University of Verona

A.Y. 2021-22

# Machine Learning & Artificial Intelligence

## Artificial Neural Networks

Vittorio Murino

## Source material

- Reference book:

    C.M. Bishop: “Neural Networks for Pattern Recognition”, Clarendon Press 1995

## Non-algorithmic approaches to information processing

- Given a problem to be solved P, it can often be formalized by {P, C, f},
  - P = problem formulation
  - C = $\{c_1,\ldots,c_n\}$ set of configurations, each of which represents a possible solution
  - f: C $\rightarrow$ R function that provides a measure of the goodness of the configurations with respect to the purpose of solving the problem.

- Solving the problem means maximizing or minimizing the f function in the C space

## Examples

- Problem 1: “sort a set of numbers in a growing way $x_1 \ldots x_n$”.
  - In this case the formalization is simple
    - P = sort the numbers $x_1 \ldots x_n$;
    - C = set of n! permutations of $x_1 \ldots x_n$;
    - f = function that adds up the distances between each number and the next
- Problem 2: “Predict the stock market index tomorrow”
  - Formalization is more difficult.

## Algorithmic solution to problems

- Find an algorithm that solves the problem
- The algorithm itself represents the solution: given the algorithm always finds the solution
- Example: Number sorting problem, the *Bubble Sort* algorithm itself is the solution to the problem:
    - given a set of numbers, following step by step the instructions contained in the description of the algorithm, you get to the solution, that is, the set of numbers sorted.

- Question: Is there always an algorithmic solution?

- Answer: Yes!
  - It is the *Exhaustive Research*, or brute force approach:
  - Calculate f(c) for each possible configuration, and choose the one that maximizes the goodness function f
  - Problem: Obviously this is not applicable if C space is large

## Non-algorithmic processing

- Problem 2 cannot be solved algorithmically (except by brute force):
  - incomplete information
  - strict lack of specification of the problem
- A non-algorithmic information processing approach can then be used:
  - provide an encoding of the C configurations and the f function, and a set of rules that make the system evolve towards the solution (given the algorithm, you do not have the solution)
  - uses information that is typical of the problem
  - can use solution instances

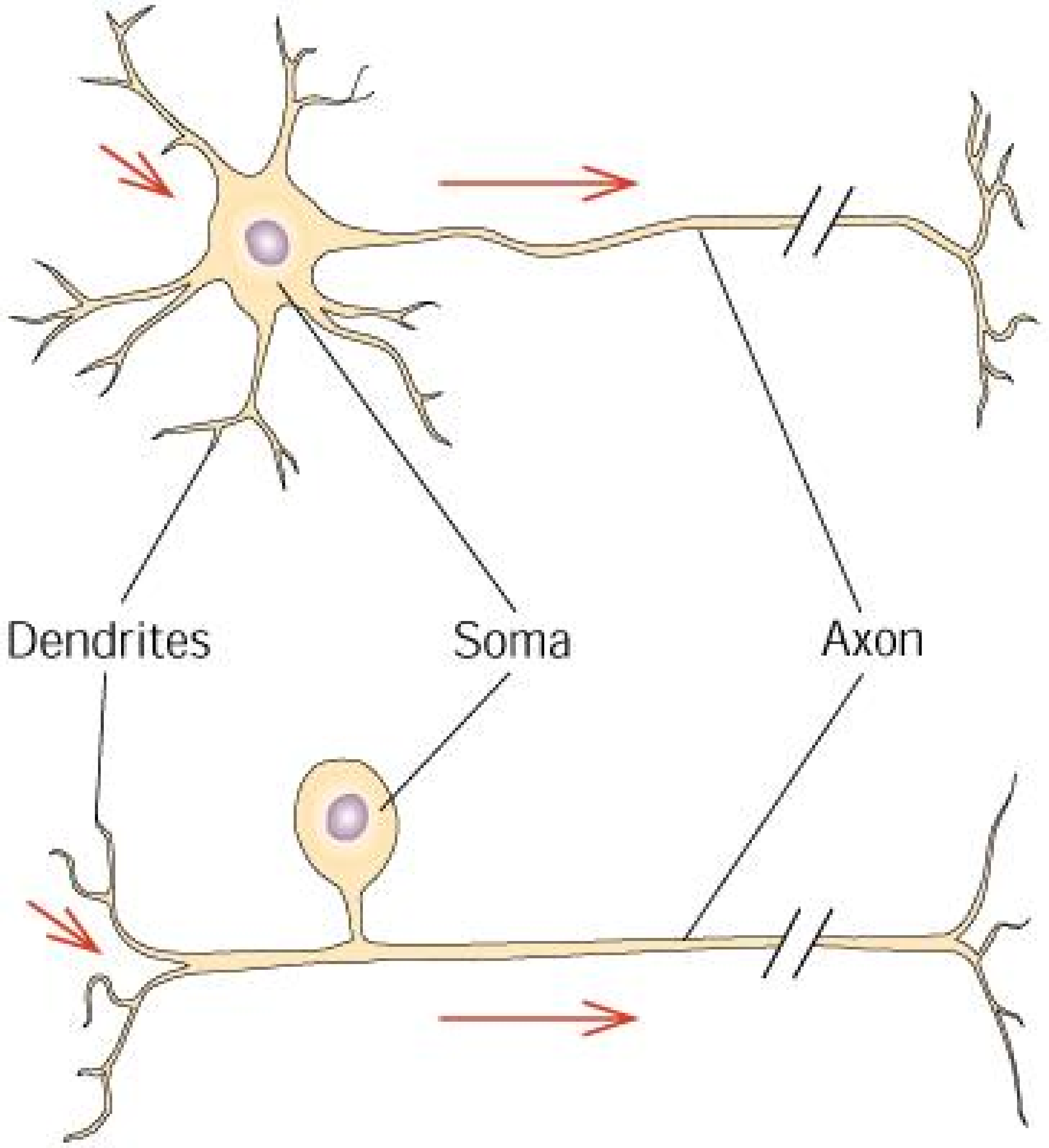
## Examples

- Genetic Algorithms:
  - is based on the theory of the evolution of the species
  - the goodness of an individual is linked to his "adaptability".
- Simulated Annealing:
  - it is based on physical models: by gradually lowering the temperature T of a system, it tends to crystallize in a configuration of minimal energy
- Neural Networks:
  - Artificial information processing system that aims to emulate the animal nervous system, i.e., to emulate the biological computation system.
  - they try to solve problems starting from examples of solutions.

