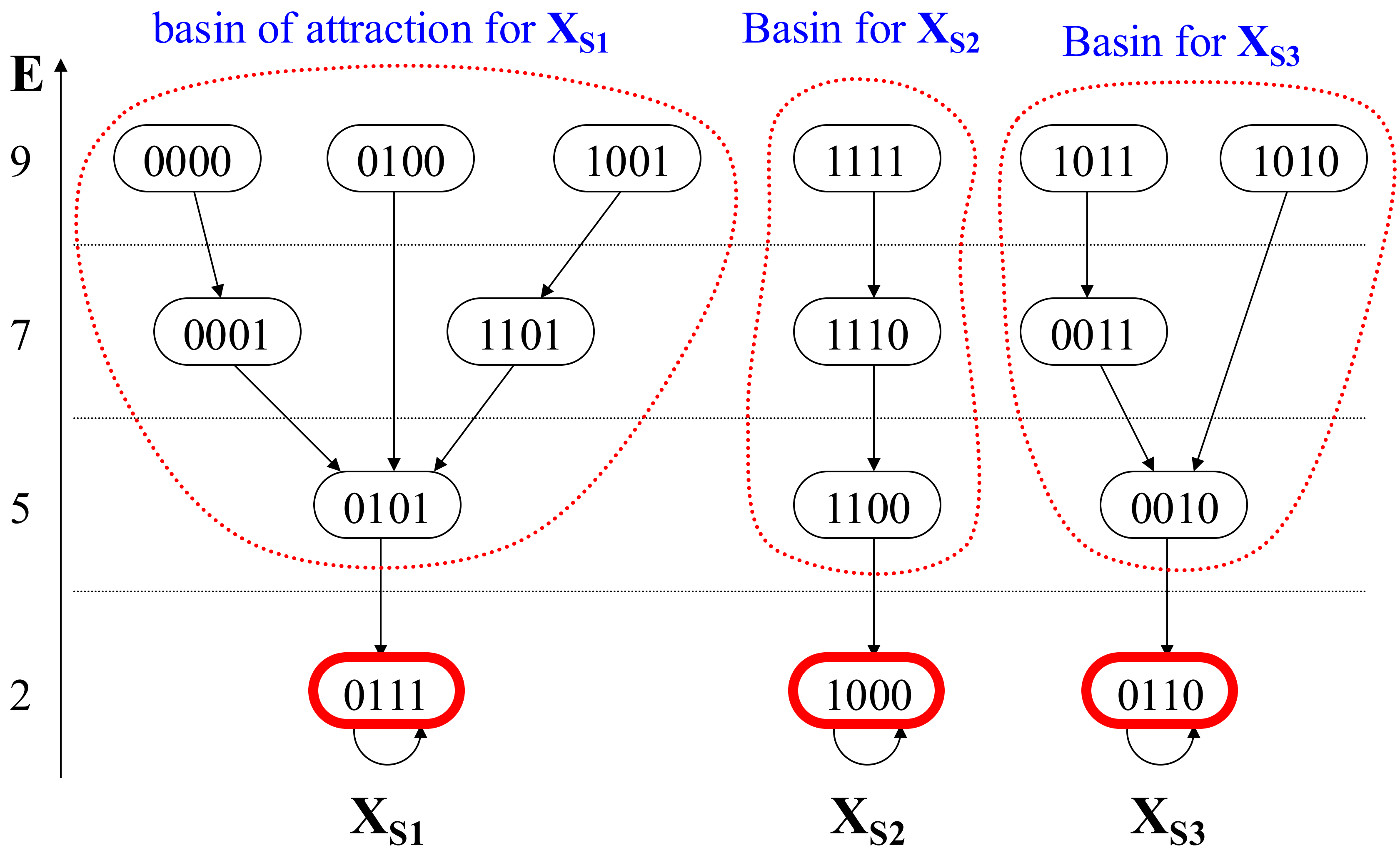


Basin of attraction:

set of states such that all trajectories that start from them end in the same stable state.

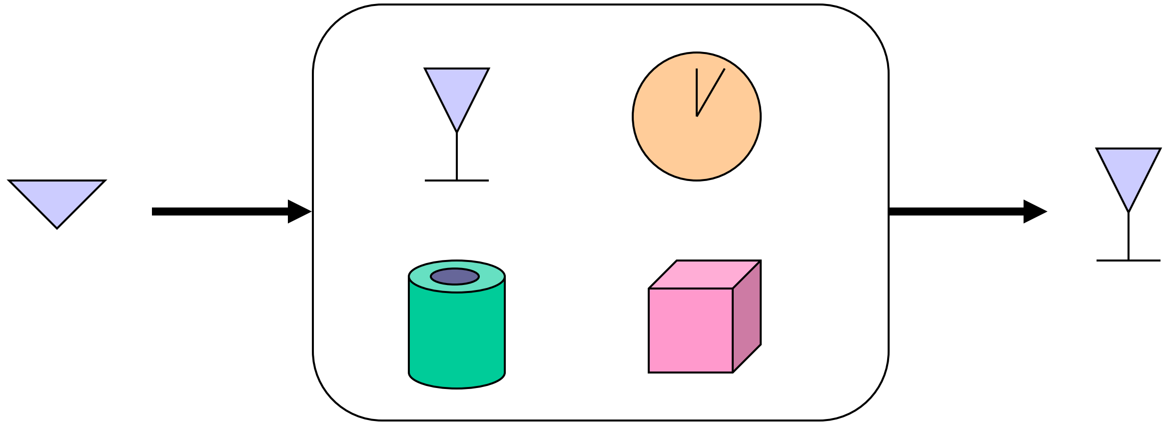


Network with 3 stable states



Associative memories

Memories whose contents can be retrieved on the basis of **partial** or **distorted** information on the content itself.



Storing pictures

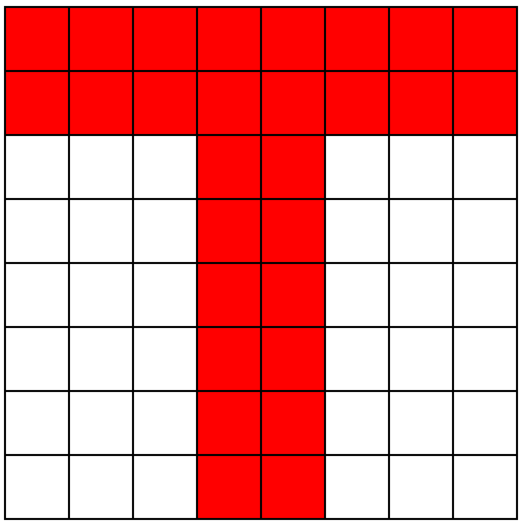


Image: $\boldsymbol{n} \times \boldsymbol{m}$ pixel

Neurons: $\mathrm{N} = \boldsymbol{n} \times \boldsymbol{m}$

Connections: $\mathrm{C} = \mathbf{N}^{2}$

States: $\mathrm{S} = \mathbf{2}^{\mathrm{N}}$

Image: $\boldsymbol{8} \times \boldsymbol{8}$ pixel

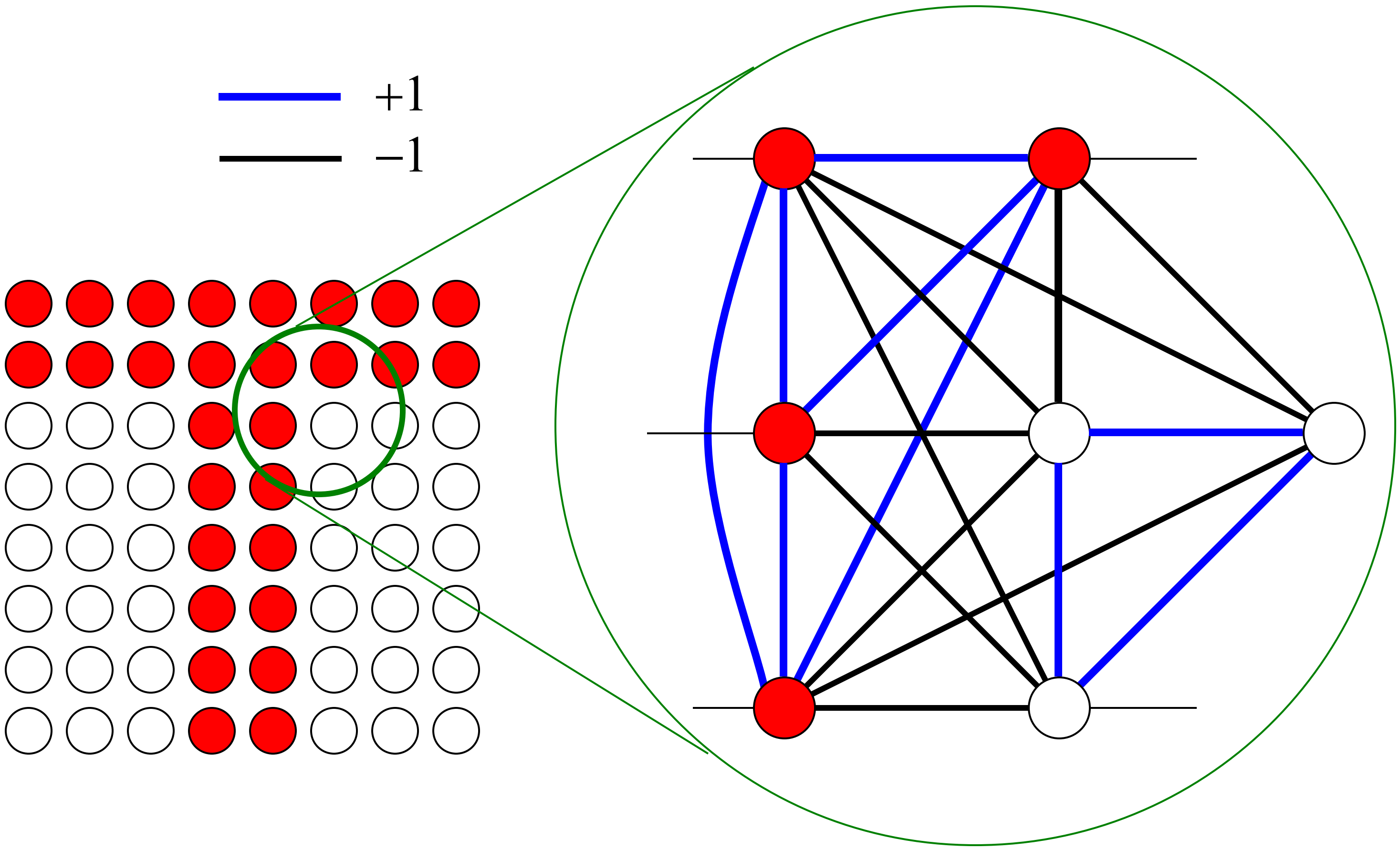
Neurons: $\mathrm{N} = \boldsymbol{64}$

Connections: $\mathrm{C} = \boldsymbol{4096}$

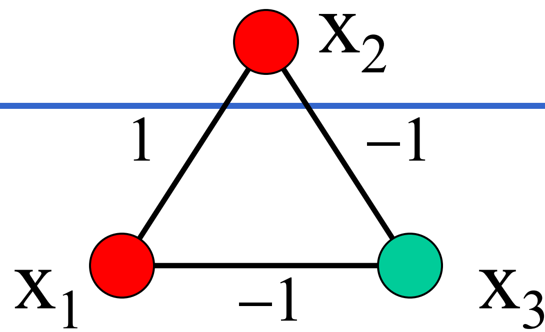
States: $\mathrm{S} \cong \boldsymbol{2 \cdot 10^{19}}$

Rule of storage

(Hopfield '82)

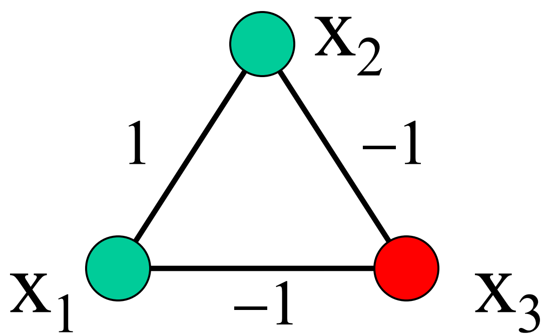


M1: $(+ + -)$



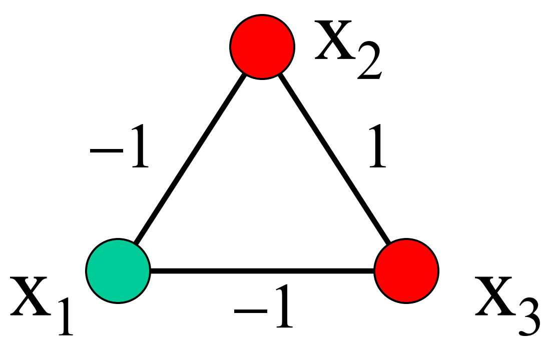
$$W_1 = \begin{bmatrix} 0 & 1 & -1 \\ 1 & 0 & -1 \\ -1 & -1 & 0 \end{bmatrix}$$

M2: $(- - +)$



$$W_2 = \begin{bmatrix} 0 & 1 & -1 \\ 1 & 0 & -1 \\ -1 & -1 & 0 \end{bmatrix}$$

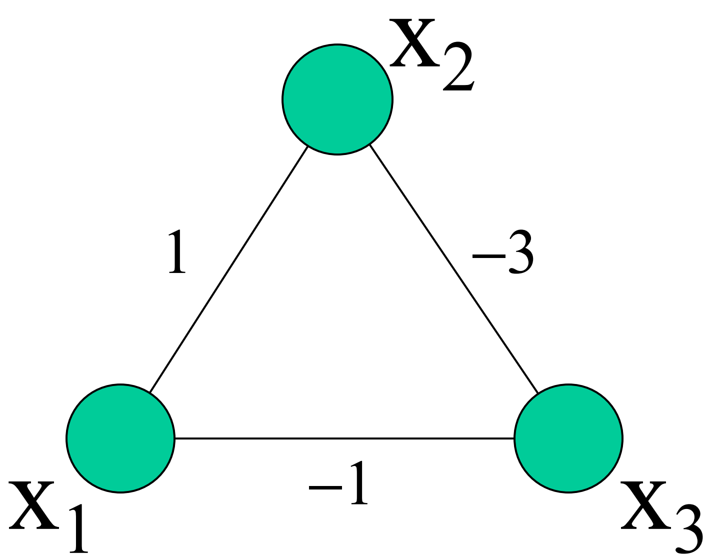
M3: $(- + +)$



$$W_3 = \begin{bmatrix} 0 & -1 & -1 \\ -1 & 0 & 1 \\ -1 & 1 & 0 \end{bmatrix}$$

Overall network

$$W = \sum_{k=1}^{m} W_k = \begin{pmatrix} 0 & 1 & -3 \\ 1 & 0 & -1 \\ -3 & -1 & 0 \end{pmatrix}$$



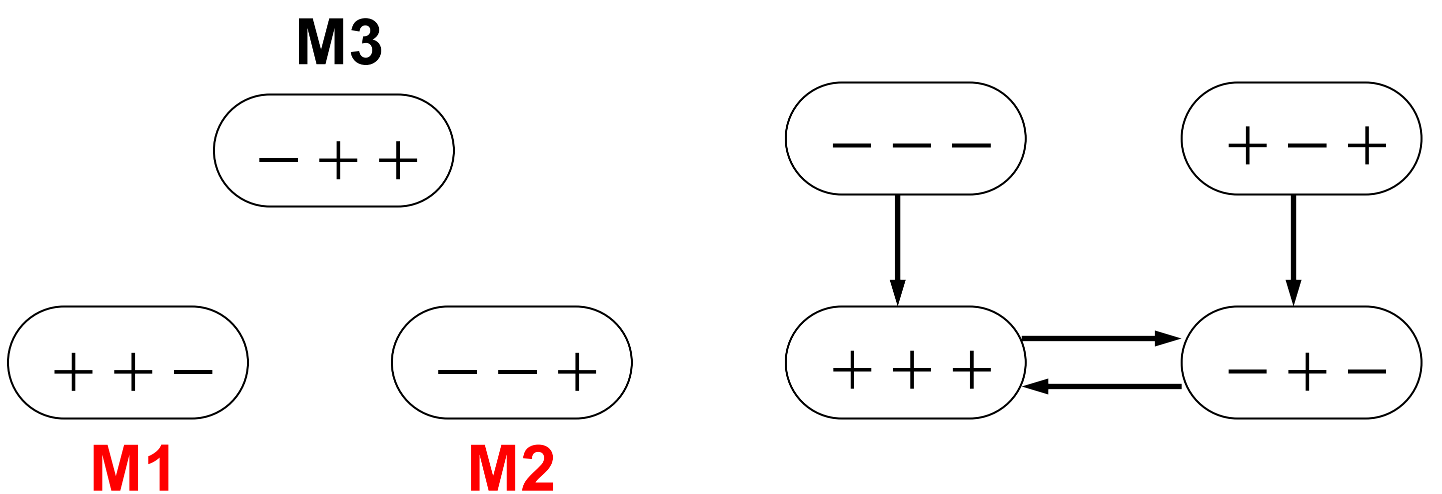
Add matrices for the individual states to made stable

DOES IT WORK???

Transition Diagram

(Synchronous activation)

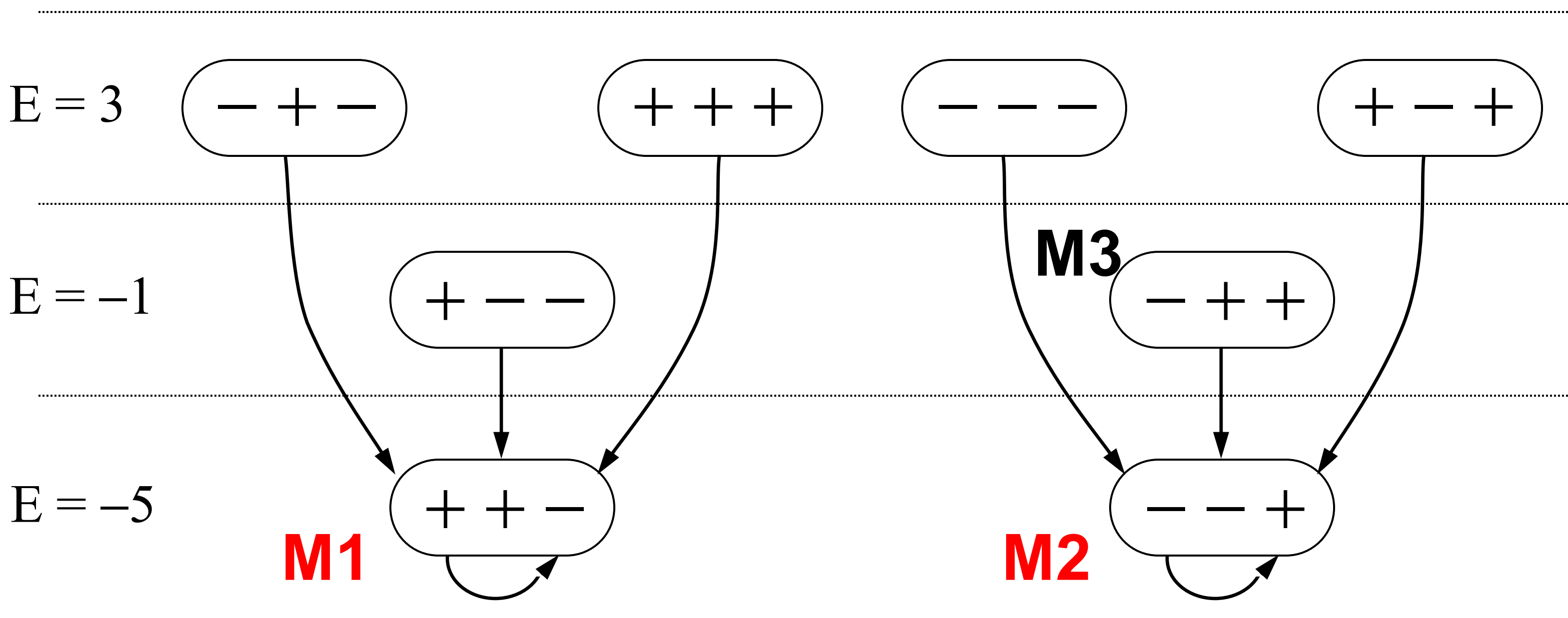
$$M = \{(++-), (--+), (-++)\}$$



Transition Diagram

(Asynchronous Activation)

$$M = \{(++-), (--+), (-++)\}$$



Remarks

When we overlap too many memories:

- Not always all memories are stable

  The creation of a local minimum can have the effect of removing another one.

- Spurious memories can appear

  The surface energy can have complex shapes.

Learning

Network capacity to change behavior in a desired direction by changing synaptic connections (weights).

The learning paradigms can be divided into three basic classes:

- **supervised**
- **competitive**
- **reinforcement**

1- Supervised learning

It is the most widely used.

The network learns to recognize a set of desired input configurations.

The network operates in two distinct phases:

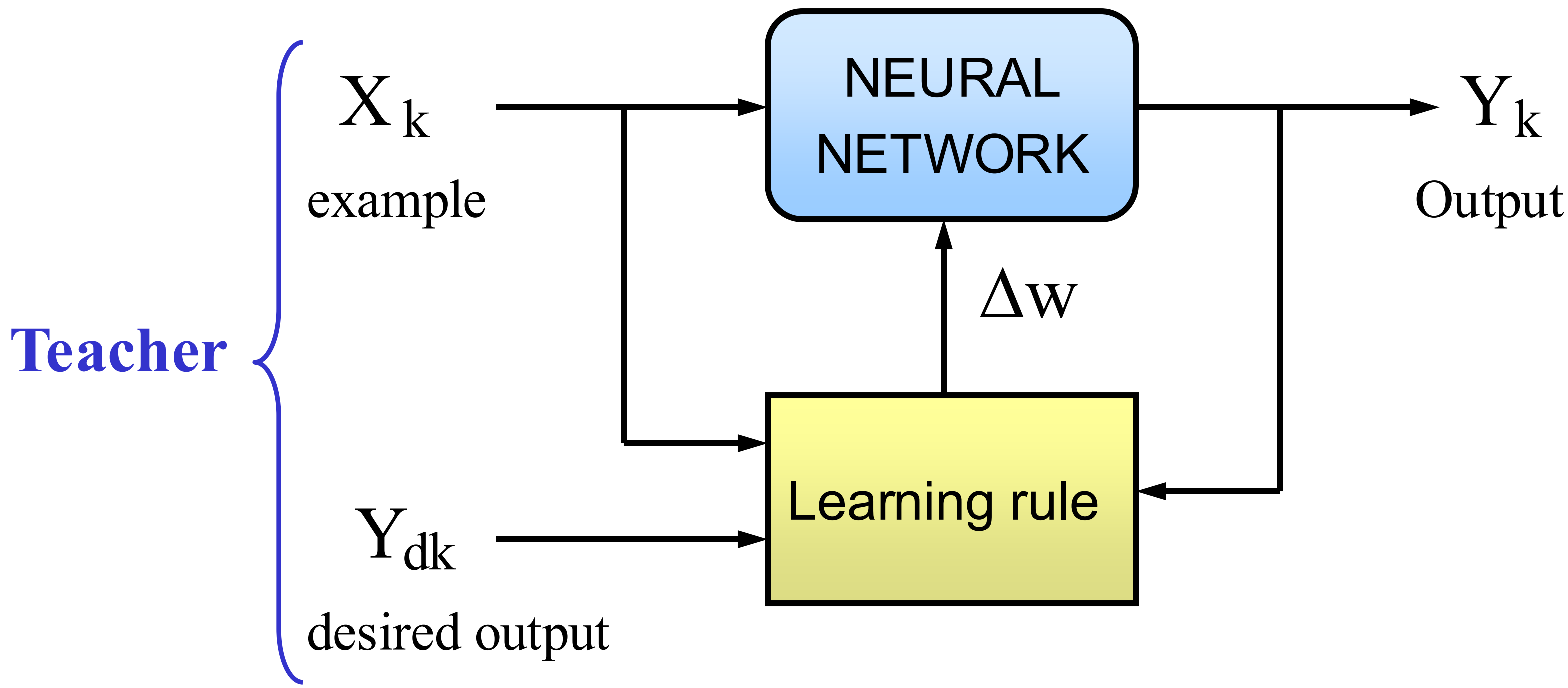
- **Learning phase**

  it stores the desired information via examples

- **Evolution phase**

  it retrieves the stored information

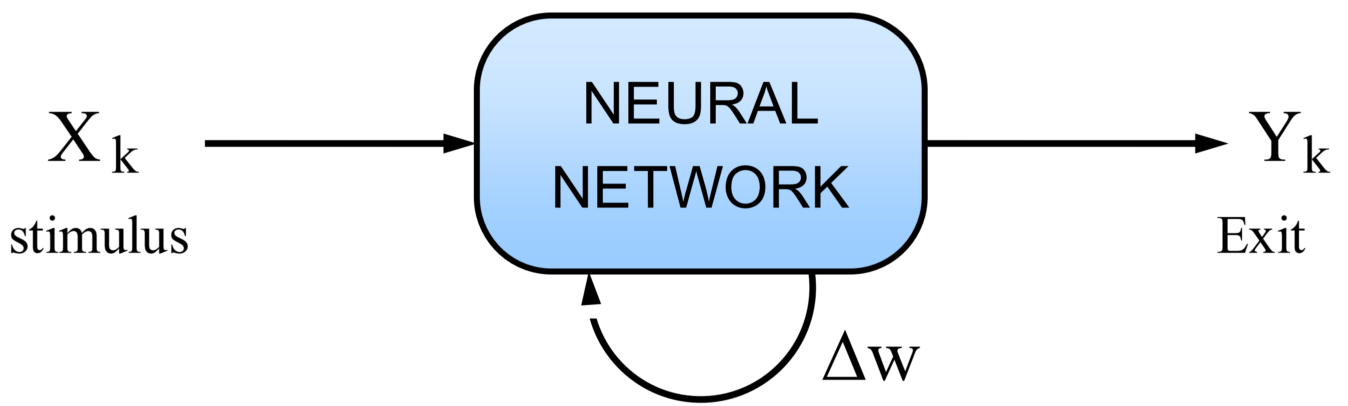
Training phase



2- Competitive learning

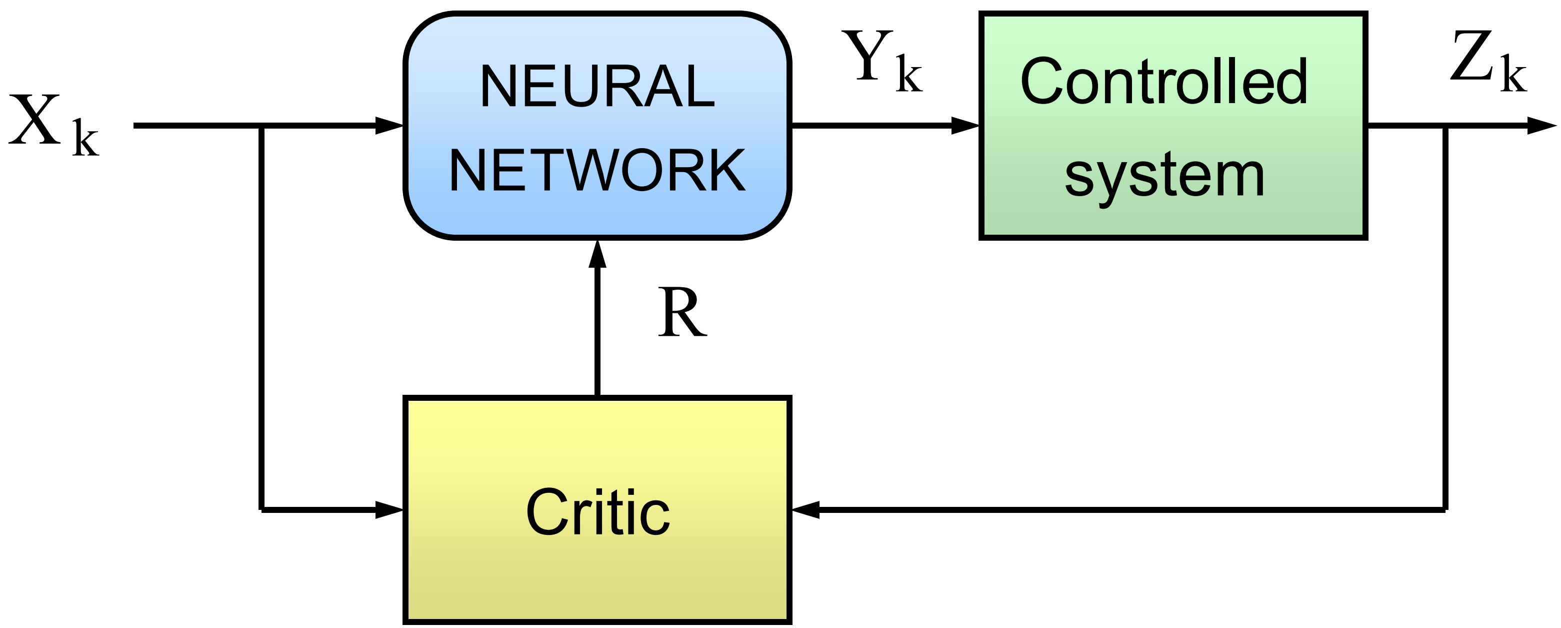
- Neurons compete for specializing in the recognition of a particular stimulus. Similar stimuli end up in the same class.

- In the end, each neuron is activated by a given stimulus (isomorphism between stimuli and output neurons).



3- Reinforcement learning

- Reinforcement learning simulates the learning mechanism in animals based on reward and punishment: used for control systems applications



Supervised learning

Supervised learning

The network learns to associate a set of given pairs (X_k, Y_{dk}).

The network operates in two distinct phases:

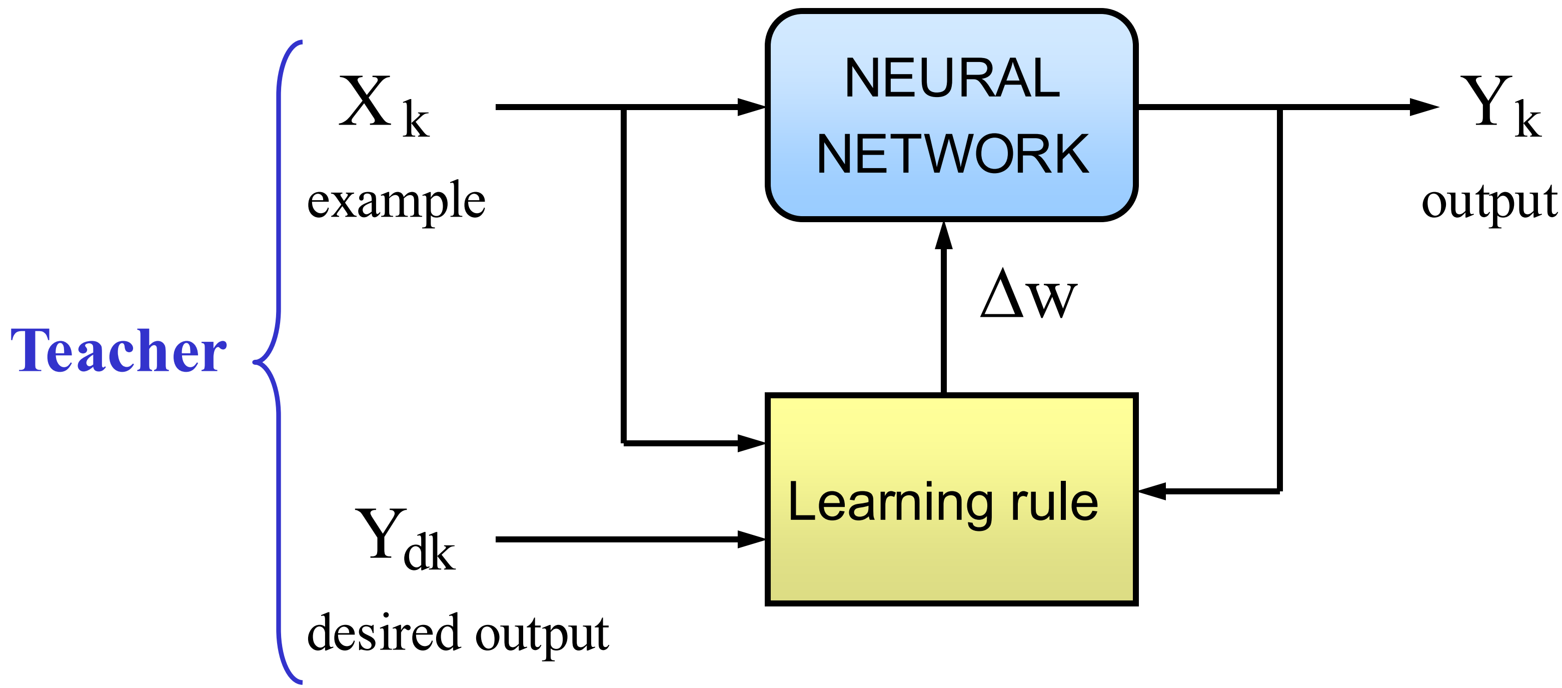
- **Learning phase**

  They store the desired information

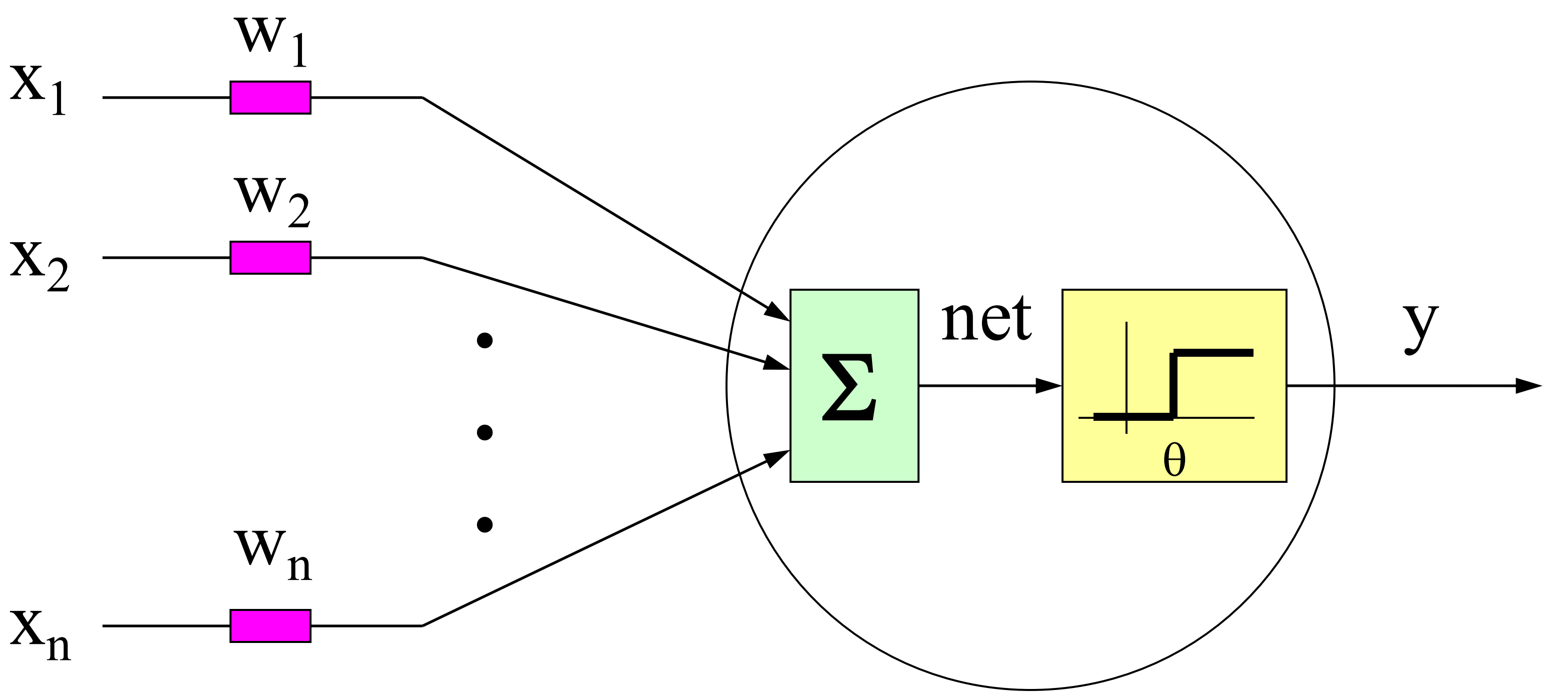
- **Evolution phase**

  retrieving the stored information

Training phase



The Perceptron (Rosenblatt '58)



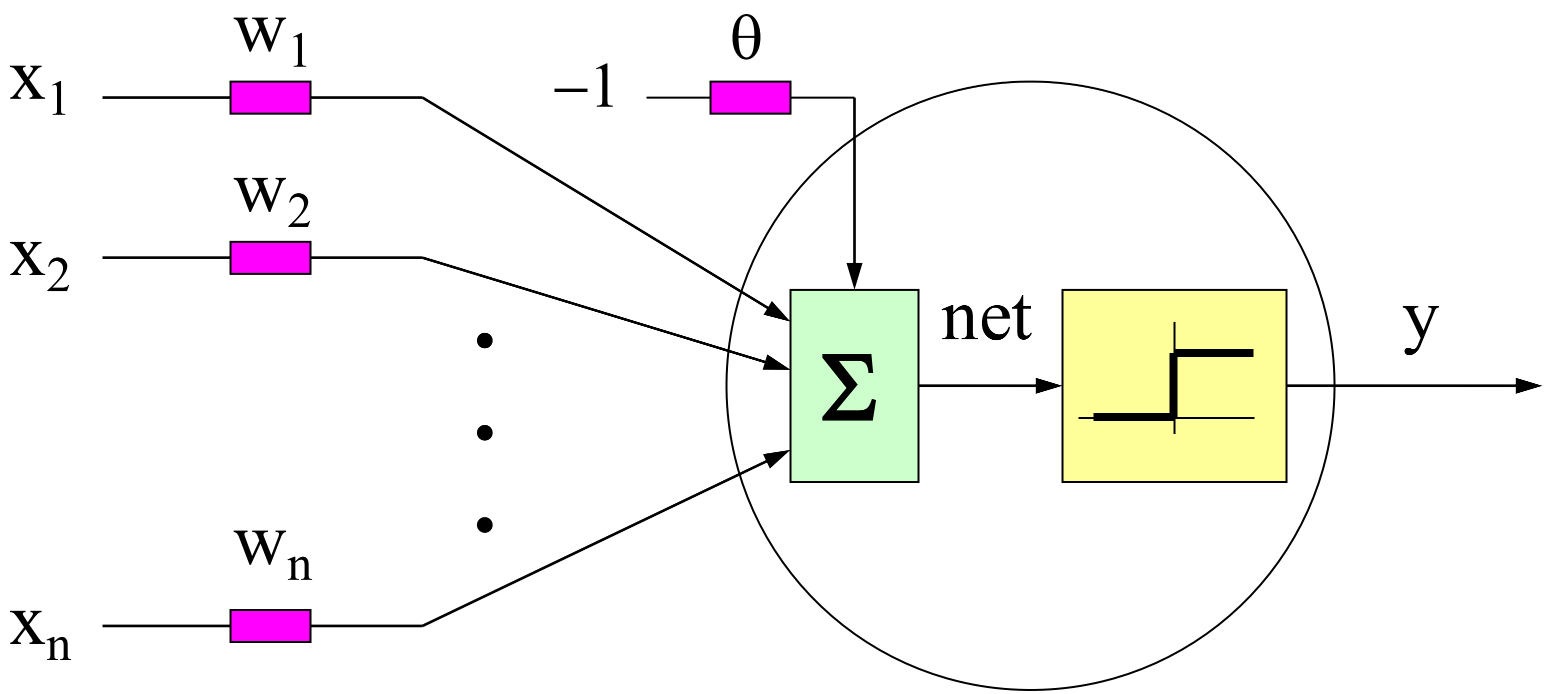
Binary input

$$net = \Sigma_i \, w_i x_i$$

$$y = HS\,(net - \theta)$$

Binary output

The Perceptron (Rosenblatt '58)