## Neural networks

- The animal nervous system has several features that are attractive from the point of view of a calculation system:
  - it is robust and resistant to failures: every day some neurons die without the overall performance suffering a substantial deterioration;
  - it is flexible: it adapts to new situations by learning;
  - allows a highly parallel computation;
  - It is small, compact and dissipates little power.
  - can also work with information
    - **approximate:** the information represented by the signal is not described exactly
    - **incomplete:** the signal may not arrive in part
    - **affected by error:** if the signals are affected by error the system must be able to regenerate them (neglect the errors)

## The neuron



## The neuron

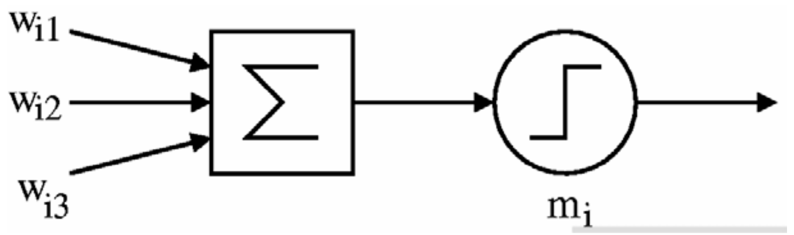
- The flow of information is unidirectional:

  Dendrites $\longrightarrow$ soma $\longrightarrow$ axon

- The neuron receives input (potentials) from the dendrites, weighing them in some way (synapses).
- If the sum of these inputs exceeds a certain threshold, the neuron turns on (emits a signal 1).

## A bit of history …

- 1943 - McCulloch e Pitts:
  - o first neuron model: biological model



  - o weighted sum of inputs: if they exceed the threshold $m_i$, then the output is 1, otherwise it is 0
- 1949, Hebb:
  - o from studies on the brain, it emerges that learning is not a property of neurons, but is due to a modification of synapses

## A bit of history …

- 60s: boom of work on neural networks, in Rosenblatt's group:
  - weight calculation
  - work on perceptrons, networks in which neurons are organized into sequential processing layers
- 1969, Minsky e Papert:
  - demonstrate the limits of the perceptron: enthusiasm about neural networks collapses.
- 70s: *associative content-addressable memory:*
  - associative networks, where similar patterns were somehow associated

## A bit of history …

- 1982, Hopfield:
  - proposes a network model to create associative memories.
  - a concept of energy function is introduced
  - the network always converges towards the nearest stable point of energy
- 1985, Rumelhart, Hinton e Williams:
  - formalize the learning of neural networks with supervision: introducing Back-Propagation, the standard method for training neural networks

## Basic scheme

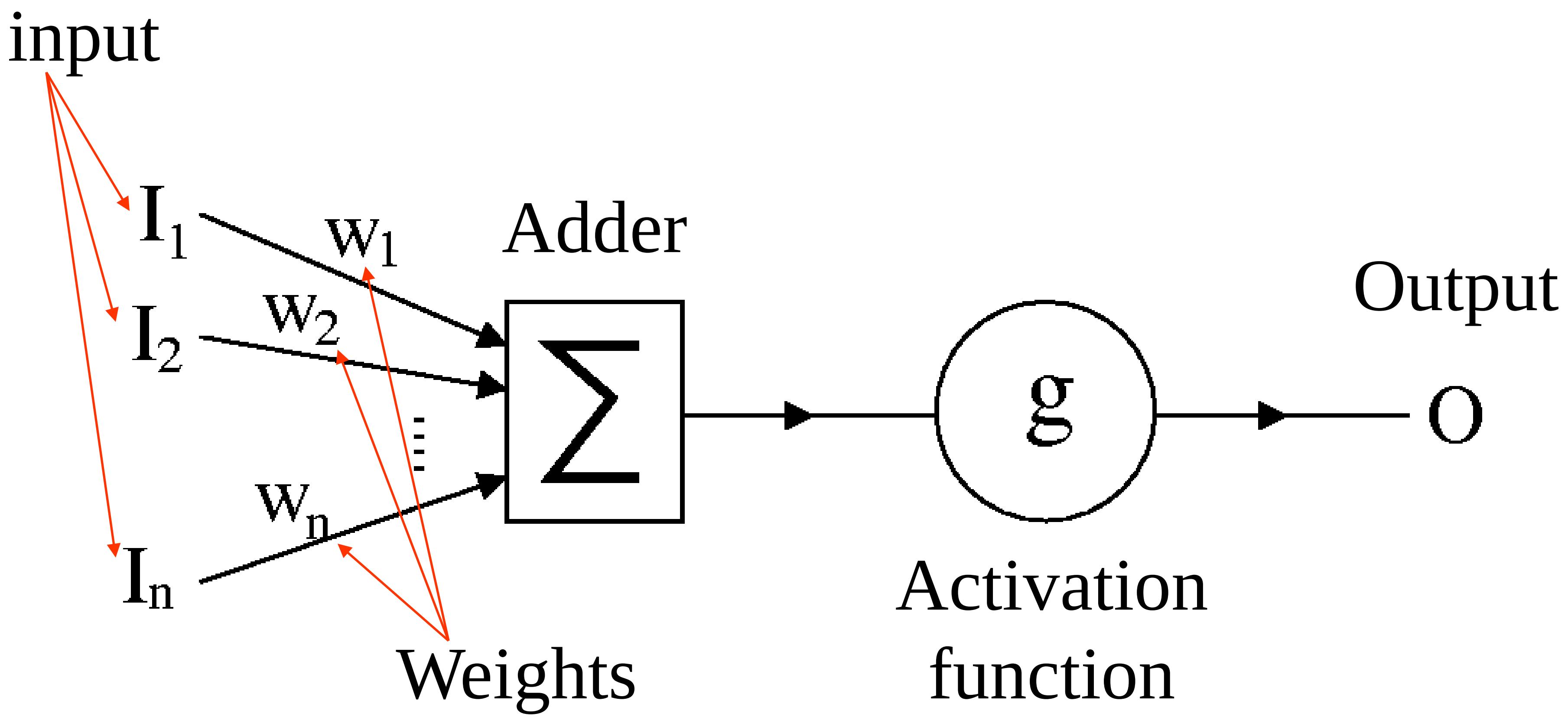
- Neural Network:
  - complex structure
  - composed of many elementary units of calculation, called *neurons*
  - neurons are connected to each other through weighted connections, called *synapses*

- There are neurons that are connected to the external environment (input or output)

- Every neuron has:
  - a set of inputs from other neurons
  - a set of outputs to other neurons
  - an activation level
  - a system to calculate the activation level at the next time

- Different neural networks differ by:
  - type of neurons
  - type of connections between neurons

## The neuron


input
$I_1$
$I_2$
$I_n$
$w_1$
$w_2$
$w_n$
Adder
$\sum$
g
Output
O
Weights
Activation function

## The neuron: components

- *Input* $I_i$: input signals to the neuron:
  - original input (derived from the problem)
  - outputs of other neurons
- *Weights* or *Synapses* $w_i$: each input is weighed by the weight of the connection: provides a measure of how much this input is relevant for the neuron
- *Adder* Σ: module that performs the weighted sum of the inputs

- *Activation function g:* function that determines the output of the neuron based on the output of the adder.

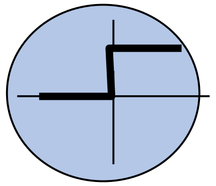
Summing up, the O output of the neuron is obtained as

$$O = g\left(\sum_{i=1}^{n} w_i I_i\right)$$

## The *Perceptron*

- In the simplest (and most biologically plausible) case, the activation function is a *threshold function* $\theta$:
  - if the weighted sum of the inputs is greater than a certain threshold $t$, then the neuron "turns on" (output 1), otherwise not.

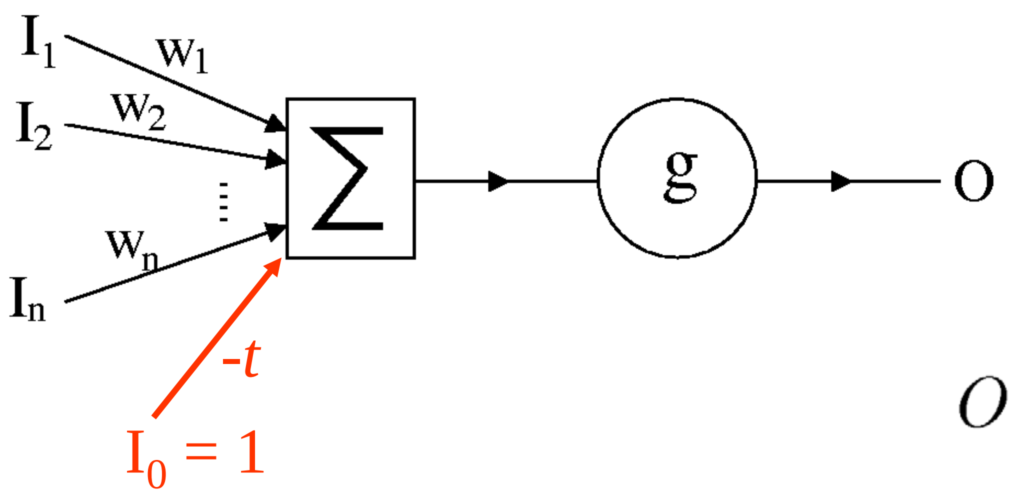
$$O = \theta\left(\sum_{i=1}^{n} w_i I_i - t\right)$$



where $\theta$ is the *Heaviside* function $\theta : R \longrightarrow \{0,1\}$

$$\theta(x) = \begin{cases} 0 & \text{se } x < 0 \\ 1 & \text{se } x \geq 0 \end{cases}$$

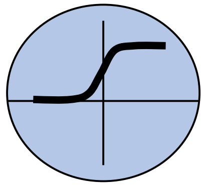
- This simple neuron is called *Perceptron*, introduced by Rosenblatt in 1962.
- Often the threshold is included in the neuron model, adding a fictitious input:
  - the value on this input is set at 1
  - the weight of the connection is given by *-t*;



$$O = g\left(\sum_{i=0}^{n} w_i I_i\right)$$

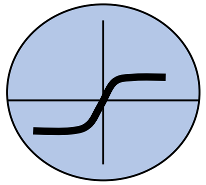
## Other activation functions

- Logistics function: $$g(x) = \frac{1}{1+e^{-x}}$$



- Hyperbolic tangent function:

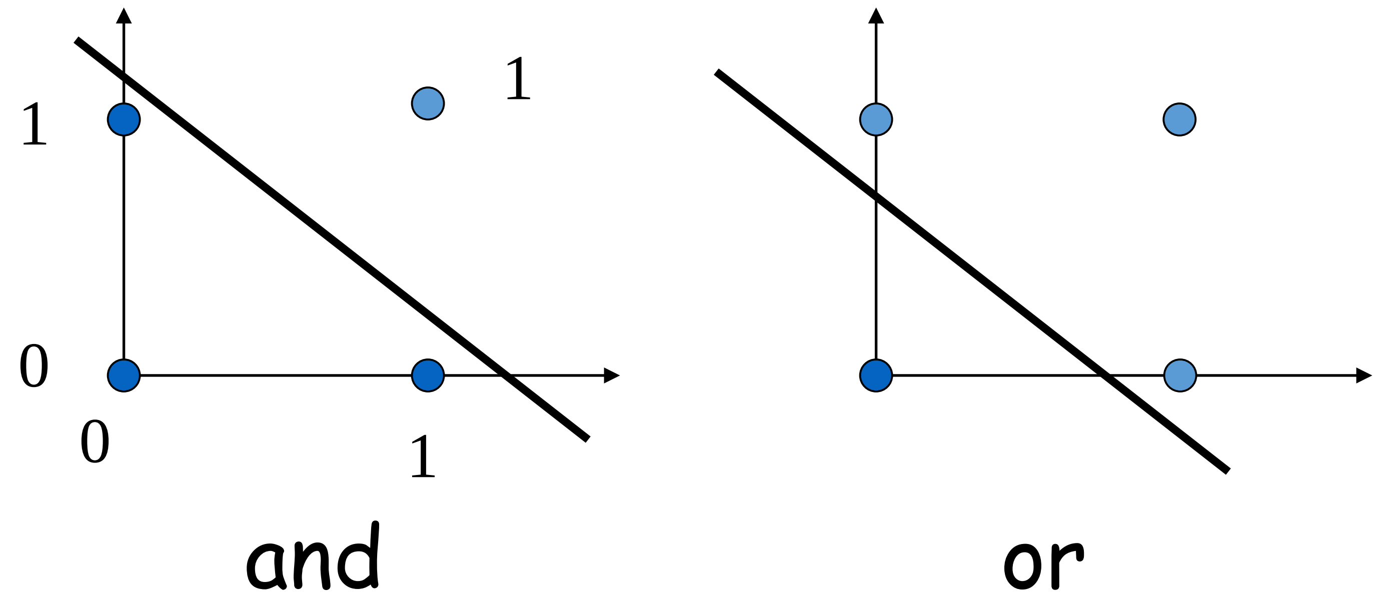
$$g(x) = tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



- These functions are interesting as they allow the neuron to have a continuous output, which allows an interpretation in a probabilistic key.

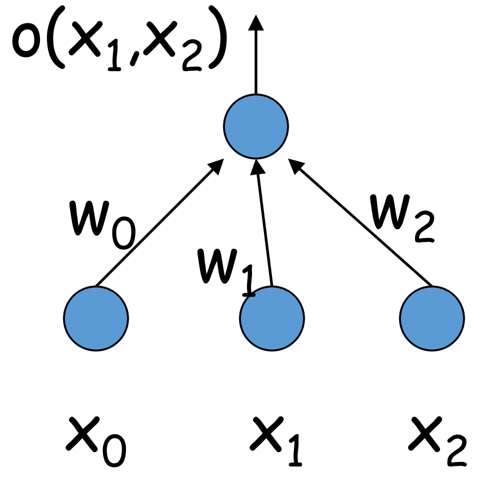
## Expressive abilities of the perceptron

- Each perceptron can represent only linearly separable functions.
- The separation surface between positive and negative examples is a plane.