Binary input

$$\text{net} = \textstyle\sum_i w_i x_i - \theta$$

$$y = \text{HS}(\text{net})$$

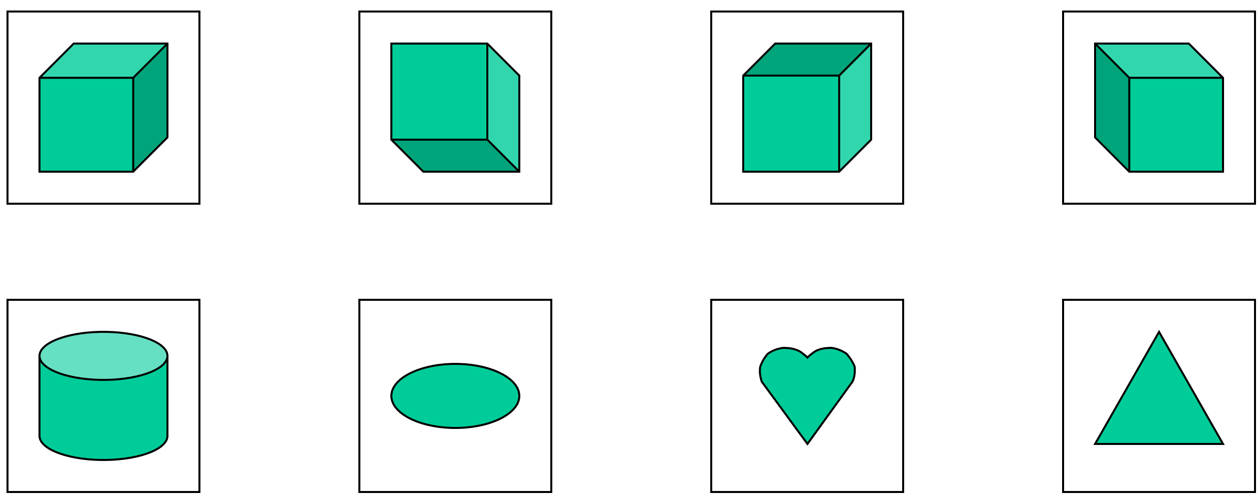
Binary output

Classification

- A perceptron can be trained to recognize whether an input pattern X belongs or not to a class C:

CUBE (1)

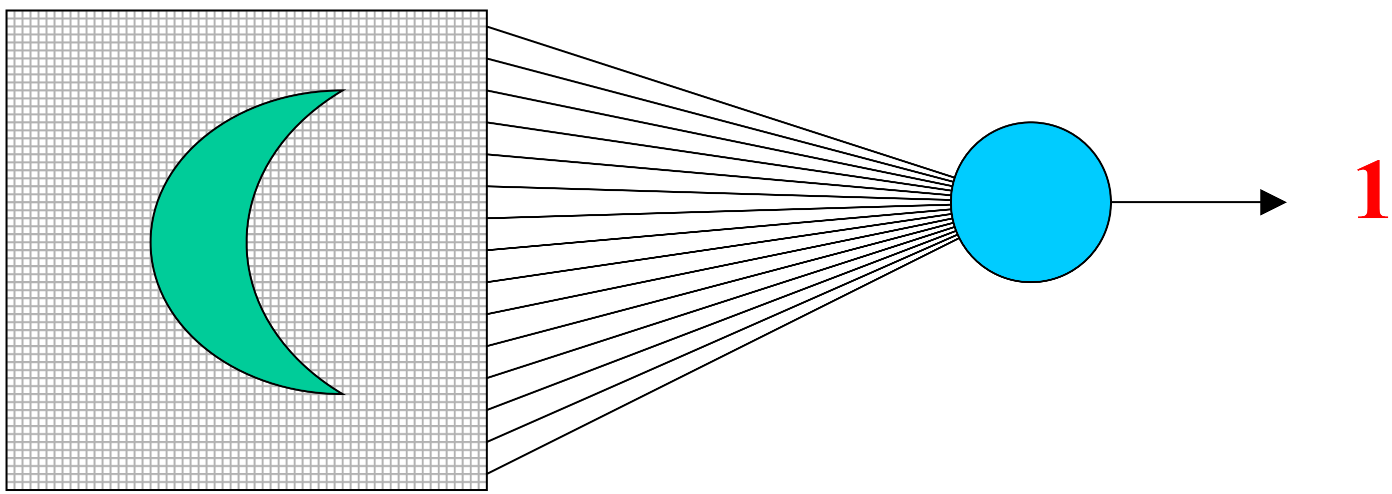
NOT CUBE (0)



The Rosenblatt experiment

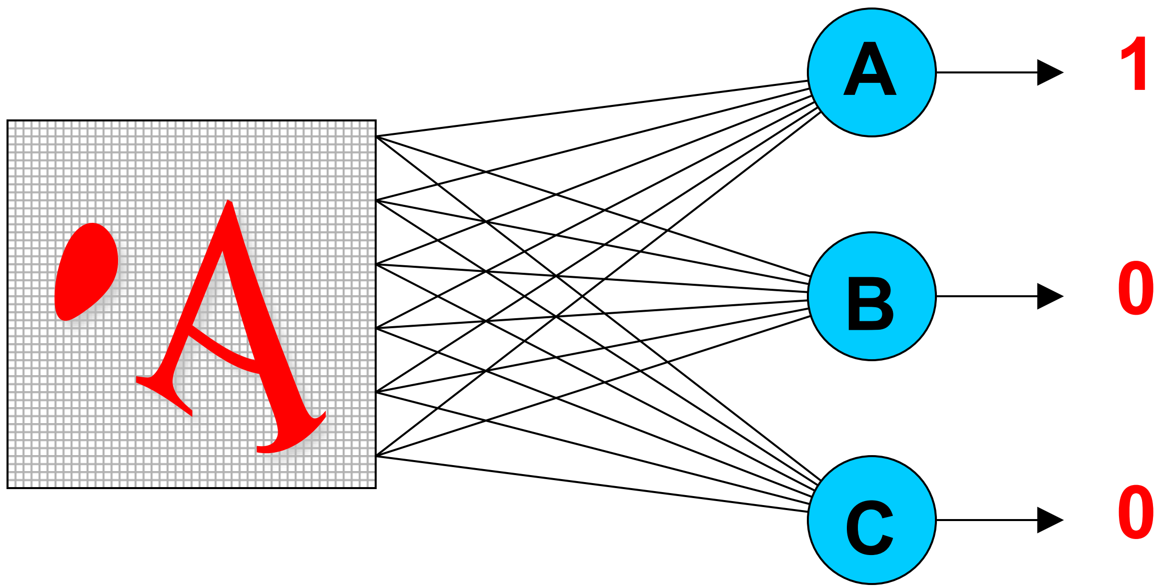
1 = concave

0 = convex

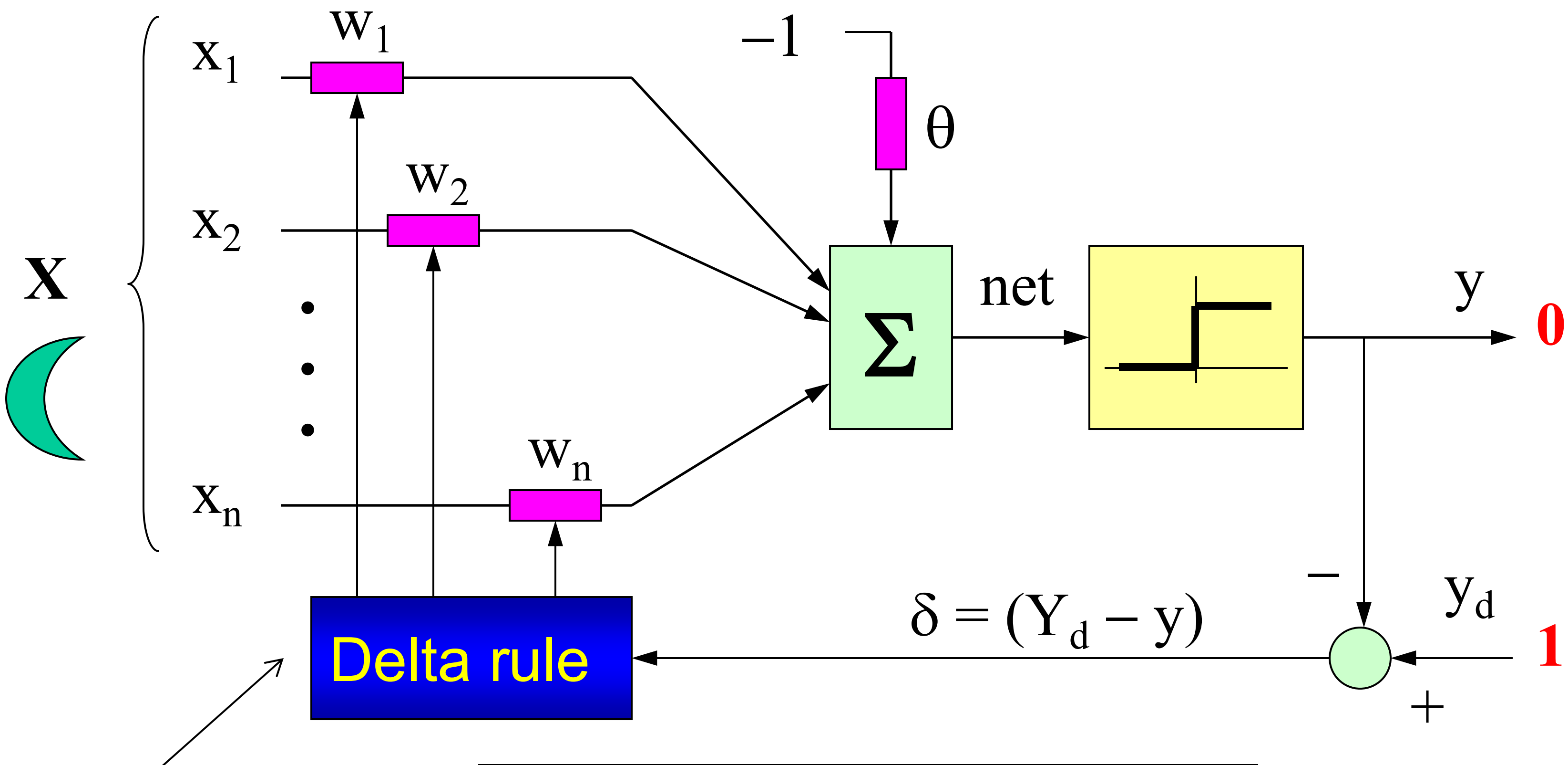


Perceptron networks

Letter Recognizer



Training



θ Built in weights.
By changing weights
we learn θ

$$w_i(t+1) = w_i(t) + \eta\delta x_i$$

η = Learning coefficient (learning rate)

Learning algorithm

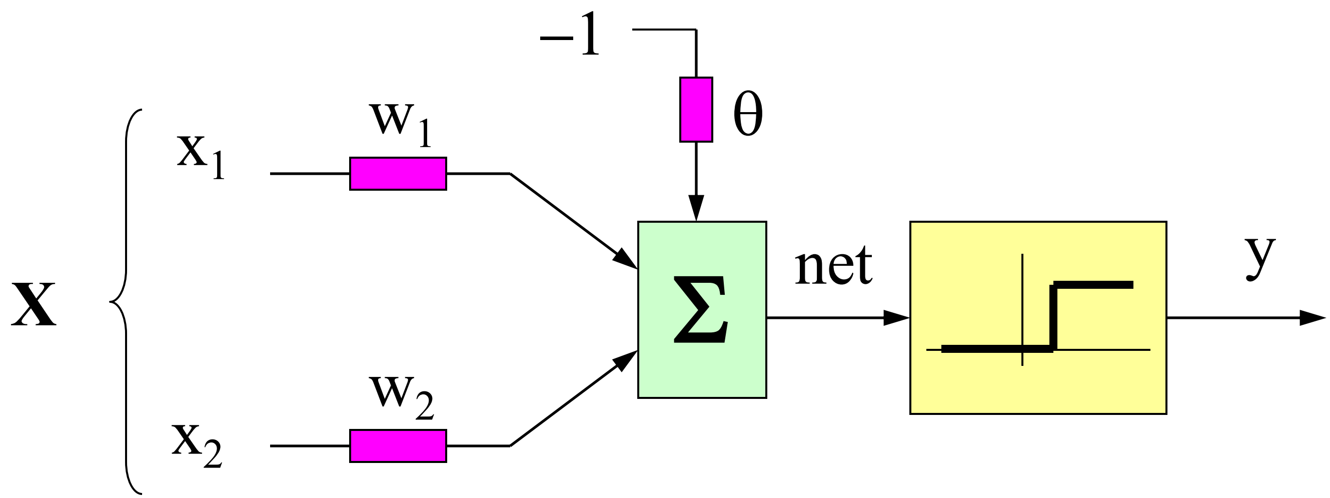
1. Given a training set of m examples:

$$TS = \{(X_k, y_{dk}), k = 1..m\}$$

2. Initialize weights with random values w_i;
3. Give as input a pair (X_k, y_{dk});
4. Compute the answer y_k of the network;
5. Update weights with the *delta rule*: (**$\Delta w = \eta \delta x$**)
6. Repeat from step 3 until all answers are correct:

$$y_k = y_{dk} \quad \forall k \in [1..M]$$

Two inputs Perceptron



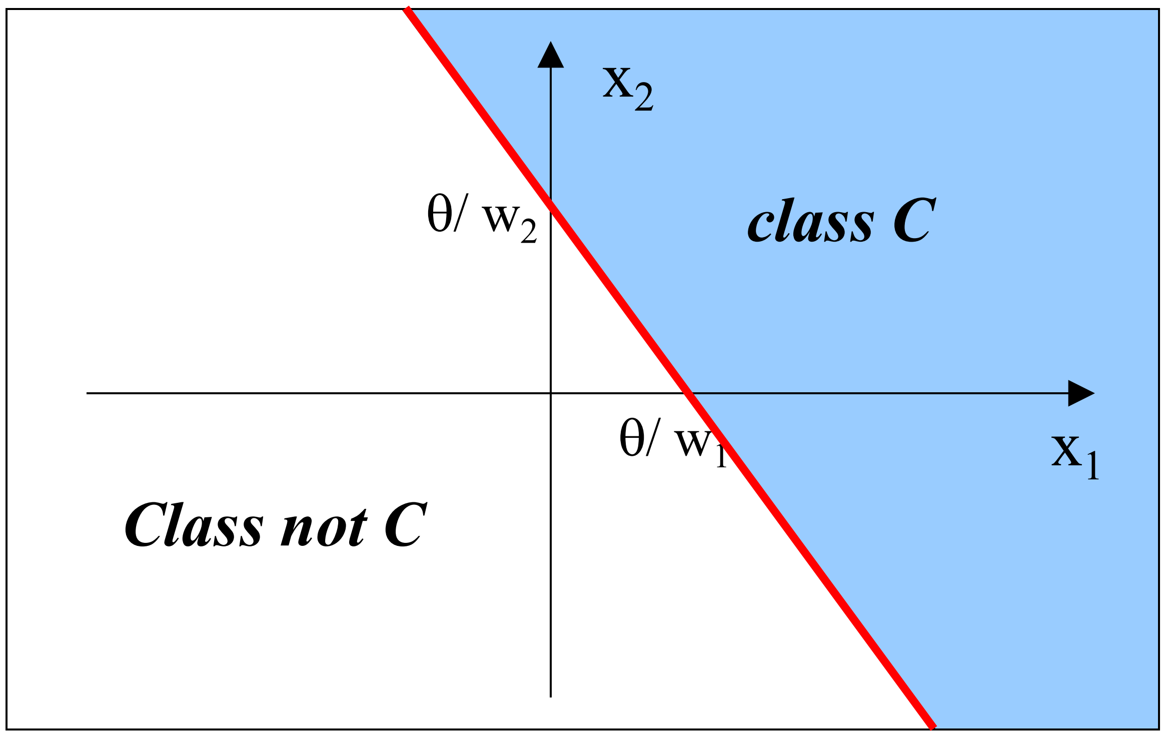
$$y = HS\,(w_1\,x_1 + w_2\,x_2 - \theta)$$

The patterns belonging to the class will be those such that:

$$w_1\,x_1 + w_2\,x_2 - \theta\ > 0$$

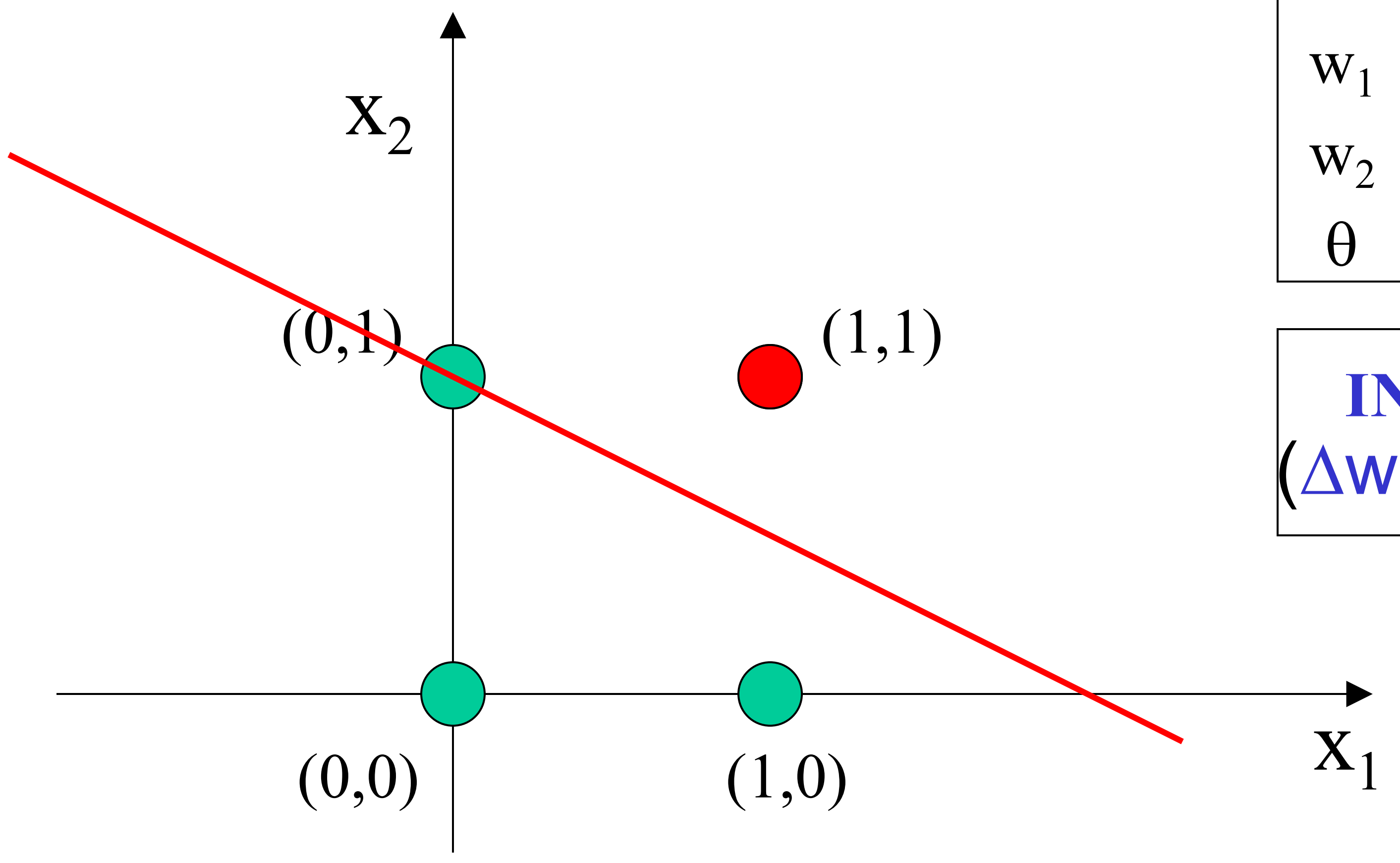
Linear separation of the input space

$$x_2 > -(w_1 / w_2)\, x_1 + \theta / w_2$$



Learning an AND

$$y = 1 \text{ if } x_2 > -(w_1 / w_2)\, x_1 + \theta / w_2$$



WEIGHTS

$w_1 = 0.5$

$w_2 = 1$

$\theta = 1$

INPUT

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

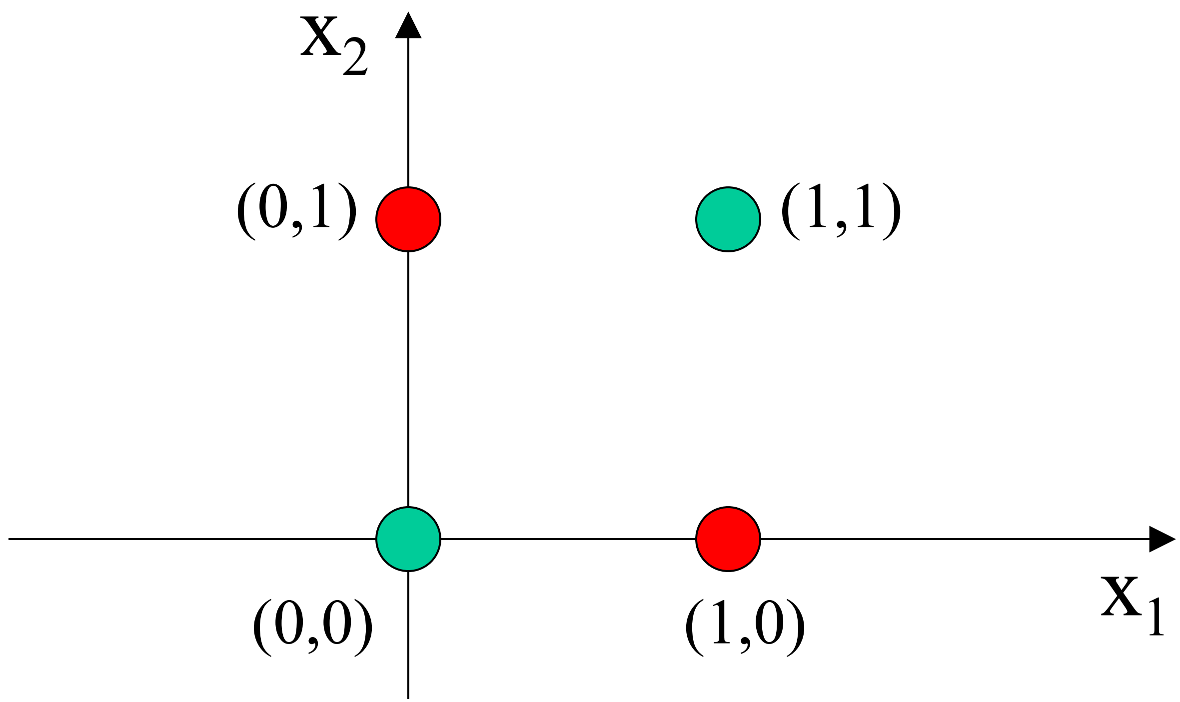
Perceptron limitations

To learn a classification, the problem must be linearly separable:

- the patterns belonging to the class C must be contained in a semiplane of the input space;
- with n inputs, the input space becomes n-dimensional and classes are separated by a hyperplane.

The problem of XOR

It is not linearly separable!



Possible solutions

- Use neurons with appropriate output functions.

- Combine neurons answer, in multilayer architectures.

Appropriate output functions

- $f\,(net) = net^2,\ w_1 = 1,\ w_2 = -1$

$$y = (x_1 - x_2)^2$$

- $f\,(net) = |\ net\ |,\ w_1 = 1,\ w_2 = -1$

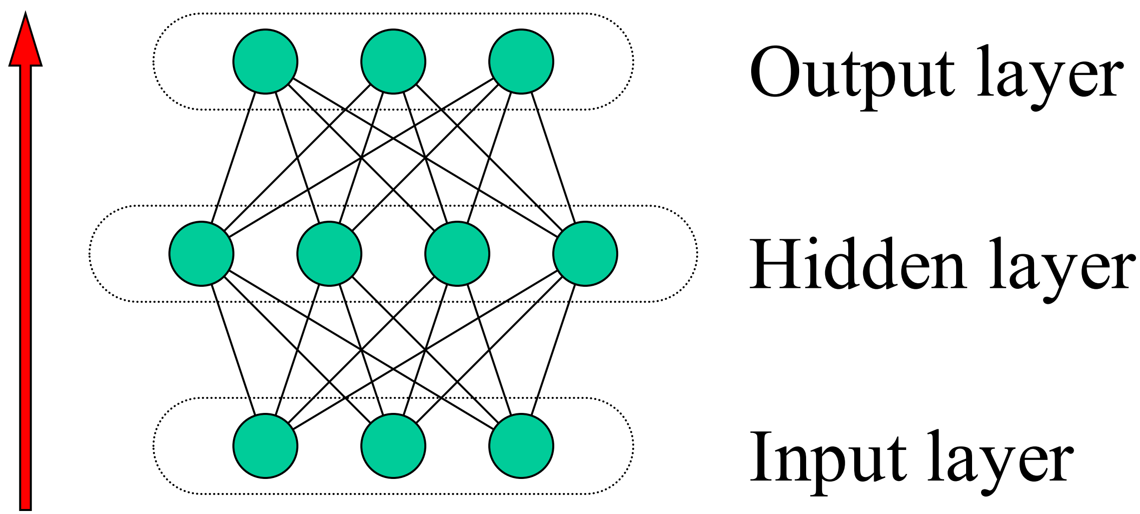
$$y = |\ x_1 - x_2|$$

- $f\,(net) = 1 - e^{-|\ net\ |},\ w_1 = 1,\ w_2 = -1$

$$y = 1 - e^{-|\ x1 - x2\ |}$$

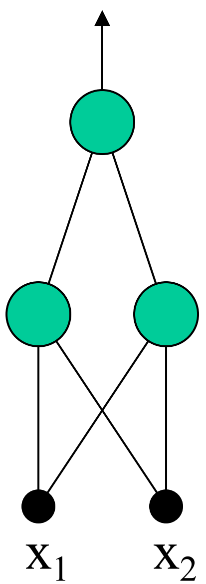
Multilayer networks

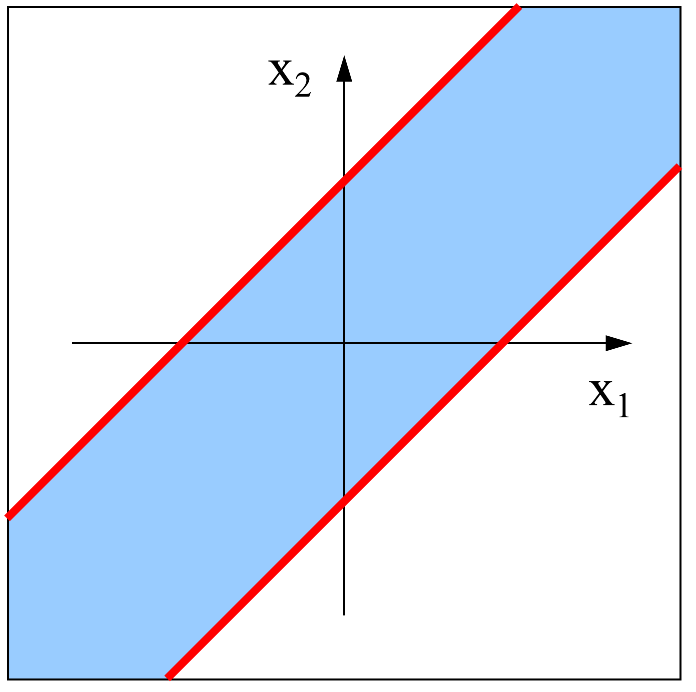
- Each neuron of a layer is connected with all neurons of the nearby layer.
- There are no connections between neurons of the same layer.



Three-layer networks

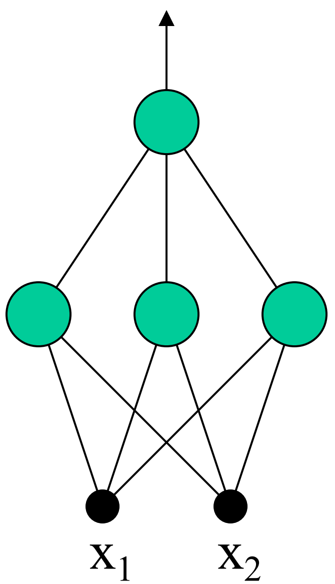
- They are able to separate convex regions number of edges ≤ number hidden neurons

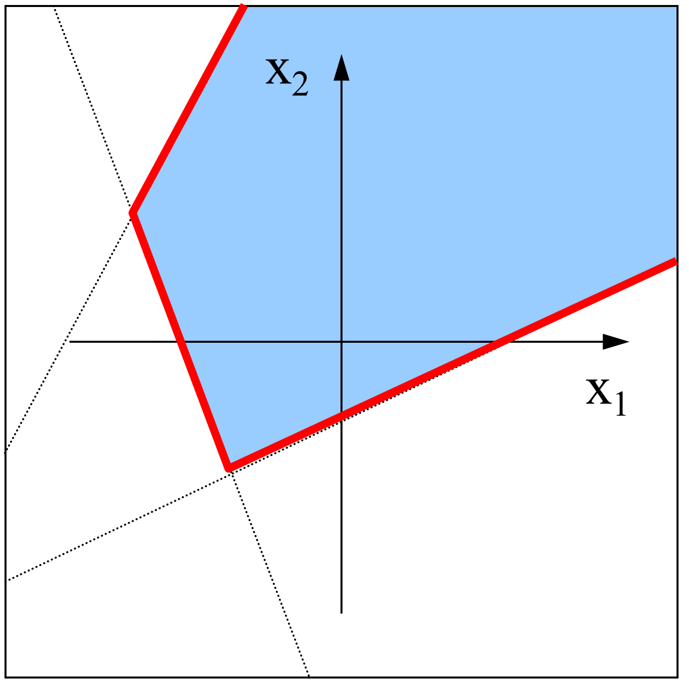




Three-layer networks

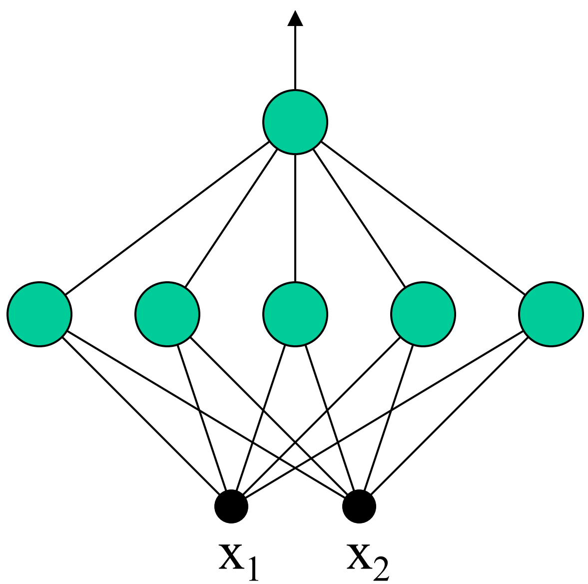
- They are able to separate convex regions
  number of edges ≤ number hidden neurons

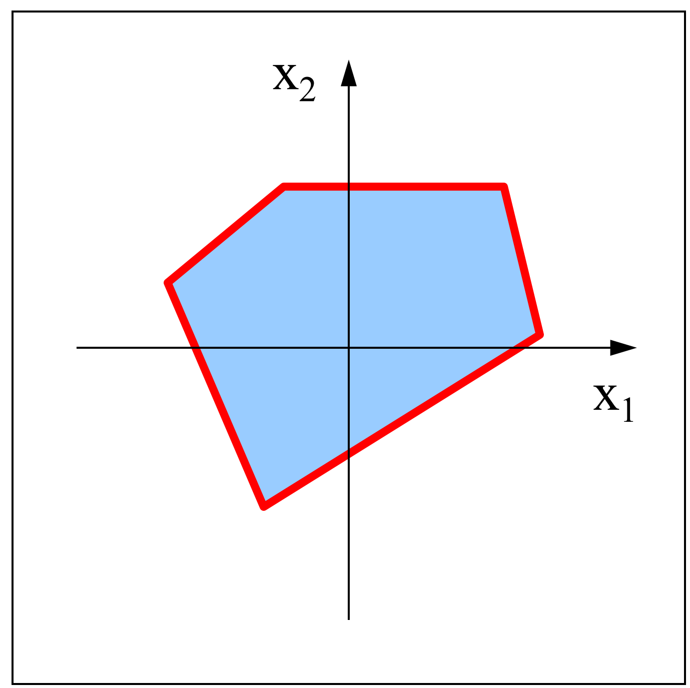




Three-layer networks

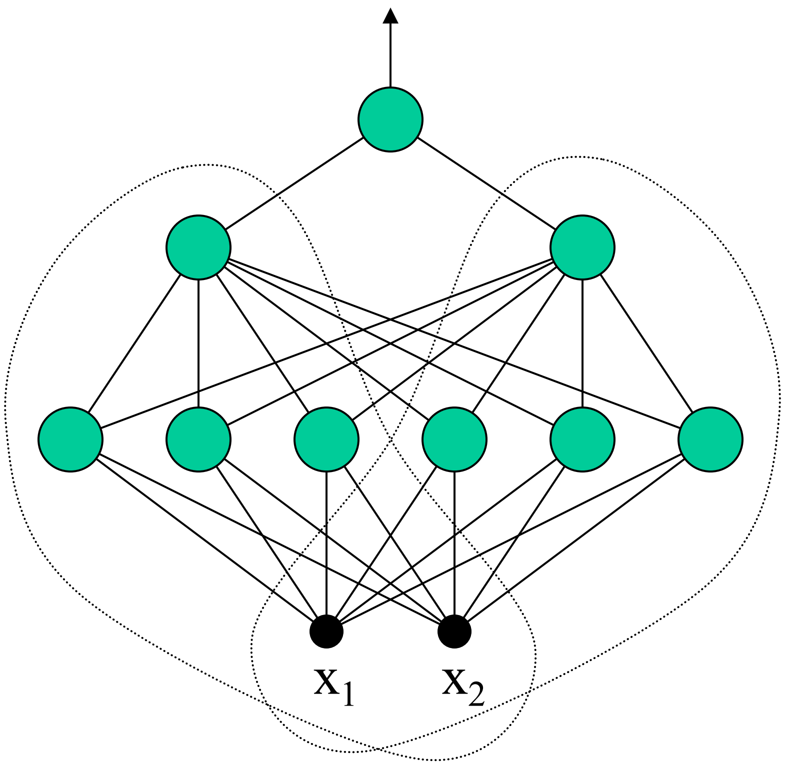
- They are able to separate convex regions
  number of edges ≤ number hidden neurons

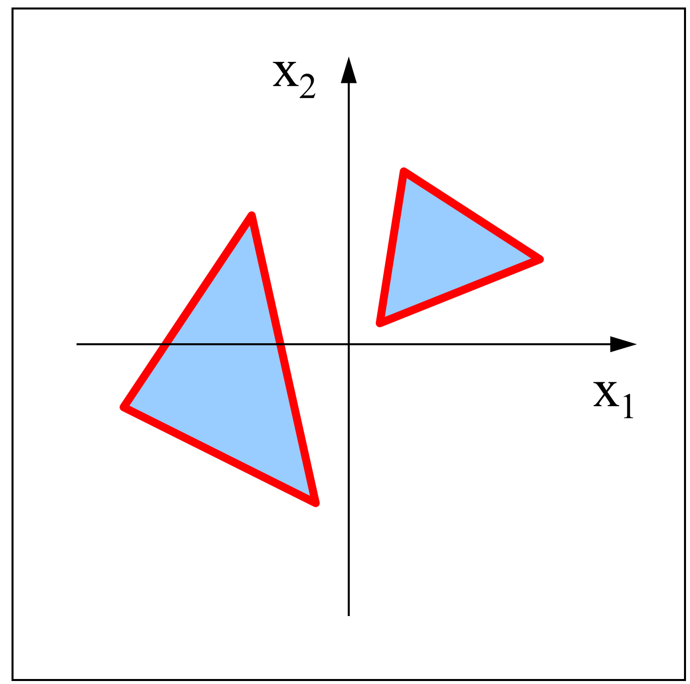




Four-layer networks

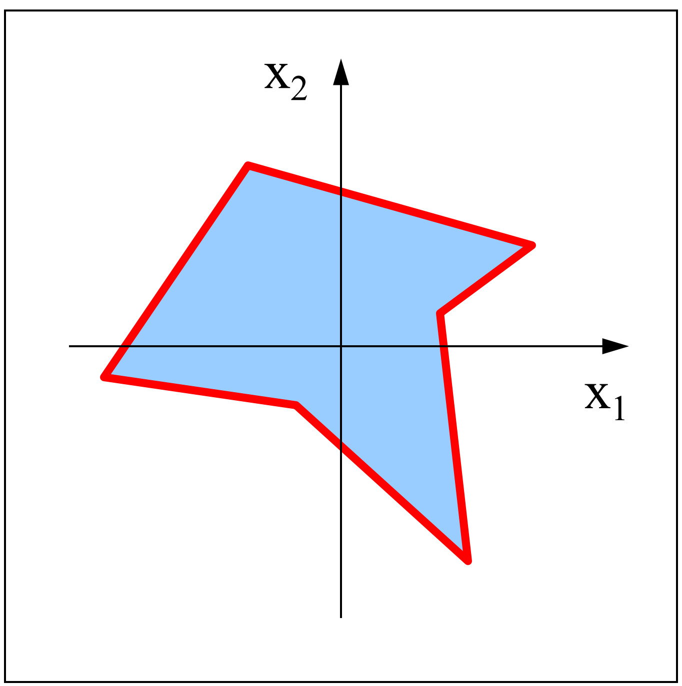
- They are able to separate regions of every shape

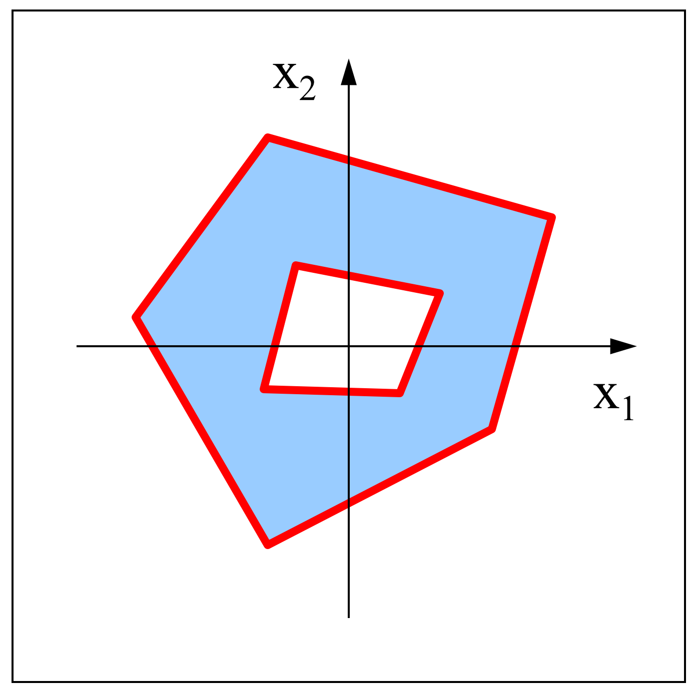




Four-layer networks

- The addition of other layers does not improve the classification ability.