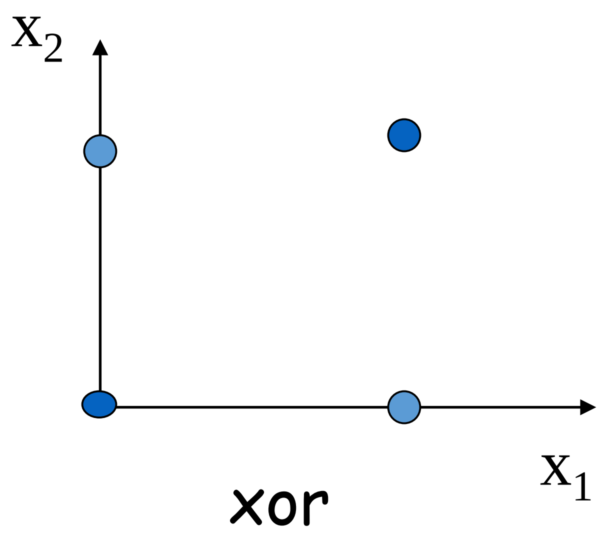


# Expressive limits of the perceptron

- The XOR function cannot be represented, being non-linearly separable



$$-w_0 + 0.w_1 + 0.w_2 \le 0$$
$$-w_0 + 0.w_1 + 1.w_2 > 0$$
$$-w_0 + 1.w_1 + 0.w_2 > 0$$
$$-w_0 + 1.w_1 + 1.w_2 \le 0$$



- We cannot find values of $w_{0,}$ $w_1$ and $w_2$ that meet the previous inequalities

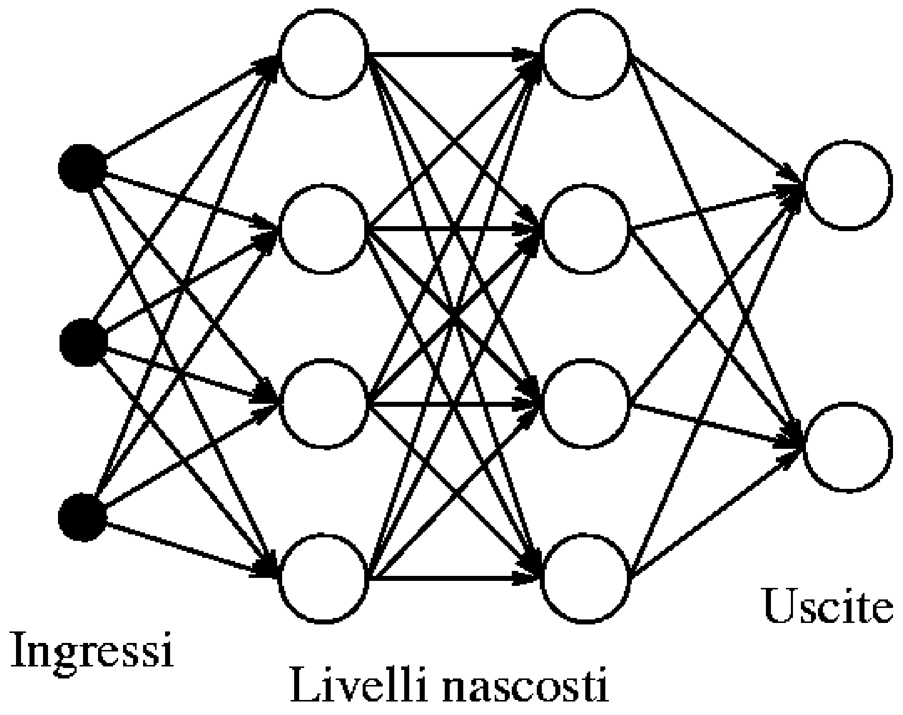
## The structure of the networks

- The aggregation of several neurons to form a network causes an explosion of complexity, as complex topologies can be obtained in which cycles are present.
- The simplest model, however, does not include cycles: the information passes from the input to the output (*feed-forward networks.*
- In case there are cycles, we will talk about *recurring networks*.

## *Feed forward* networks

- Simpler topology: no loops.

- The network is organized in layers:
  - each neuron of a layer receives inputs only from the neurons of the previous layer;
  - propagates the outputs only to the neurons of the following layers
- The layers between the inputs and the output are called hidden layers (in this case 2).



- In this type of networks, self-connections or connections with the neurons of the same layer are not possible..

- Each neuron therefore has the function of propagating the signal through the network, with a flow of information from the inputs to the outputs.
  - An immediate consequence of this is that the network always responds in the same way when solicited with the same inputs..

- In the event that the elementary units are the perceptrons, it will be called *Multilayer Perceptron* or *MLP*.

## Classification capacity

- Classes are representable as regions in the input space.

- In the simple case of two classes, we want to create a tool that outputs 0 if the input belongs to the first class and 1 if it belongs to the second class.

- The goal then becomes to approximate a region in space (e.g. the region of inputs that belong to class 1, whereas class 2 is the complementary):
  - inputs belonging to the class 1 region will "turn on" the output of the neural network

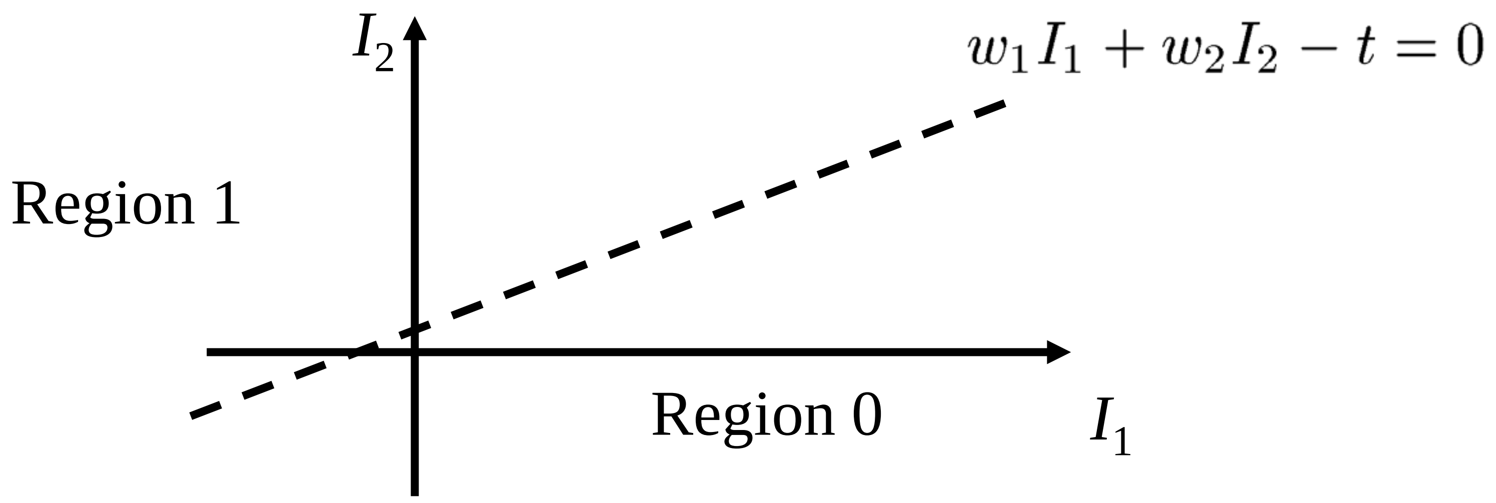
- Goal: To build a neural network that can recognize a region.

- Question: How does the network topology affect the complexity of recognizable regions?

- Consider a perceptron with two inputs $I_1$ and $I_2$, the output is given by:

$$O = \begin{cases} 1 & \text{se } w_1 I_1 + w_2 I_2 - t \geq 0 \\ 0 & \text{se } w_1 I_1 + w_2 I_2 - t < 0 \end{cases}$$

- The neuron then splits the input plane into two half-planes.:
  - on the one hand there are the inputs that produce as output of neuron 1, on the other there are the inputs that produce output 0

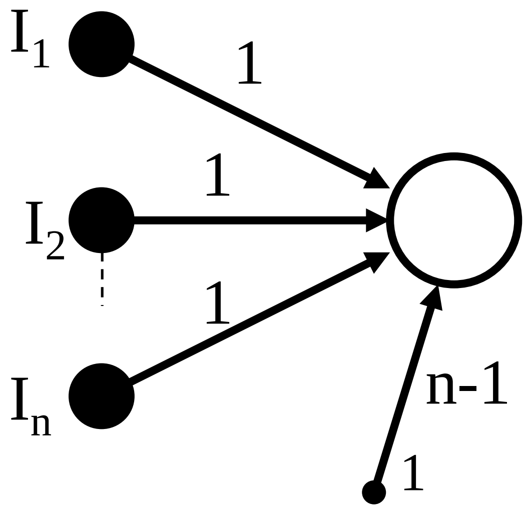


the line that delimits them is identified by:

$$w_1 I_1 + w_2 I_2 - t = 0$$

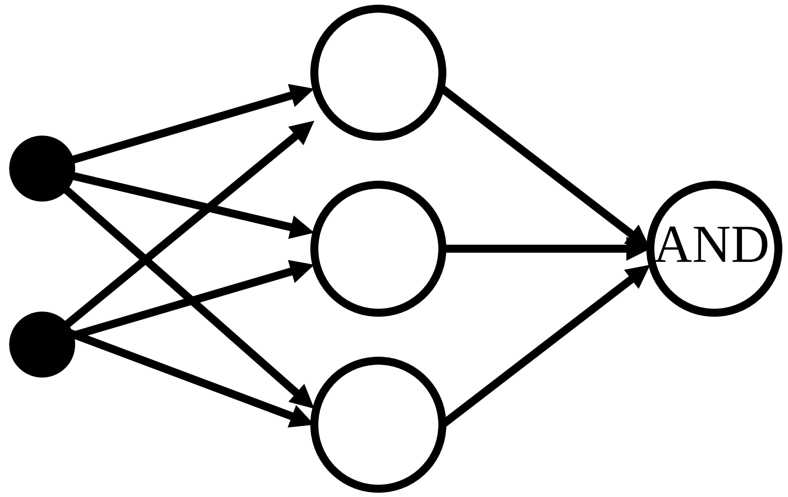
- For *n* inputs, a perceptron defines a hyperplane of dimensionality *n-1*, which divides the input space into two half-spaces: on the one hand there are the values for which the neuron turns on, on the other the remaining ones