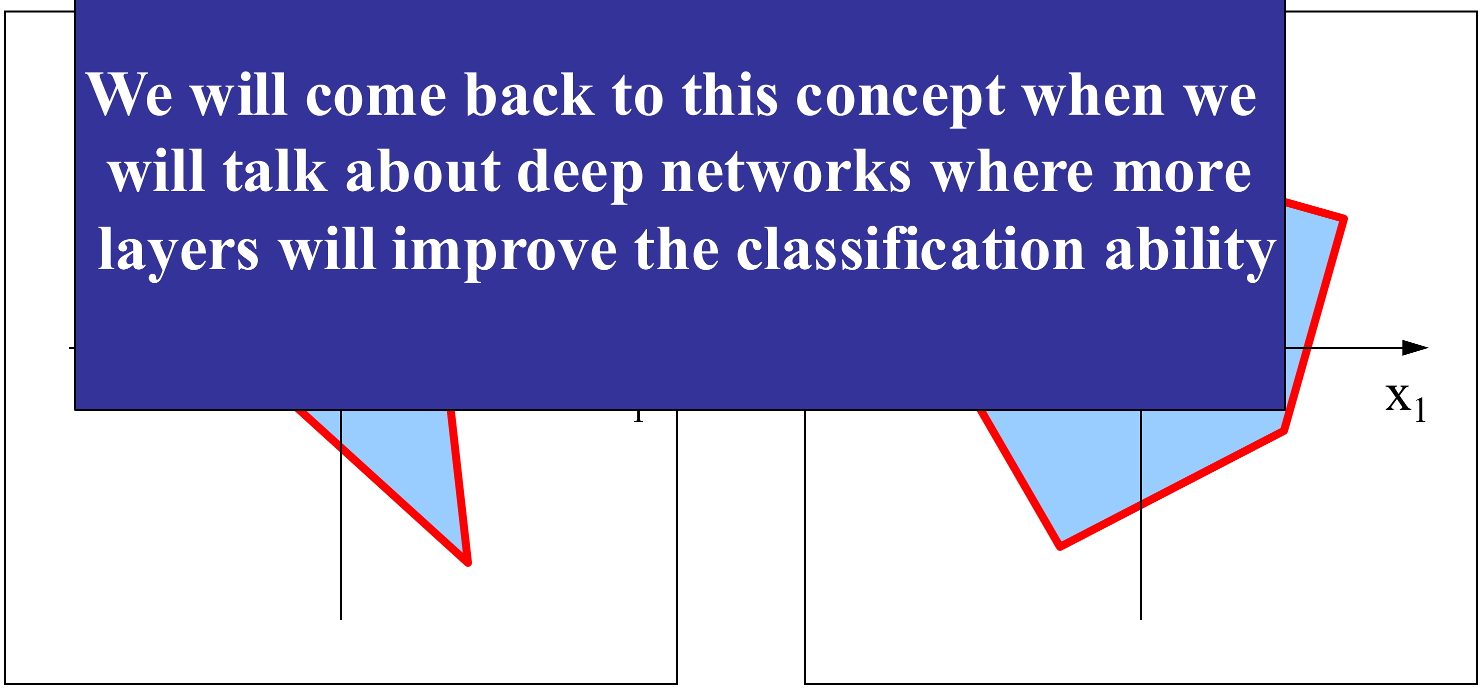




Four-layer networks

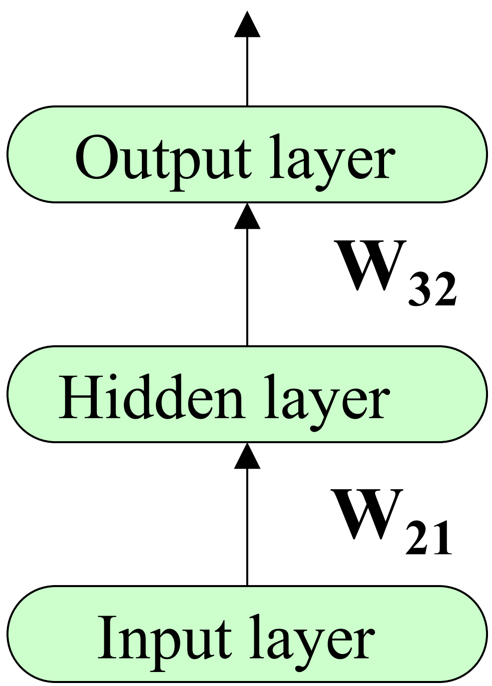
- The addition of other layers does not improve the classification ability.

We will come back to this concept when we will talk about deep networks where more layers will improve the classification ability



Importance of the non-linearity

- If the output functions were linear, a network with N layers would always be reduced to 2 layers:



$$Z = W_{32}\, Y = (W_{32} W_{21})\, X = W_{eq} X$$

$$Y = W_{21}\, X$$

$$X$$

Implications

To perform complex classifications, neurons must be non-linear and be organized on multiple layers.

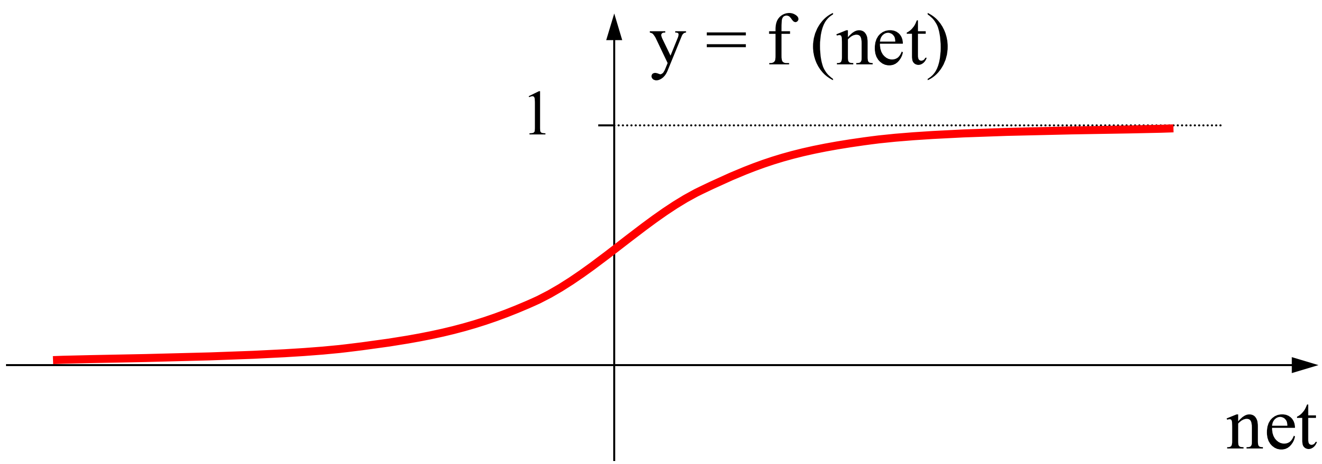
Problems

- How do you train a multilayer network?
- What is the desired output of the hidden neurons?

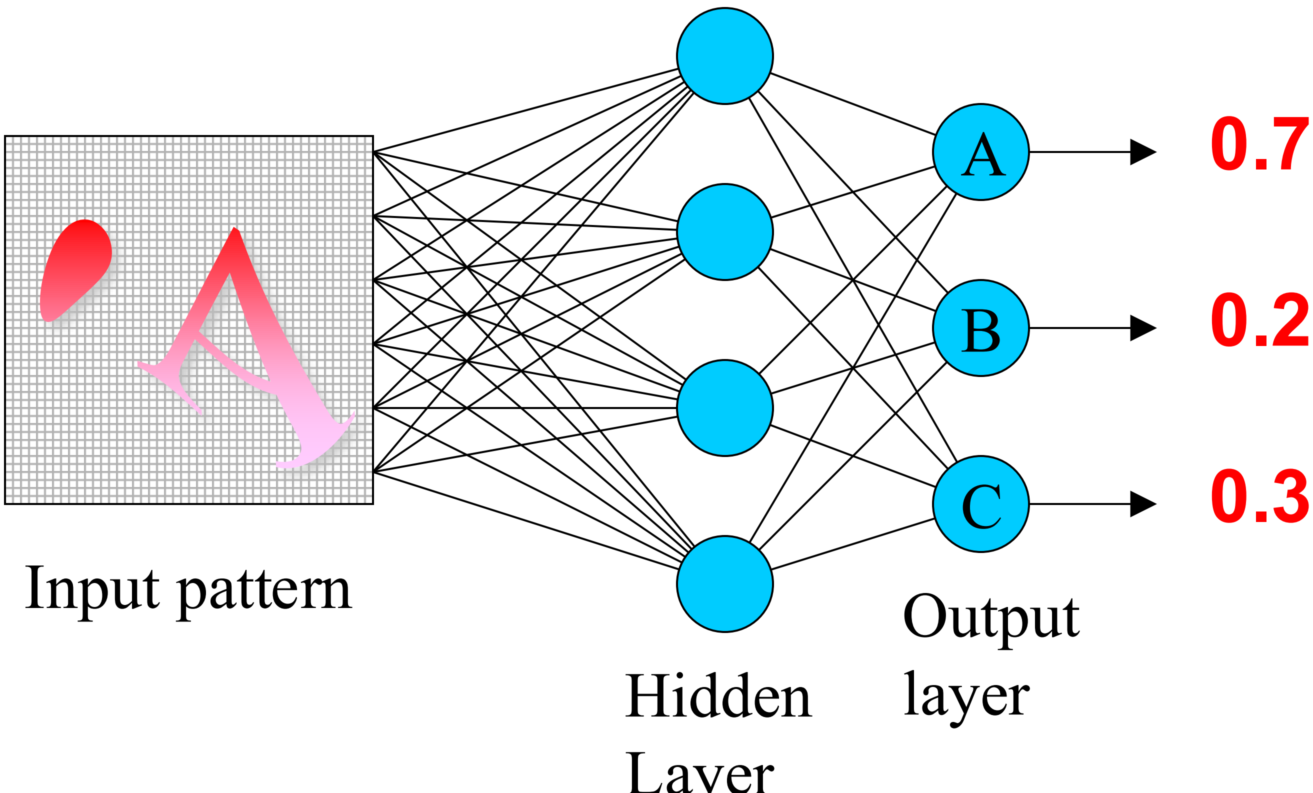
Back Propagation

(Rumelhart-Hinton-Williams, '85)

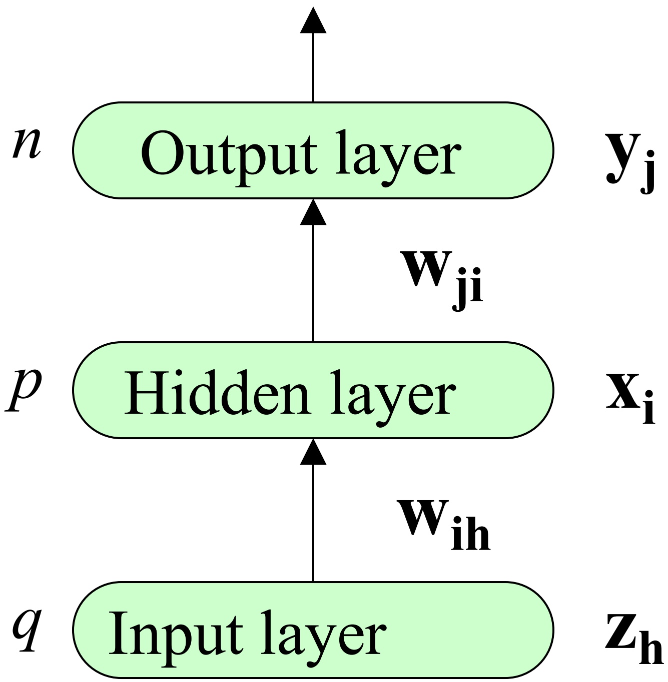
- Layered networks
- Real-valued inputs $\in [0,1]$
- Neurons with nonlinear sigmoid output function (it must be differentiable):



Letter Recognizer



Back Propagation: definitions



$\mathbf{t_j}$ desired output

error on example k

$$E_k = \sum_{j=1}^{n} (t_{kj} - y_{kj})^2$$

Global error

$$E = \sum_{k=1}^{M} E_k$$

Training Set

TS = {(X_k, y_{dk}), K = 1, M}

Back Propagation: aims

- **Learning**

  train the network on a set of desired associations (X_k, t_k): **Training Set (TS)**

- **Convergence**

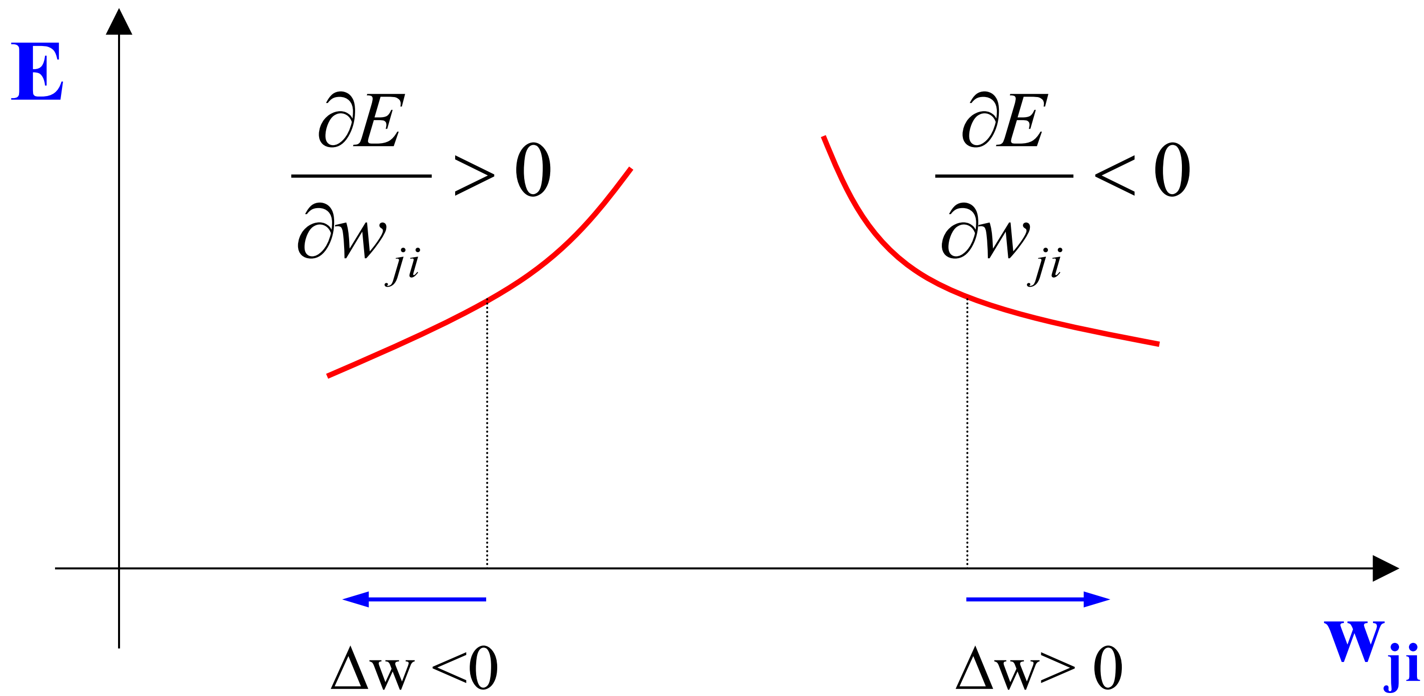
  reduce the global error **E** to variation of weights, so that $\mathbf{E < \varepsilon}$

- **Generalization**

  ensure that the network behaves well on unseen examples.

Convergence

To reduce the error on variation of weights, we adopt a **gradient descent method**:



Updating weights

Therefore, the weights are changed according to the following law:

$$\Delta \boldsymbol{w}_{ji} = -\eta \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{w}_{ji}}$$

Gradient rule

η = Learning coefficient (learning rate)

Updating weights

$$\Delta \boldsymbol{w}_{ji} = \eta \delta_j \boldsymbol{x}_i$$

For the output layer

$$\delta_j = (\boldsymbol{t}_j - \boldsymbol{y}_j) \boldsymbol{f}'(\boldsymbol{net}_j)$$

For the hidden layer

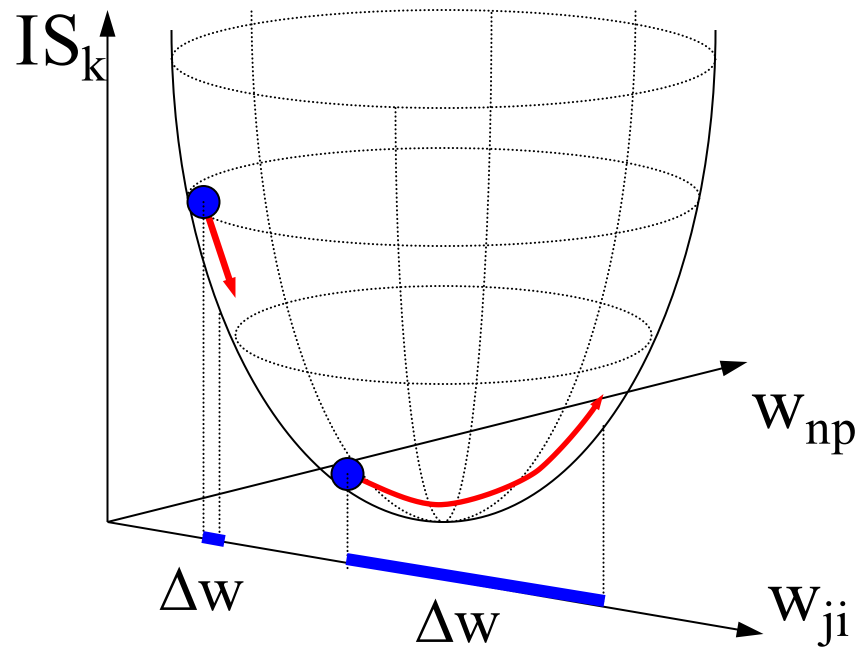
$$\delta_i = \boldsymbol{f}'(\boldsymbol{net}_i) \sum_{j=1}^{n} \delta_j \boldsymbol{w}_{ji}$$

Back Propagation: Algorithm

1. randomly initialize the weights;
2. **do {**
3. initializes the global error $\mathbf{E = 0}$;
4. **for each** $(X_k, t_k) \in TS$ **{**
5. compute $\mathbf{y_k}$ and error $\mathbf{E_k}$;
6. compute $\boldsymbol{\delta_j}$ on the output layer;
7. compute $\boldsymbol{\delta_i}$ on the hidden layer;
8. update weights of the network: $\boldsymbol{\Delta w = \eta\delta x}$;
9. updates the global error: $\mathbf{E = E + E_k}$; **}**
10. **} while (E> ε)**;

Back Propagation: Remarks

- The error has a quadratic form in the space of weights:



$$\Delta w_{ji} = \eta \delta_j x_i$$

η too small $\Rightarrow$ slow learning

η too big $\Rightarrow$ fluctuations

Possible solutions

- Vary η in function of error, in order to accelerate the convergence in the beginning and reduce the oscillations in the end.
- Smooth oscillations with a low pass filter on weights:

$$\boxed{\Delta \boldsymbol{w}_{ji}(\boldsymbol{t}) = \eta \delta_j \boldsymbol{x}_i + \alpha \Delta \boldsymbol{w}_{ji}(\boldsymbol{t}-1)}$$

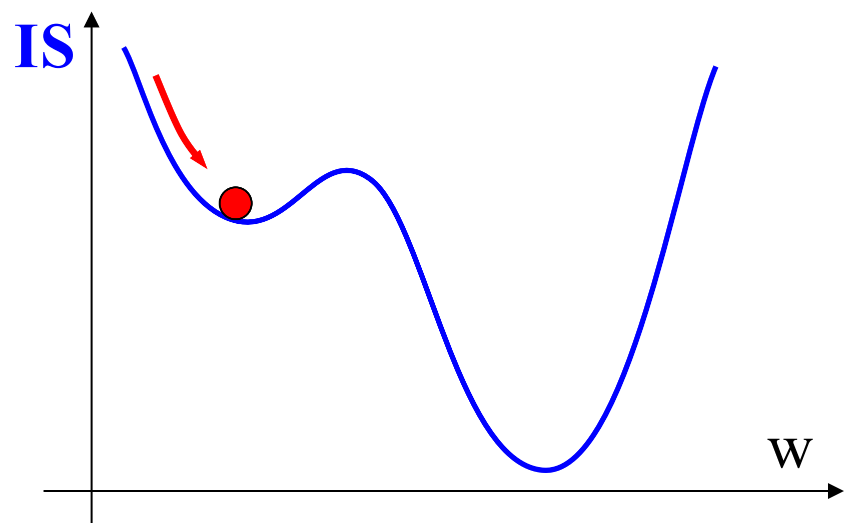
α and said **momentum**

Back Propagation: Remarks

- The quadratic form is distorted by the non-linear output function.