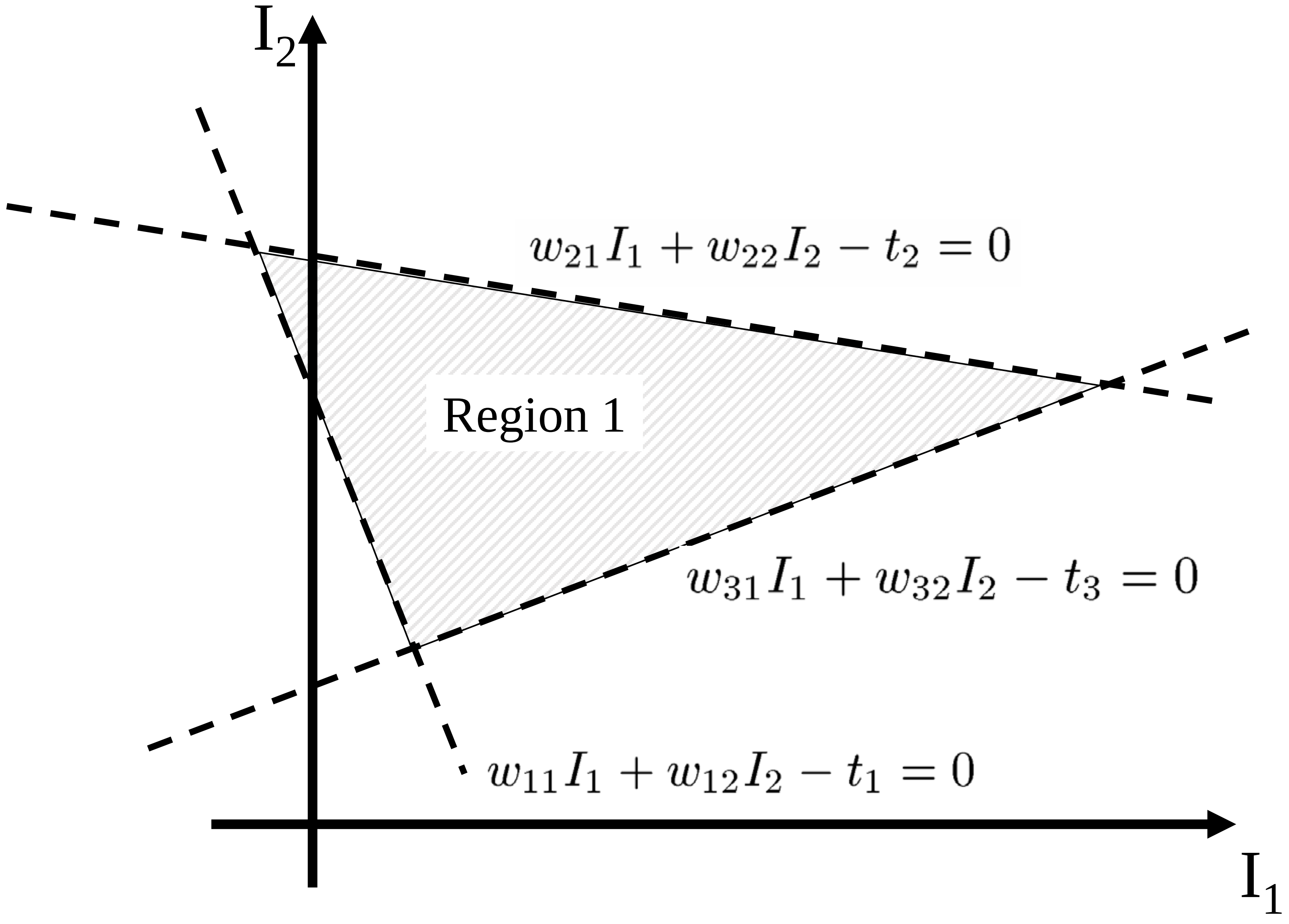
- Consider a neuron that acts as AND



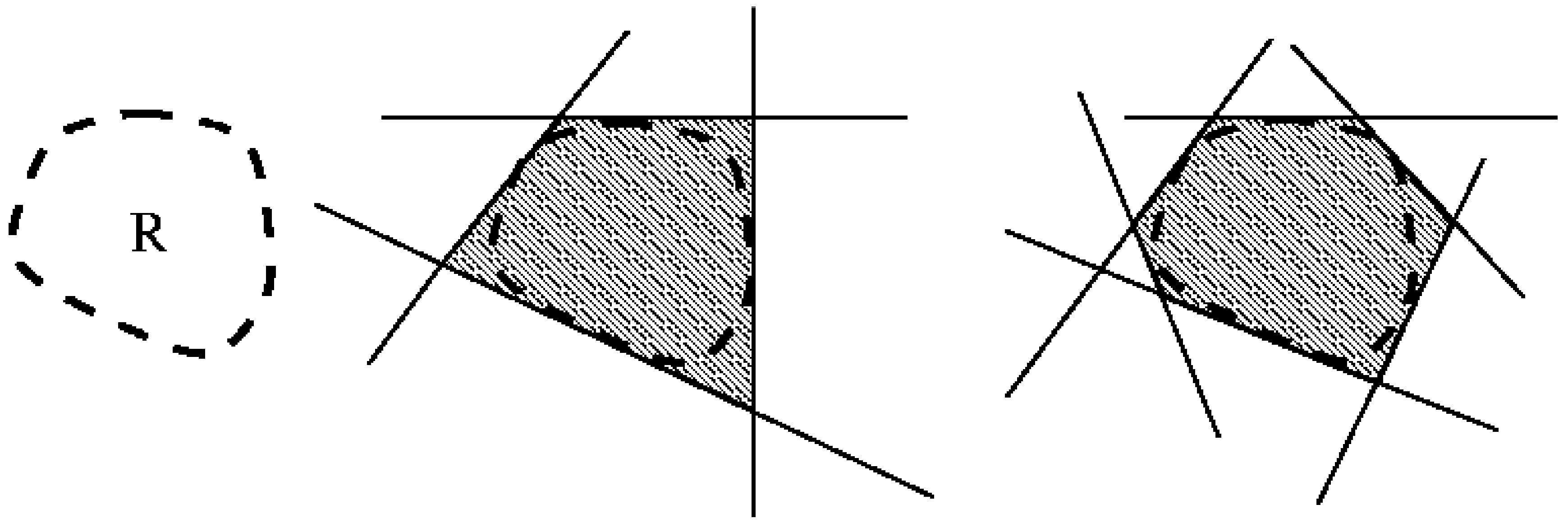
AND function of inputs
$I_1 \dots I_n$

- Consider an MLP - Multilayer Perceptron with 2 layers :
  - each neuron identifies a half-plane
  - the neuron of the second level performs an AND of the output of the neurons at the previous level
  - with three neurons we can identify a closed region of triangular shape




$I_2$
$w_{21}I_1 + w_{22}I_2 - t_2 = 0$
Region 1
$w_{31}I_1 + w_{32}I_2 - t_3 = 0$
$w_{11}I_1 + w_{12}I_2 - t_1 = 0$
$I_1$

- Using 4 neurons in the first layer, we can get a quadrilateral, and so on...
- Given a region R to be delimited in the input space, the more perceptrons, the more precise the approximation that the network will produce.



4 neurons

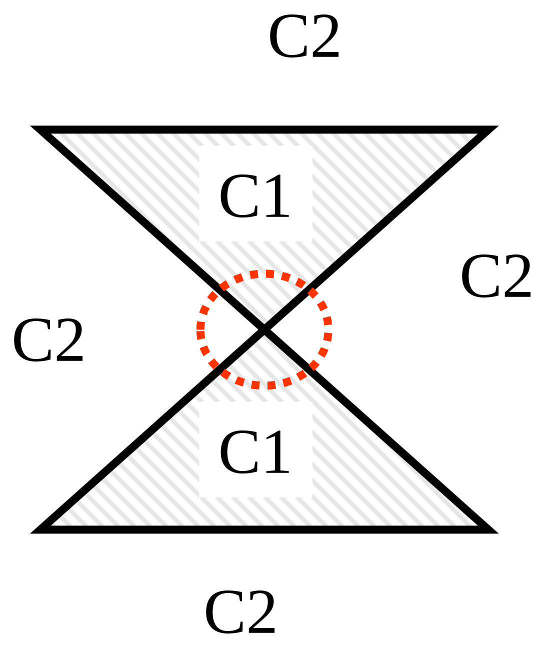
6 neurons

- *1 hidden layer*: we can obtain approximations of convex and connected regions

- Question: Are there any rules to figure out how the approximating capacity varies along with the variation of the structure?
- Yes! There are three theorems

- **Theorem 1** (Huang & Lipmann 1988): *feed forward* neural networks with a hidden layer can approximate "almost all" the separation surfaces between two classes.
  - “Almost all”: by increasing the number of neurons in the last layer (from 1 to 2 or 3), it is possible to classify even unconnected or convex regions.

- **Theorem 2** (Gibson & Lowan 1990): There are regions that neural networks with a hidden layer fail to approximate:



The problem lies in the point of contact, circled in red

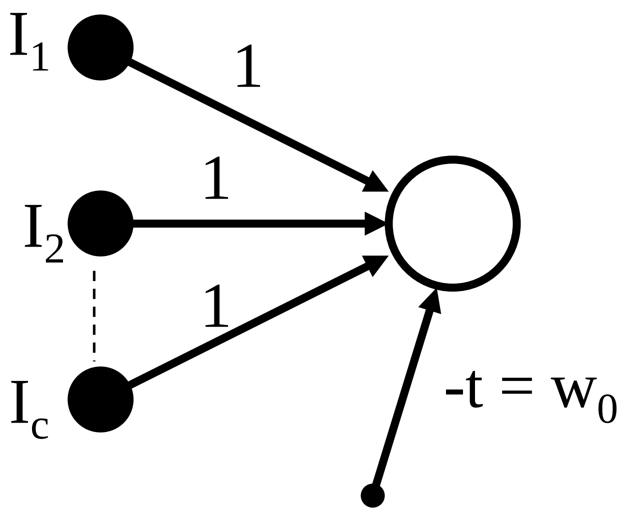
- **Theorem 3** (Lipmann 1987): Neural networks with two hidden layers can approximate any region of the space (in a 2-class problem).
  - The accuracy of the approximation then depends on the number of neurons used in the intermediate layers

- In practice? There are no fixed rules, often one resorts to rules of thumb or heuristics.

## Example: Template recognizer

- Objective: to build a binary pattern recognizer **b** = $b_1 \ldots b_c$.

- We use a very simple network, with c input and only one neuron: the output will be 1 if the pattern presented corresponds to the reference pattern **b**, and 0 otherwise.



- How do you build this neural network?
- The weight of each $w_i$ connection is fixed to

$$w_i = \begin{cases} 1 & \text{se} \quad b_i = 1 \\ -1 & \text{se} \quad b_i = 0 \end{cases}$$

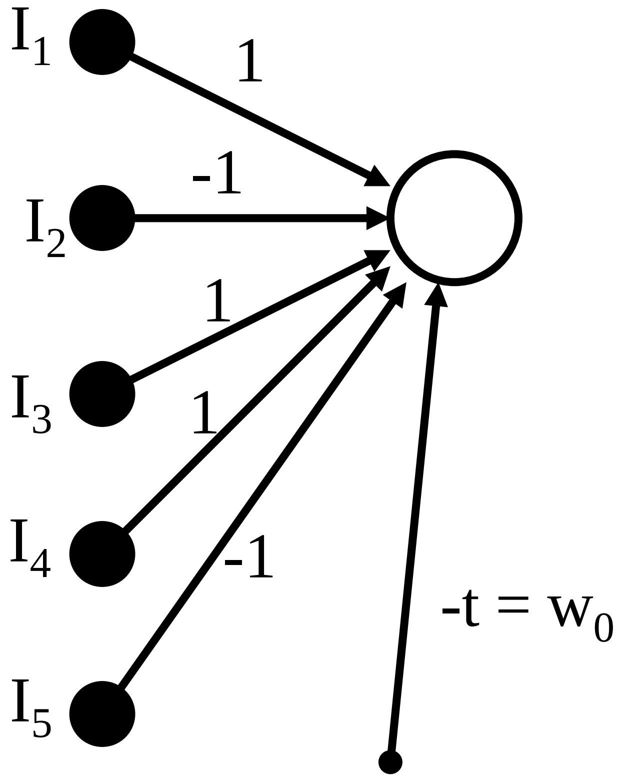
- We define *m* as

$$m = \sum_{i=1}^{c} b_i$$

- We obtain

$$\begin{cases} \sum_{i=1}^{c} I_i w_i = m & \text{se } I_i = b_i \quad \forall i = 1 \ldots c \\ \sum_{i=1}^{c} I_i w_i \leq m - 1 & \text{altrimenti} \end{cases}$$

- Example: **b**=10110, *m*=3; the network becomes:



Let's try to input the network
**I** = 10110 = **b**

$$\sum_{i=1}^{c} w_i I_i =$$

$$= 1 \cdot 1 + 0 \cdot -1 + 1 \cdot 1 + 1 \cdot 1 + 0 \cdot -1 =$$

$$= 3 = m$$

Let's try now to give a wrong input
**I** = 11110

$$\sum_{i=1}^{c} w_i I_i =$$

$$= 1 \cdot 1 + 1 \cdot -1 + 1 \cdot 1 + 1 \cdot 1 + 0 \cdot -1 =$$

$$= 2 \leq m - 1$$

- Choosing as threshold *m*-1, we will have that:

$$\begin{cases} \sum_{i=0}^{c} I_i w_i = 1 & \text{se } I_i = b_i \quad \forall i = 1...c \\ \sum_{i=0}^{c} I_i w_i \leq 0 & \text{altrimenti} \end{cases}$$

- Using the *Heaviside* function as an activation function, we get exactly the classifier sought

$$O = \begin{cases} 1 & \text{se } I_i = b_i \quad \forall i = 1...c \\ 0 & \text{altrimenti} \end{cases}$$

- Consideration: we can approximate with a neural network any Boolean function:

| | | |
|---|---|---|
| 000 | 1 | $\leftarrow$ |
| 001 | 0 | |
| 010 | 1 | $\leftarrow$ |
| 011 | 0 | |
| 100 | 0 | |
| 101 | 1 | $\leftarrow$ |
| 110 | 0 | |
| 111 | 0 | |

Given a generic Boolean function, I can build neurons that recognize the patterns that give true (those with the arrows), and thus putting them in AND

## Neural network training

- Once all the characteristics of the neural network have been established, such as
  - topology
  - number and type of neurons,
  - connections, etc.

  it is necessary to determine the weights of the connections in order to build a classifier, having available a <u>series of examples</u> taken from the problem in question: this operation is called neural network *training*.

- Let’s consider the case of *supervised training*, where, for each pattern in the set of examples, the desired output value is also specified.
- Supervised training therefore consists of:
    - present examples from the problem under consideration to the network
    - adjust the weights of the connections based on the discrepancies between the output produced and the desired output (ground truth).

The set of examples together with the desired output is called the *training set*, and is a table of the type:

| In | Out |
|---|---|
| $x_1$ | $y_1$ |
| $x_2$ | $y_2$ |
| $\vdots$ | $\vdots$ |
| $x_p$ | $y_p$ |

## The *training set*

- Training set $T = \{(x_1,y_1)\ldots(x_p,y_p)\}$:
  - o $x_i$ input vectors
  - o $y_i$ class to which the input vector belongs
- Represents instances of the problem: describes some of the values that variables take on in classes.
- It comes from measurements, so it can contain incomplete, inaccurate, noisy or approximate information.
- It represents a **critical factor**: it must be representative of the phenomenon in question:
  - o it must contain representative examples of every class;
  - o all classes must be well represented.

## Formalization

- Given the training set $T = \{(x_1,y_1)\ldots(x_p,y_p)\}$, the goal is to adjust the weights of the network based on these examples
  - the network must work well at least on these examples

- Define $f_W(x_i)$ the output of the network for input $x_i$, which depends on the weights $W$

- Once the topology is fixed, $W$ identifies univocally a neural network.

- We will look for the configuration of weights $W$ that minimizes a cost function $E(W)$.
- Different types of cost function can be defined: the simplest one is the *mean quadratic error*, defined as

$$E(W) = \frac{1}{2} \sum_{i=1}^{p} (y_i - f_W(x_i))^2$$

The goal then becomes to find the configuration $W^*$ that minimizes the error $E(W)$

$$W^* = \arg\min E(W)$$

## Minimization of error

- Since this function is differentiable, we can calculate its minimum using the *gradient descent method*:
  - it starts from an arbitrary configuration, often chosen randomly, denoted by $W^{(0)}$.
  - The weights are then updated iteratively according to the formula

$$W^{(k+1)} = W^{(k)} + \Delta W^{(k)}$$

- where

$$\Delta W^{(k)} = -\eta \left. \frac{\partial E}{\partial W} \right|_{W=W^{(k)}}$$

  - where $\eta$ represents a small positive real number called <u>learning rate</u>.

- At each iteration the idea is therefore to move, within the space of the weights, along the direction of maximum decrease of the error, that is, in the direction opposite to that of the gradient.

- The adjustment that is made to the weights at each iteration is determined by the learning parameter $\eta$:
  - the larger it is, the greater the correction that will be made at each iteration.

## Weight update: two approaches

- *Batch approach:* weight changes are made only after all the patterns of the training set have been presented to the network:
  - in other words, the error of the network is evaluated over the whole (or a part, *batch* in fact) set of examples before updating the weights;
  - the idea is to make a few but substantial changes.

- *Online* approach: changes take place after the presentation of each individual pattern:
  - we then proceed with many small changes.

- The second approach, by setting the learning parameter η small enough and randomly choosing the examples to be presented to the network, allows for extensive exploration of the space of the cost function.
- The first approach is now more used not on the entire training set but on selected subsets (*batches*).