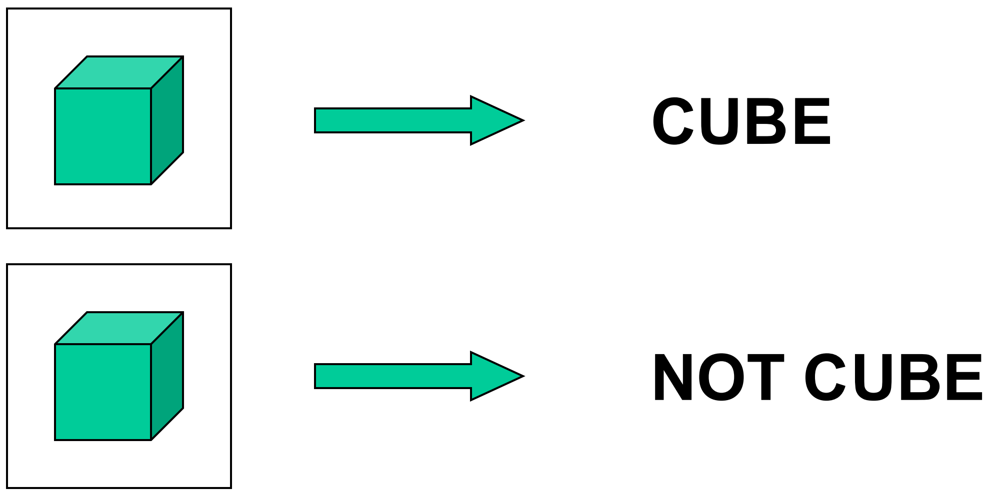
Risk of stopping in a local minimum

Restart with new weights or change some weights randomly



Back Propagation: Remarks

- If examples are inconsistent, the learning convergence is not guaranteed:



In real cases, inconsistency can be introduced by noise on the input data.

Generalization

- Generalization is the network's ability to recognize stimuli that are slightly different from those with which she was trained.

- To assess the network's ability to generalize the examples of TS, it defines another set of examples, said Validation Set (VS).

- Learning on the TS ($\mathbf{E_{TS} < \varepsilon}$),

- Evaluating the error on the VS ($\mathbf{E_{VS}}$).

Generalization

- The number of parameters to be adjusted depends on the number of hidden neurons in the network.
- A few hidden neurons may not be sufficient to reduce the global error.
- Too many hidden neurons could overfit the network on the TS specific examples.
- The network would respond well on TS, but the error would be high on other examples (overtraining).

Kohonen NETWORKS

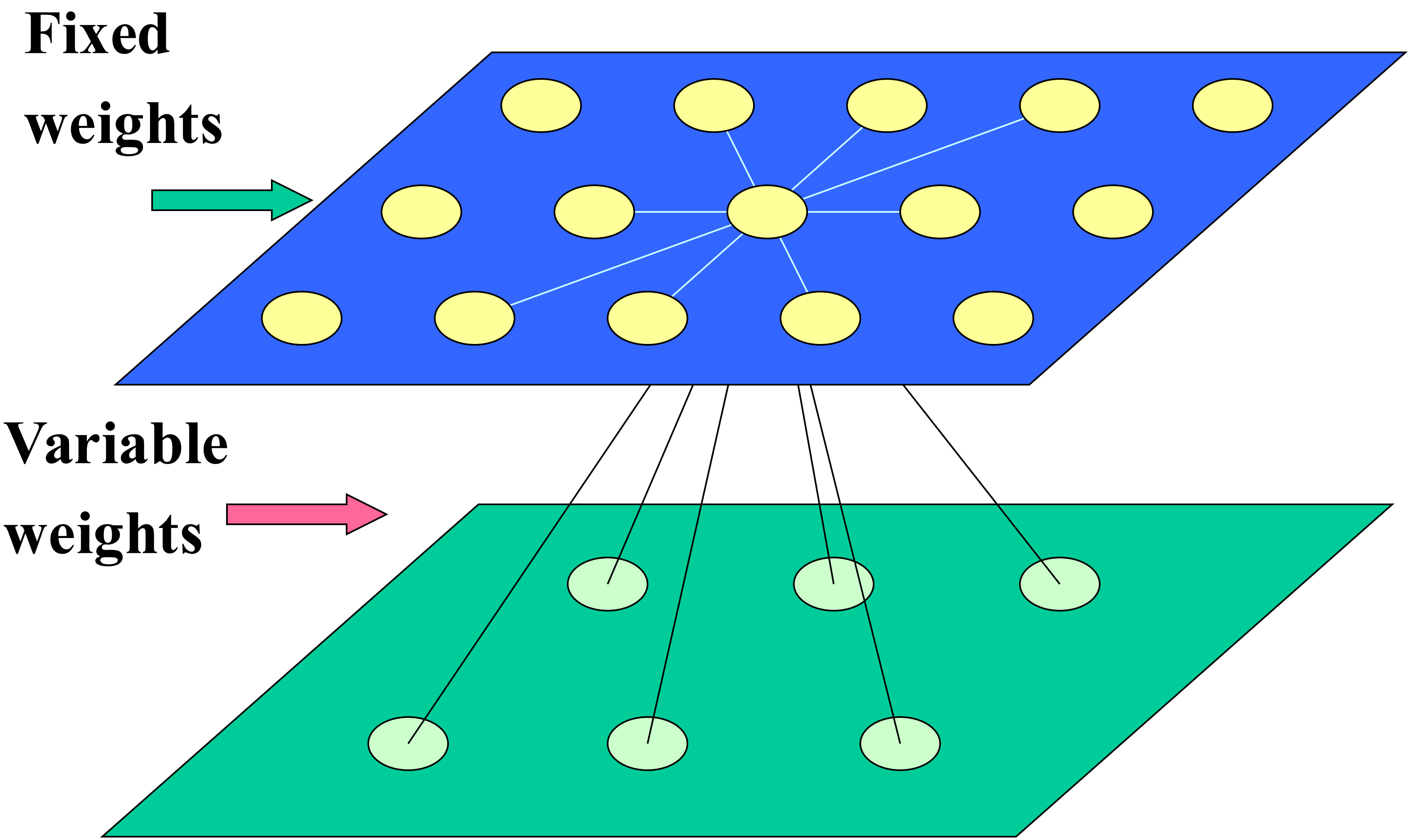
Competitive Learning
&
Self Organizing Maps

Kohonen networks

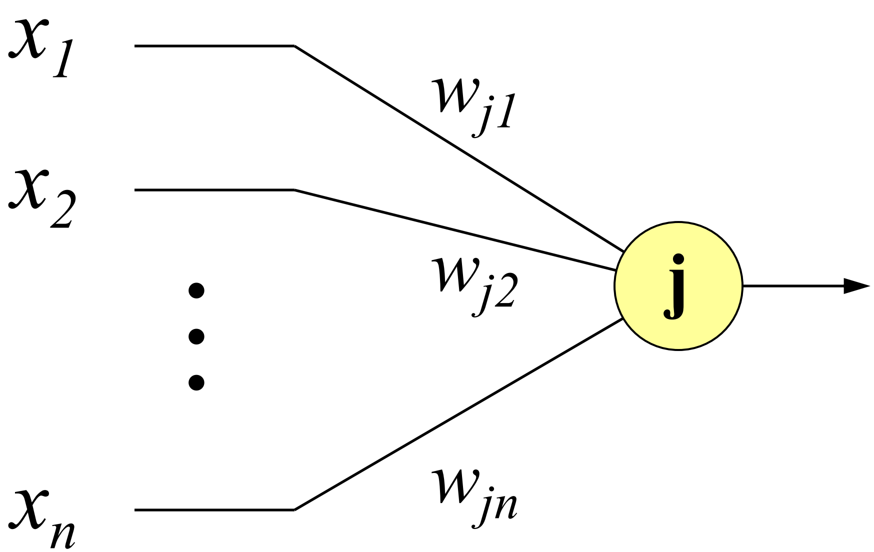
In 1983, Teuvo Kohonen managed to build a neural model that replicates the process of formation of the sensory maps in the cerebral cortex:

- layered network
- unsupervised learning based on the competition between neurons

Architecture



Linear neurons



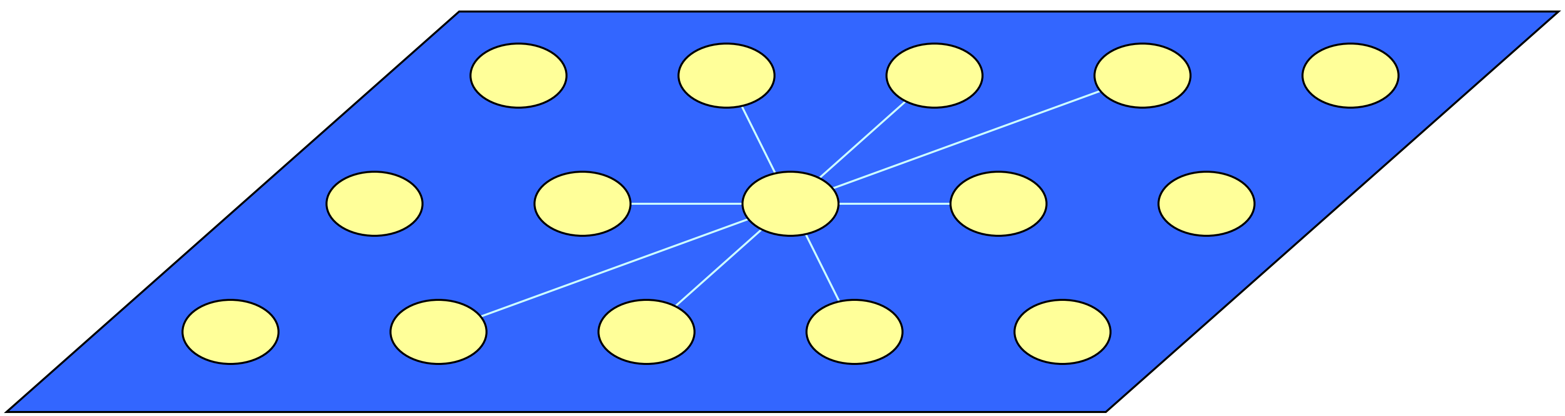
$$y_j = \sum_{i=1}^{n} w_{ji} x_i$$

$$y_j = \sum_{i=1}^{n} w_{ji} x_i = W_j \bullet X = |W_j| \bullet |X| \cos\theta$$

Distribution of fixed weights

The fixed weights depend on the distance of the neurons:

- **neighboring neurons ⇒ positive weights**
- **distant neurons ⇒ negative weights**



Competitive learning

- Neurons compete to respond to a stimulus.
- The neuron with greatest output wins the competition and specializes to recognize the stimulus.
- Thanks to the excitatory connections, the neurons close to the winner are sensitive to similar inputs.

Isomorphism between the input space and output space

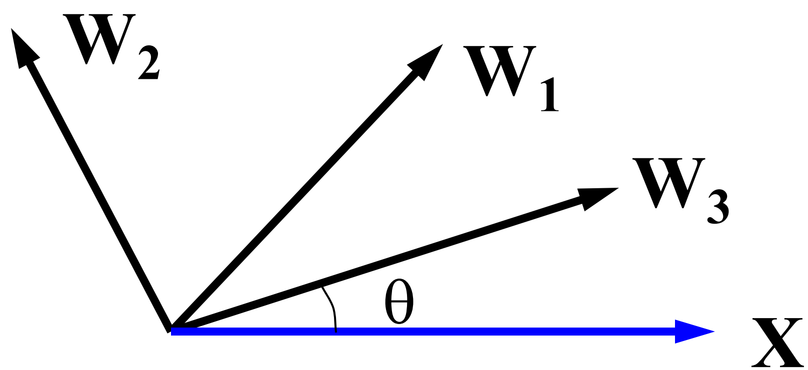
Implementation

- For efficiency reasons, output neurons are not connected together.
- The winner neuron is chosen with an overall strategy comparing the outputs of all neurons.
- We can use two techniques:
    1. Choose the neuron with maximum output;
    2. Choose the neuron whose weight vector is the most similar to the stimulus.

Winning neuron (Method 1)

The winning neuron on input X is the one with the highest output:

$$y_j = \sum_{i=1}^{n} w_{ji} x_i = W_j \bullet X = |W_j|\,|X| \cos\theta$$



Definition of distance

Euclidean distance:

$$DIS(X,Y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Manhattan Distance:

$$DIS(X,Y) = \sum_{i=1}^{n}\left|x_i - y_i\right|$$

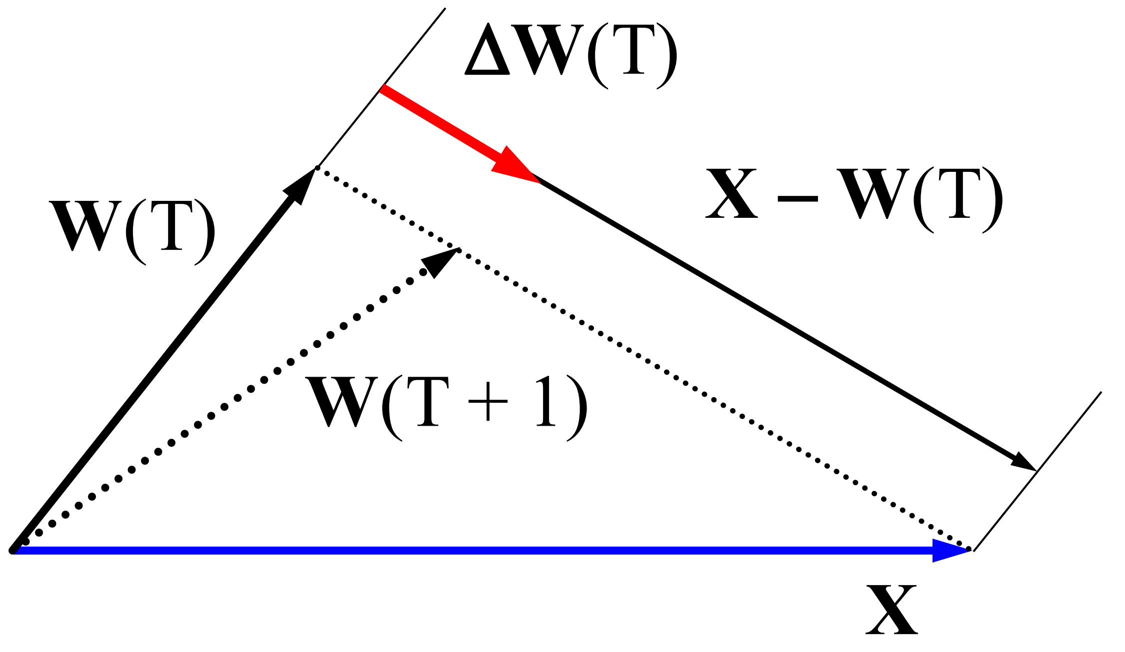
Hamming distance:

(For binary vectors only)

$$DIS(X,Y) = \sum_{i=1}^{n}(x_i != y_i)$$

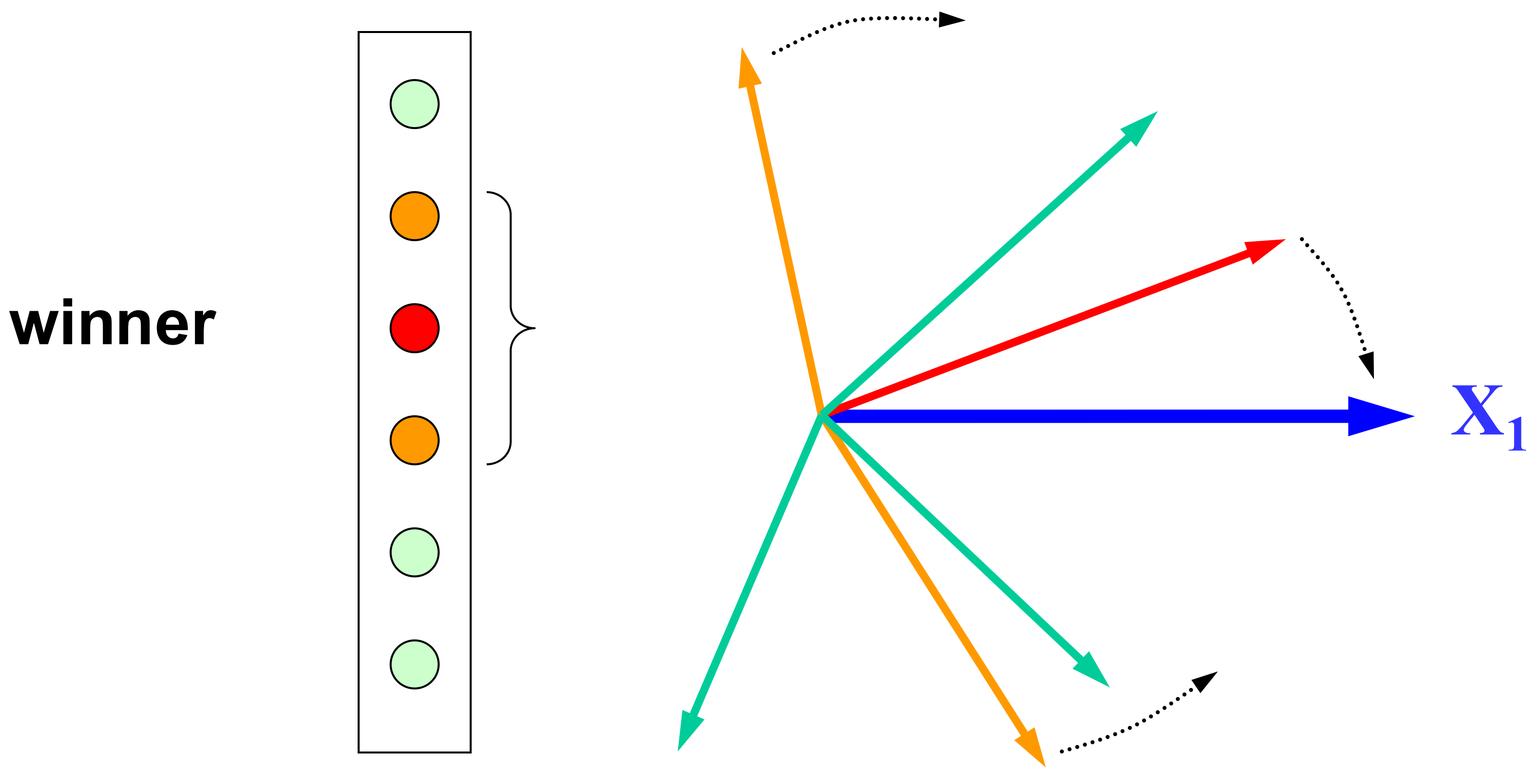
Learning law

$$\Delta \mathbf{W}(\mathrm{T}) = \alpha\,(\mathbf{X} - \mathbf{W})$$



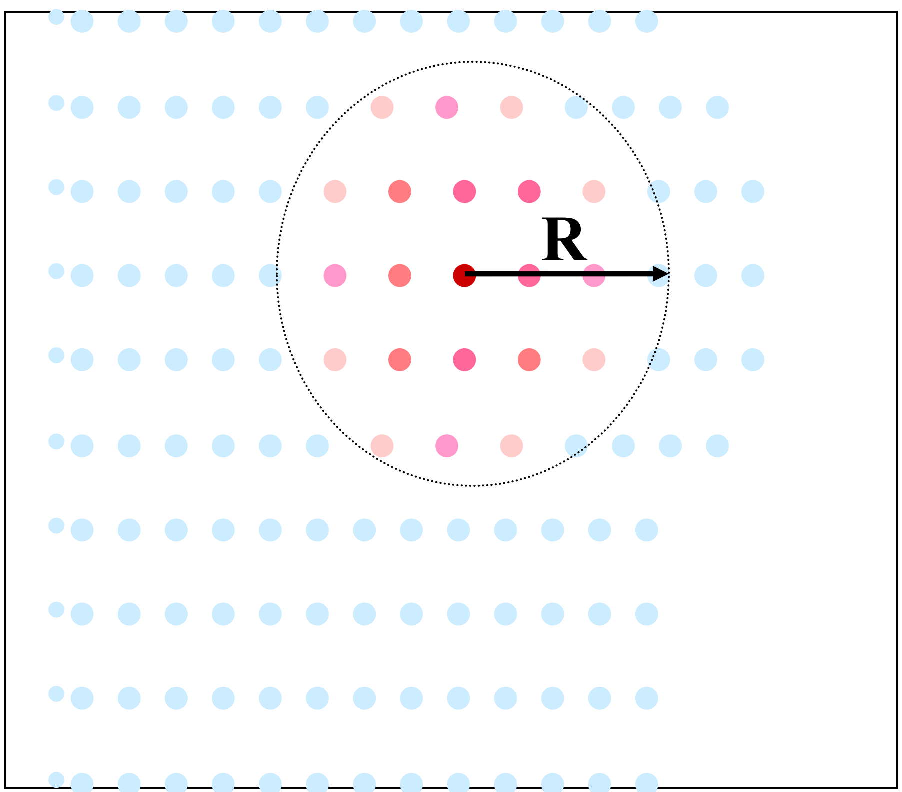
Neighborhood update

To simulate the radial connections, neurons close to the winner have an updated weight:



Interaction radius

The neighborhood is the set of neurons within a given distance R from the winning neuron:



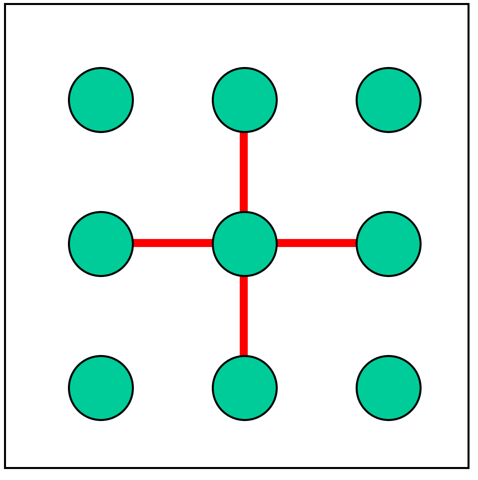
R = radius of interaction

Defining the map

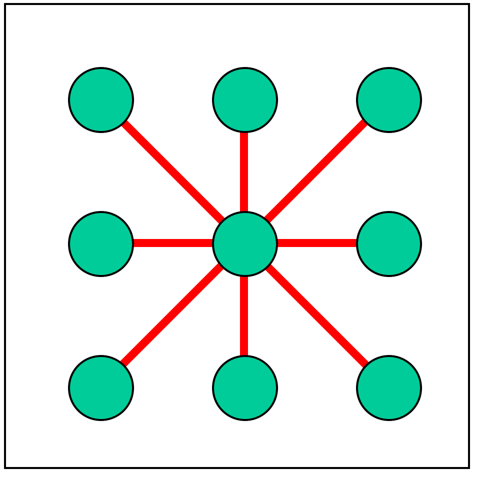
- To allow the formation of the map, it is necessary to define a topology on the output layer.
- Each neuron has to have a position identified by a vector of coordinates.
- The output map is usually defined as a to one or two dimensional space.

Neighborhood types

proximity 4



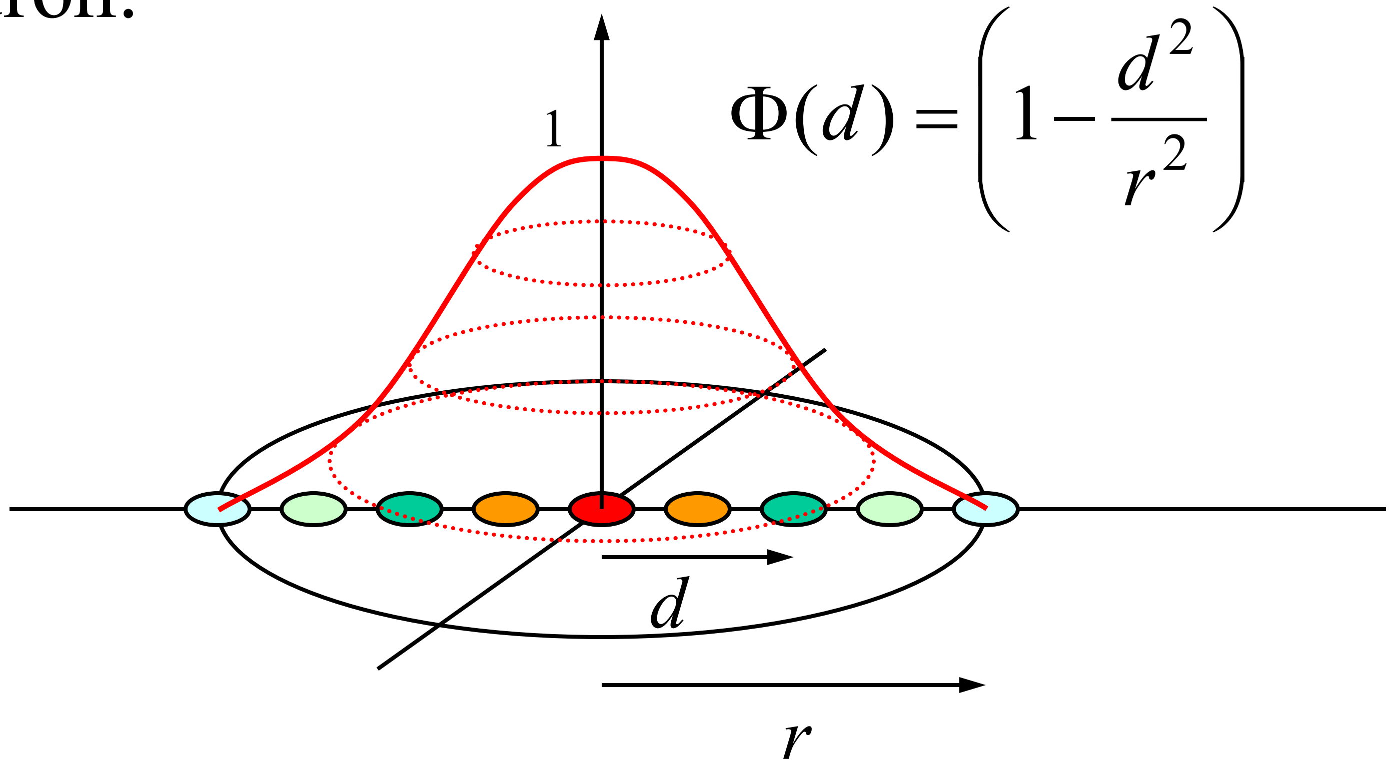
proximity 8



Weight change

The weights of the neurons are varied according to their distance from the winner neuron:

$$\Phi(d) = \left(1 - \frac{d^2}{r^2}\right)$$



Weight change

Given j_0 the index of the winner neuron, we have:

$$\forall j \in \text{neighborhood}\,(j_0, r)$$

$$d = DIS(j, j_0)$$

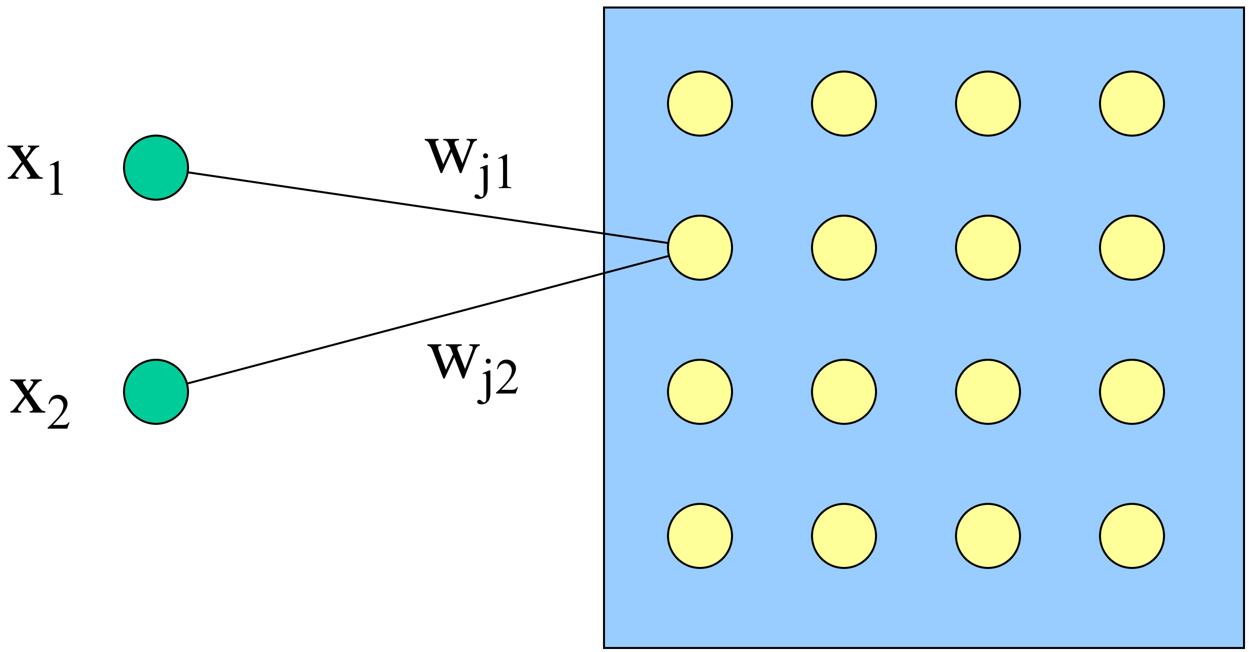
$$\Delta w_j = \alpha\, \Phi(d)\, (X - W_j)$$

Quantities **r** is **α** decrease over time.

Learning algorithm

1. Randomly initialize the weights;
2. Initialize the parameters: $\alpha = A$; $r = R$;
2. **do** {
4. **for each** ($X_k \in TS$) {
5. Compute all the outputs y_j;
6. Chose the winner neuron j_0;
7. update the weights of the neighborhood;
8. }
9. reduce α and r;
10. } **while** ($\alpha > \alpha_{min}$);

Example



Input = vector of coordinates on a two-dim space

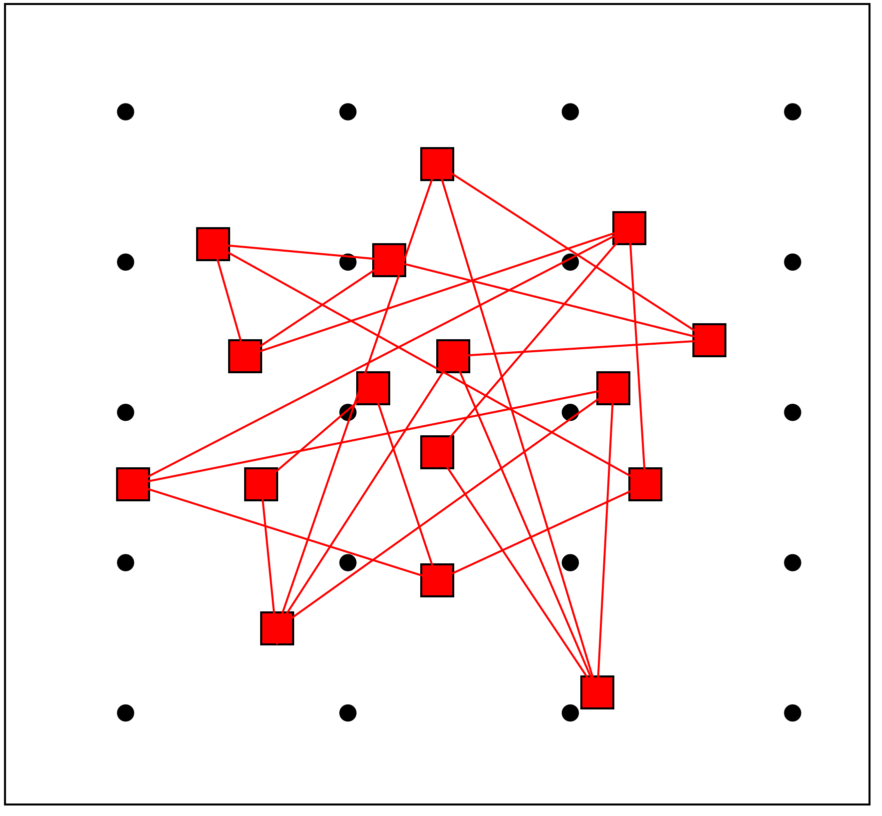
Map = grid with proximity 4

Initial State

N = M

- Input (stimulus)

■ Weight (neurons)



$\begin{cases} \text{M inputs} \\ \text{N neurons} \end{cases}$

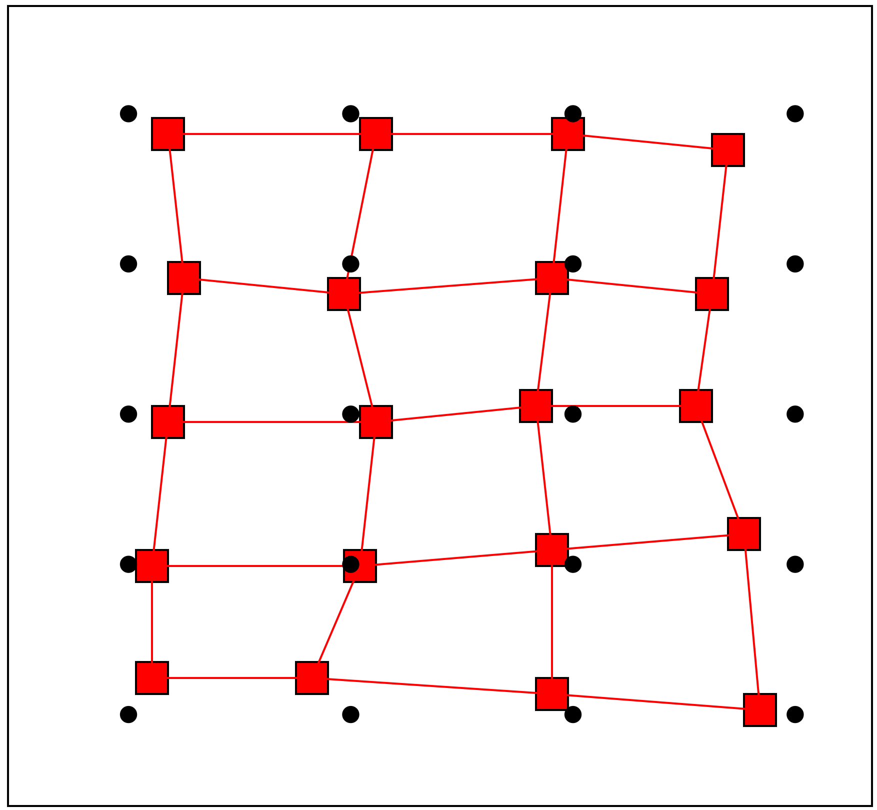
Final state

N = M

- input

weight

M inputs
N neurons



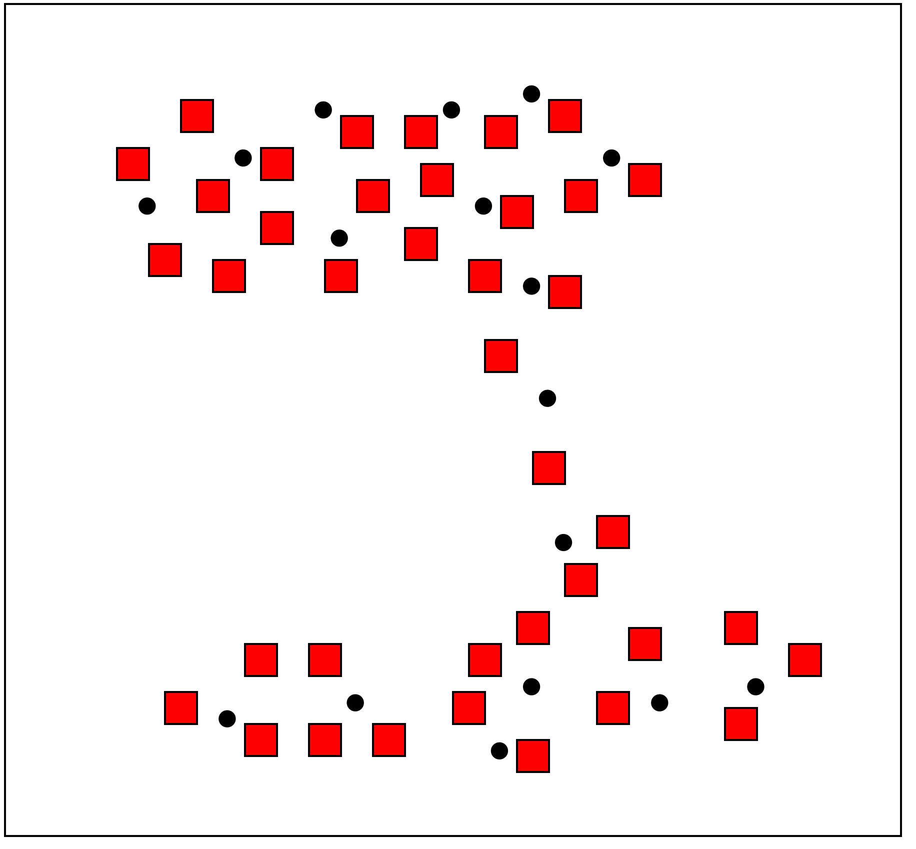
Example

N> M

• input

■ weight

M inputs
N neurons



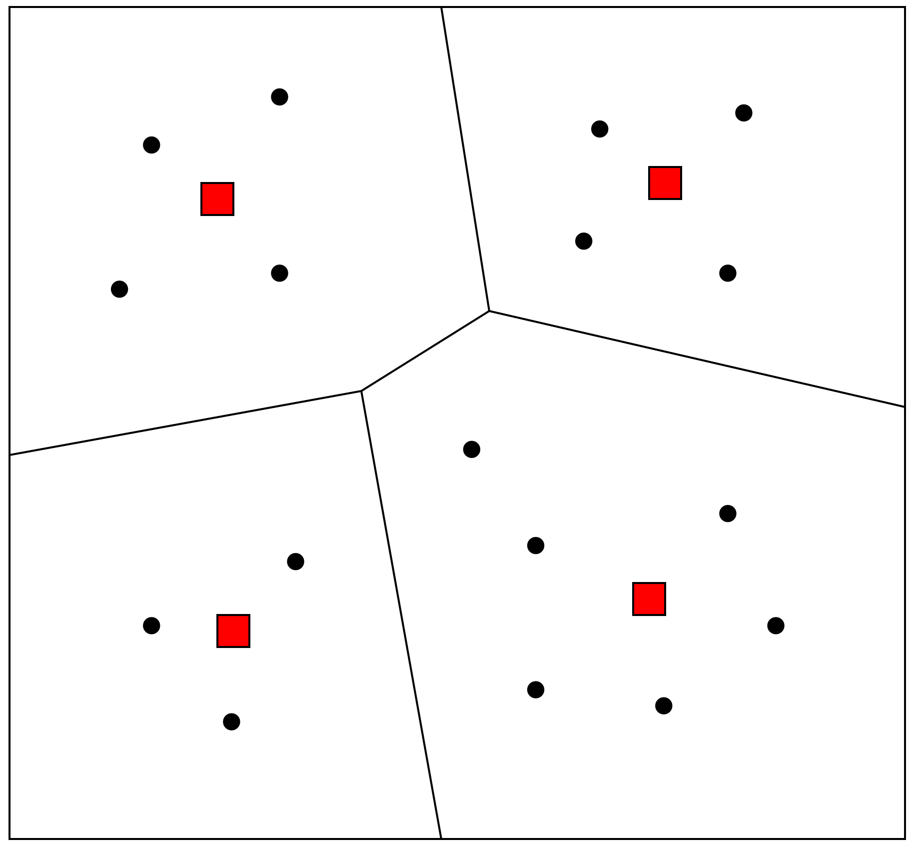
Example

N <M

• input

■ weight

M inputs
N neurons



Applications

Clustering

- Group a huge set of data in a limited number of classes, based on the similarity of the data.

Compression

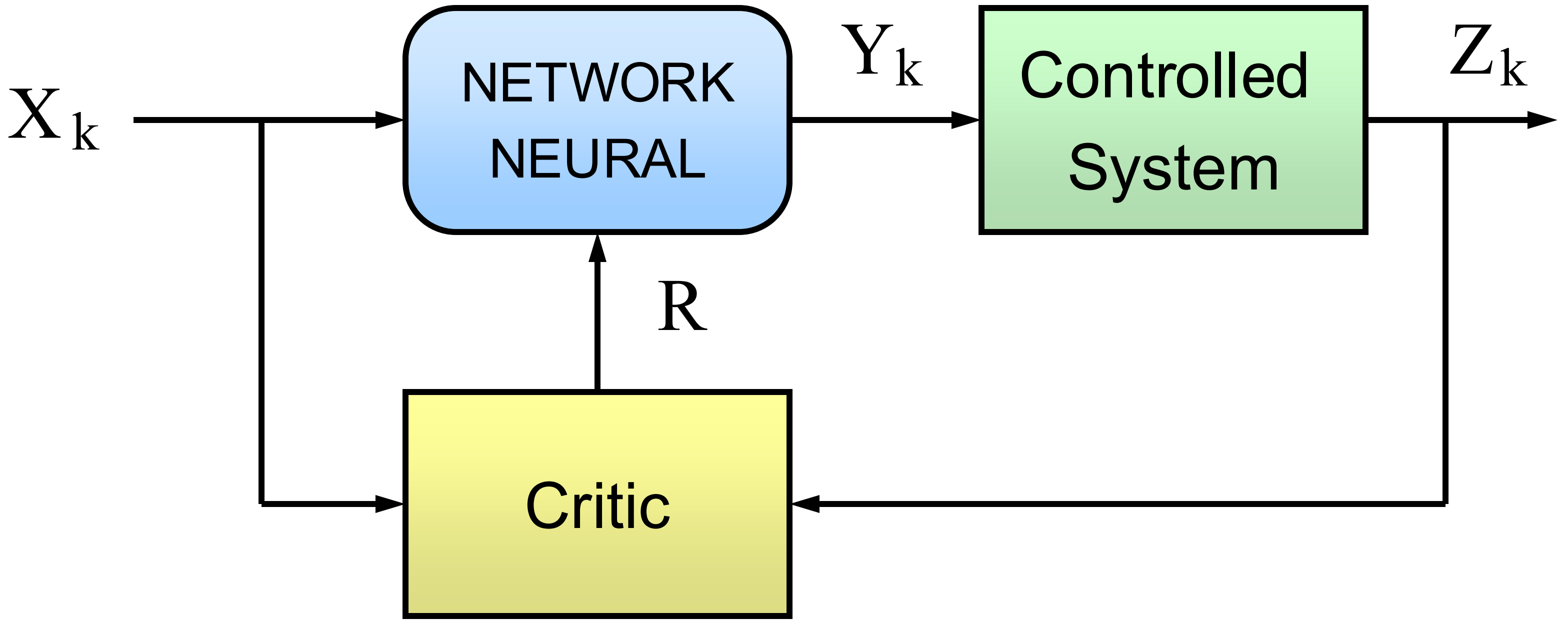
- convert an image with millions of colors in a compressed image to 256 levels (not fixed).

Classification

- Classify a set of sentences in a set of separate classes for related topics.
- Used by some search engines to rank the preferences of connected users.

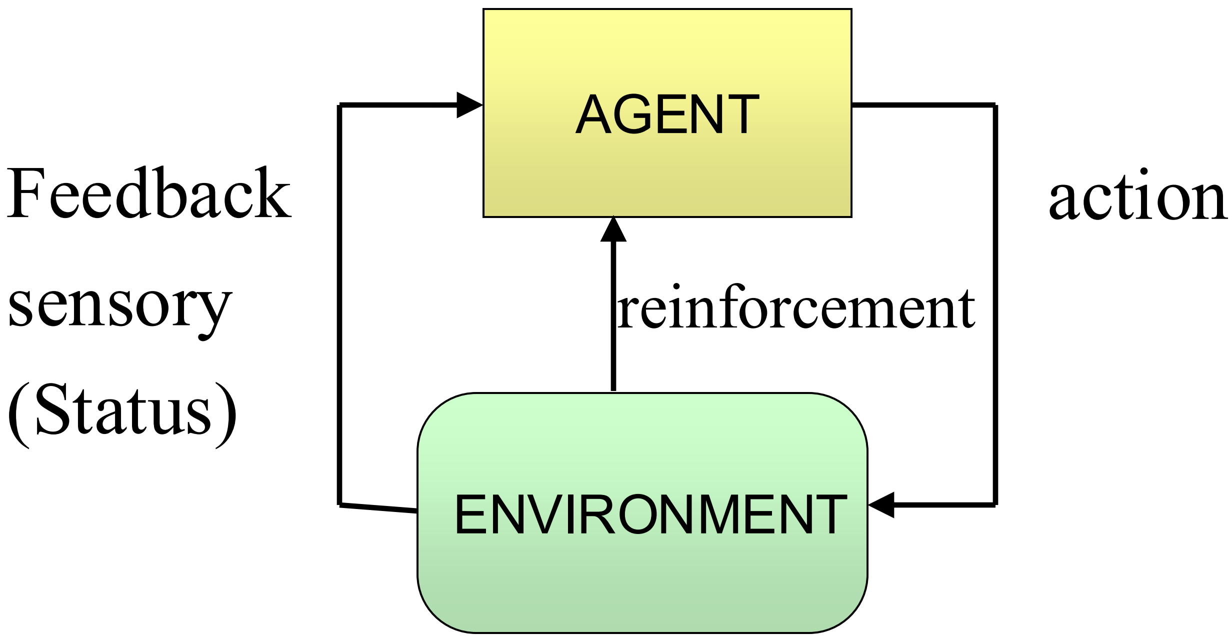
Reinforcement learning

Reinforcement learning



Rewards and punishments

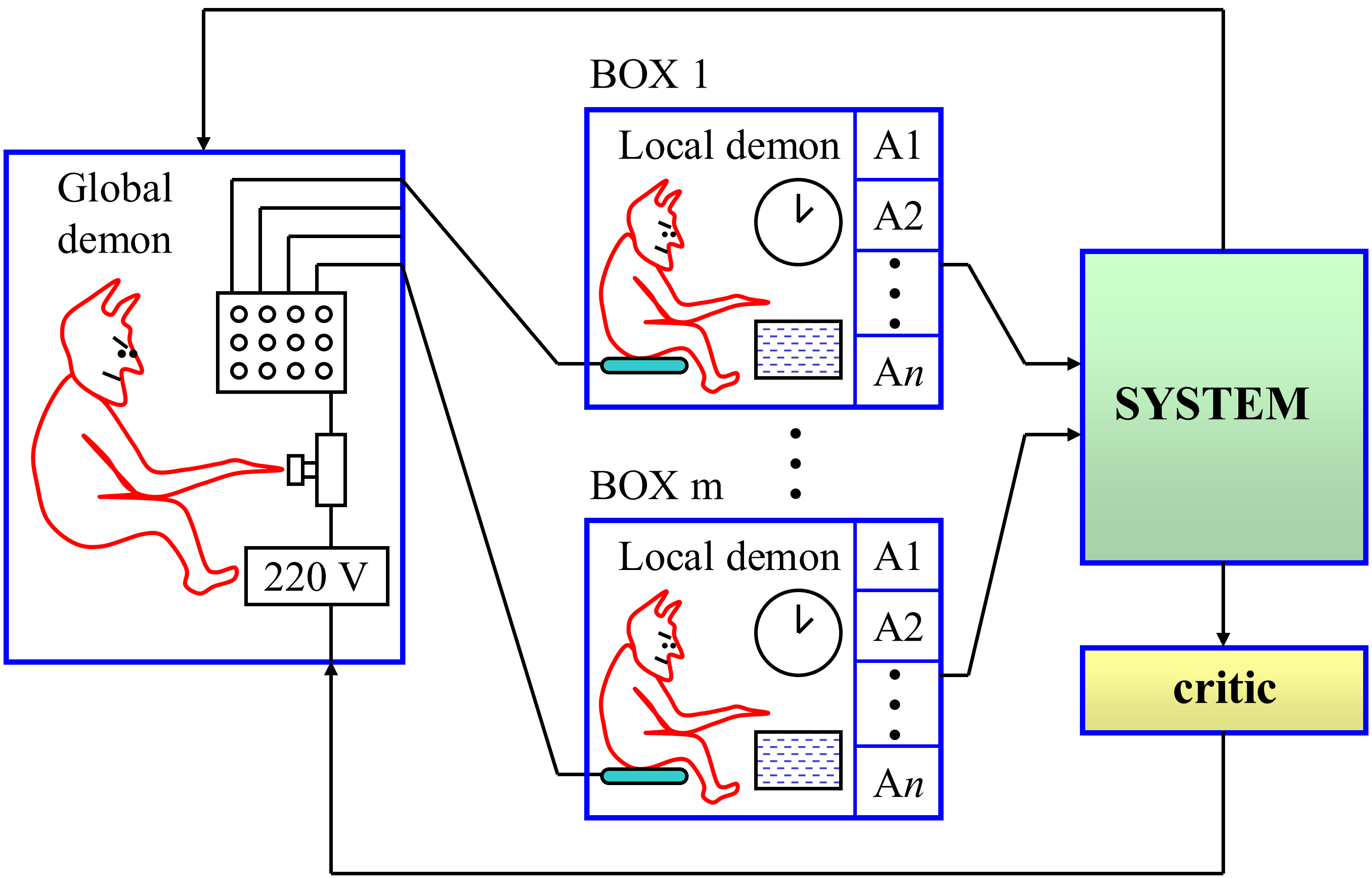
An **agent** operates in the environment and adapts the actions on the basis of the produced consequences.



Boxes Model (Michie / Chamber '68)

- Learning based on punishment.
- It partitions the state space into N disjoint regions (box).
- Whenever the system enters a state (box) a control action is selected.
- The controller has to learn to perform the actions which delay the punishment as much as possible.

Model boxes

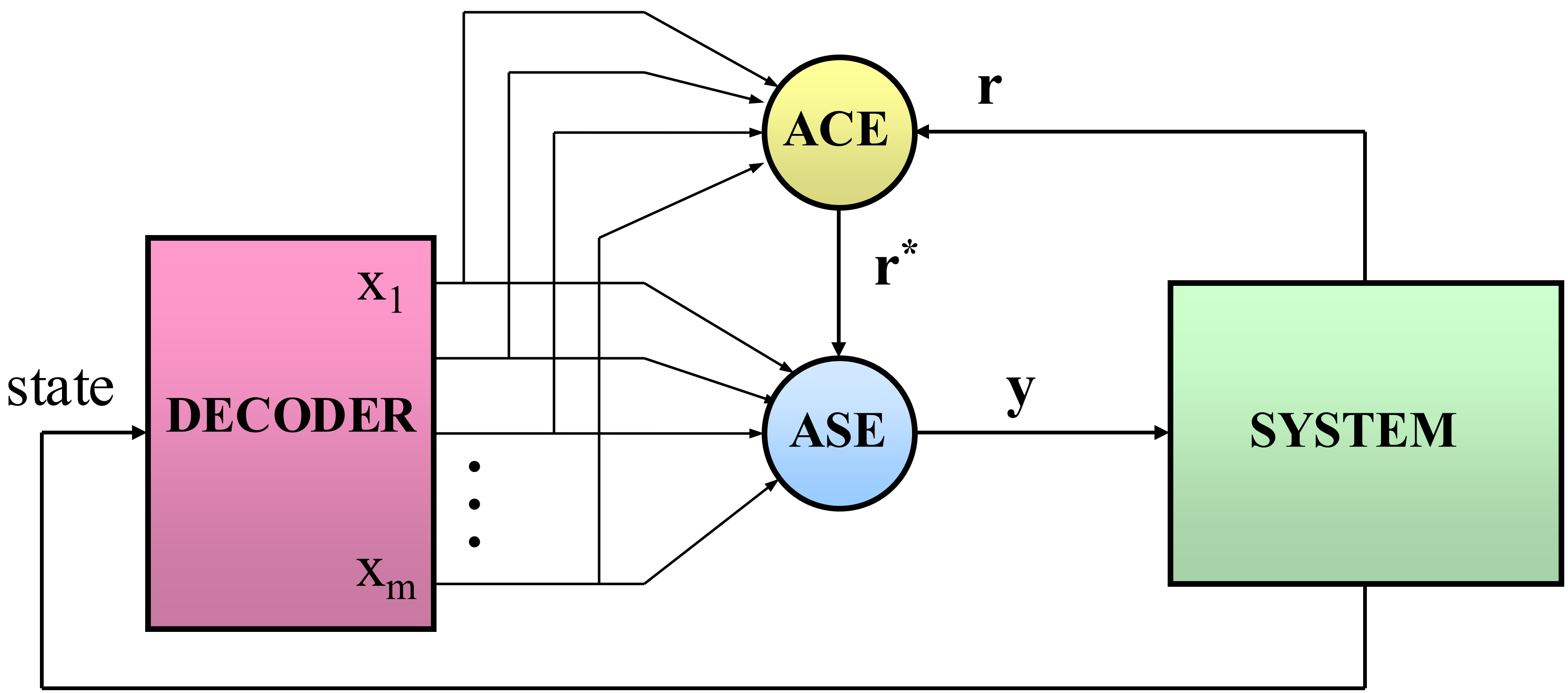

BOX 1
Local demon
A1
A2
An
Global
demon
220 V
SYSTEM
BOX m
Local demon
A1
A2
An
critic

Neural Model: ASE-ACE

(Barto-Sutton-Anderson, '83)

ASE: Associative Search Element

ACE: Adaptive Critic Element

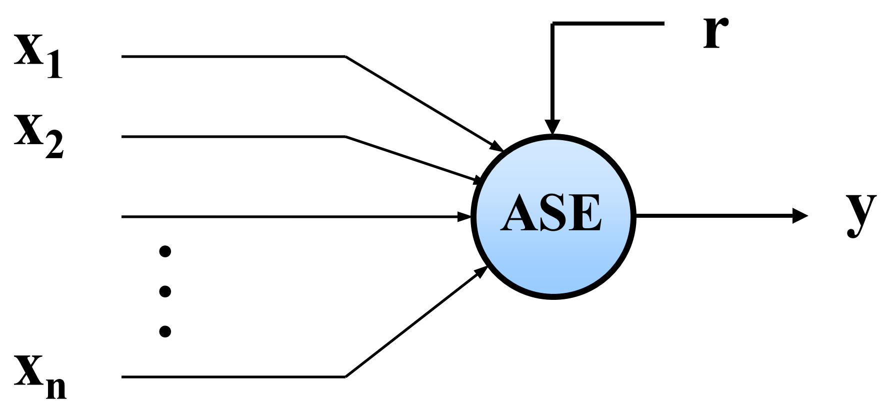


Adaptive Search Element

in the system state $\mathbf{x_i}$

$$x_i = 1$$

$$x_j = 0 \quad \forall\, j \neq i$$



$$y(t) = \operatorname{sgn}\left[\sum_{i=1}^{n} w_i x_i + n(t)\right]$$

n (t) is a Gaussian variable with zero mean and variance σ^2

Adaptive Search Element

$$y(t) = \operatorname{sgn}\left[\sum_{i=1}^{n} w_i x_i + n(t)\right]$$

ASE can only generate two actions: $\begin{cases} \mathbf{a}^{+} & (Y = 1) \\ \mathbf{a}^{-} & (Y = 0) \end{cases}$

When the system is in the state $\mathbf{x_i}$ ($\mathbf{x_i} = 1$) we have that:

if $\mathbf{w_i = 0}$, actions $\mathbf{a}^{+}$ and $\mathbf{a}^{-}$ have the same probability

if $\mathbf{w_i > 0}$, the $\mathbf{a}^{+}$ choice is more likely to be performed

if $\mathbf{w_i < 0}$, the $\mathbf{a}^{-}$ choice is more likely to be performed

Adaptive Search Element

The weights of ASE are updated with the following law:

$$\Delta w_i(t) = \alpha\, r(t)\, e_i(t)$$

α It is a positive constant (learning rate)

r (t) is the reinforcement signal

$$r(t) = \begin{cases} -1 & \text{in case of failure} \\ 0 & \text{otherwise} \end{cases}$$

$e_i(t)$ is a signal (eligibility) introducing a short-term memory on synapses:

$$e_i(t+1) = \delta\, e_i(t) + (1 - \delta)\, y(t)\, x_i(t)$$

Adaptive Search Element

$$\Delta w_i(t) = \alpha\, r(t)\, e_i(t)$$

$$e_i(t+1) = \delta\, e_i(t) + (1 - \delta)\, y(t)\, x_i\, (t)$$

A failure (**$r < 0$**) reduces the probability of chosing recent actions that have caused it:

$$\mathbf{a^+} \Rightarrow \mathbf{e_i(t) > 0} \Rightarrow \mathbf{\Delta w_i < 0}$$

$$\mathbf{a^-} \Rightarrow \mathbf{e_i(t) < 0} \Rightarrow \mathbf{\Delta w_i > 0}$$

Adaptive Search Element

$$\boxed{\Delta w_i(t) = \alpha\, r(t)\, e_i(t)}$$

$$\boxed{e_i(t+1) = \delta\, e_i(t) + (1 - \delta)\, y(t)\, x_i\,(t)}$$

A success (**$r > 0$**) increases the probability of selecting the recent actions that have caused it:

$$\mathbf{a^{+} \;\Rightarrow\; e_i(t) > 0 \;\Rightarrow\; \Delta w_i > 0}$$

$$\mathbf{a^{-} \;\Rightarrow\; e_i(t) < 0 \;\Rightarrow\; \Delta w_i < 0}$$

ACE role

- As the number of failures is reduced, the system tends to learn more slowly.
- The adaptive critic element (**ACE**) generates a more informative **secondary reinforcement**.
- Observing the system status and failures, the ACE learns to **predict** dangerous conditions.
- It generates an award ($\mathbf{r^* > 0}$) if the system moves away from a dangerous state, punishment ($\mathbf{r^* < 0}$) otherwise.

Learning - conclusions

Network capacity to change behavior in a desired direction by changing synaptic connections (weights).

The learning paradigms can be divided into three basic classes:

- **supervised**
- **competitive**
- **reinforcement**