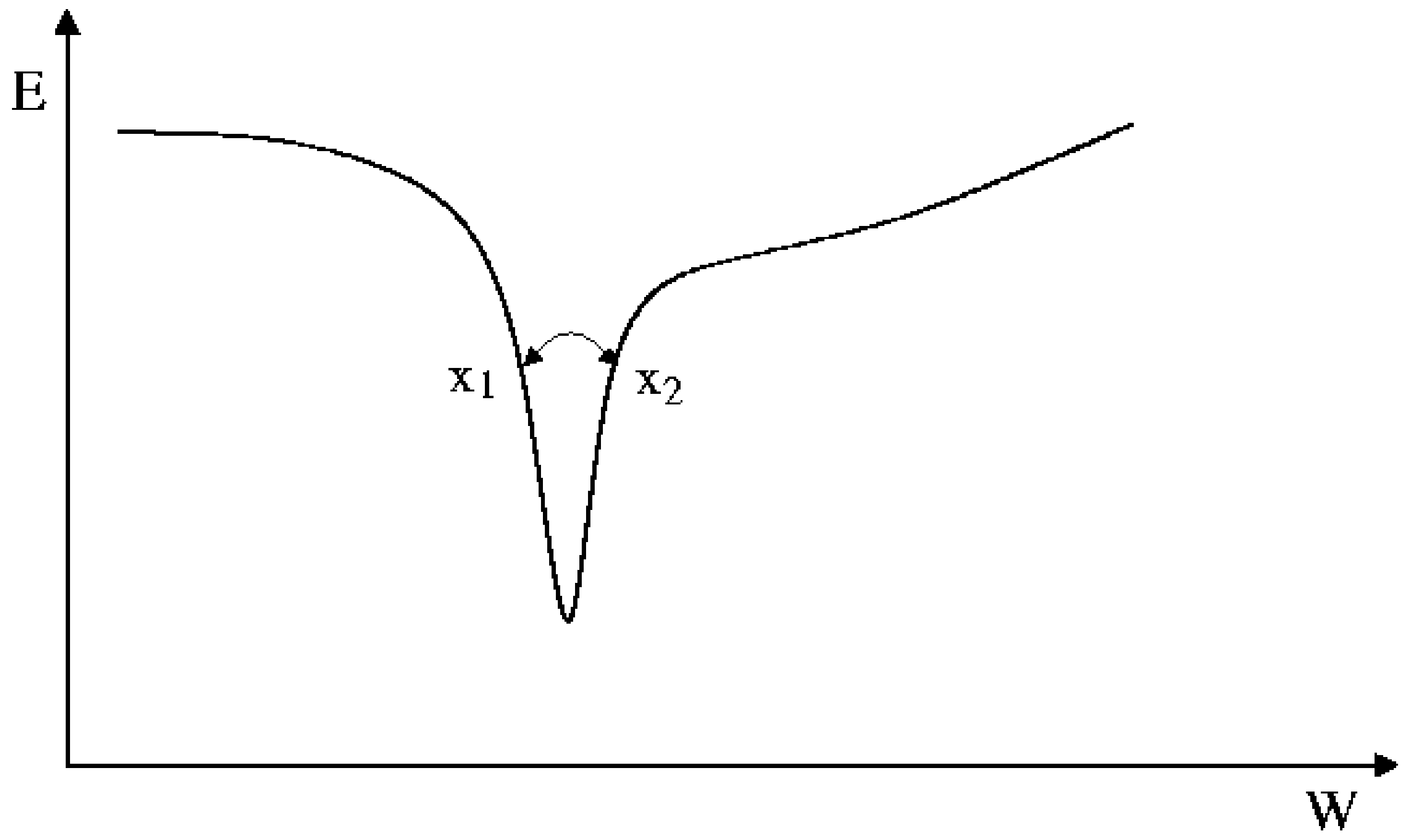
- *Epoch*: the presentation to the network of all the patterns of the set of examples:
  - it represents the unit of measurement of the training time

## Drawbacks of the gradient descent method

**Drawback 1**

The learning parameter $\eta$ is a critical factor:

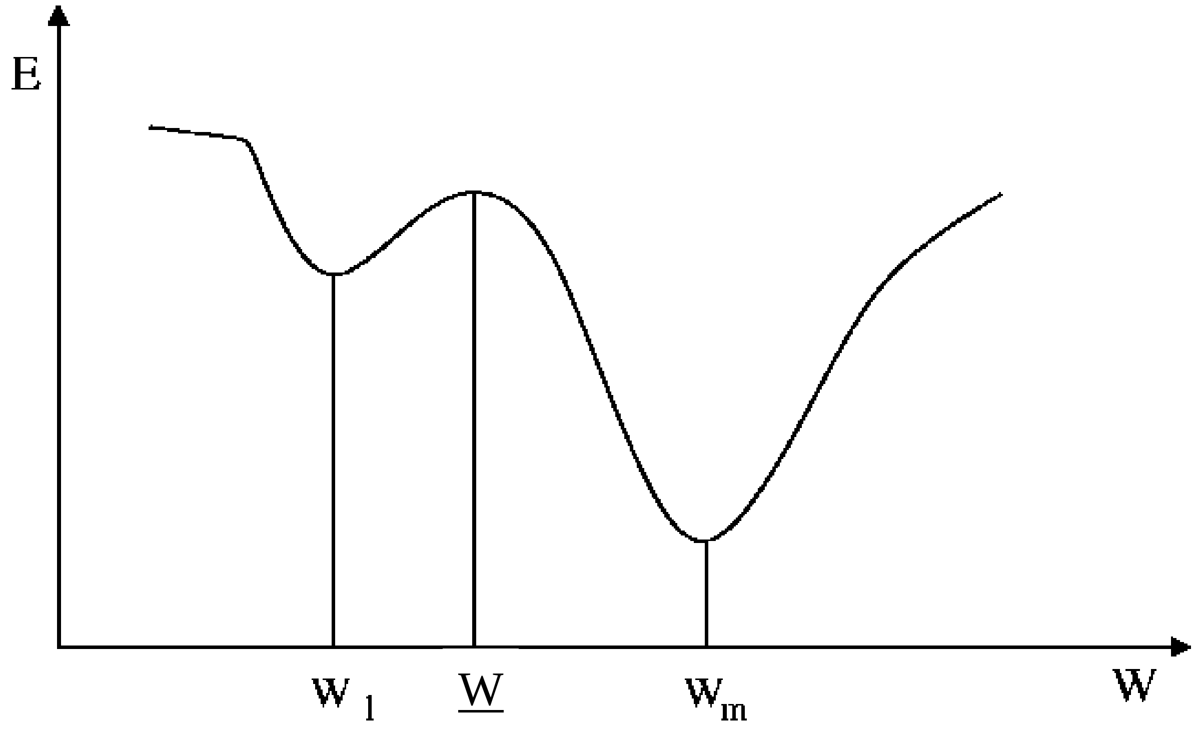
- it represents the extent of the correction made on the weights at each iteration of the algorithm, and thus determines the speed of convergence;
- if $\eta$ is too small, convergence can be too slow;
- on the other hand, $\eta$ too large can prevent an accurate determination of the minimum configuration and we may experience oscillations.



For $\eta$ too large, the algorithm may fail to descend into the valley, oscillating between the two points $x_1$ and $x_2$.

## Drawback 2

This technique converges to the nearest local minimum: the choice of initial weight values can be critical.



- If we choose as initial configuration any point bigger than $\underline{W}$, the algorithm will succeed to reach the minimum $W_m$ of the error function, otherwise it stops in the local minimum $W_l$.

- One solution could be to introduce in the formula of weights' updating some terms called *moments:*
  - the idea is to add to the variation on the weights to be made at each iteration, a contribution that derives from the previous step, imposing a kind of *inertia* on the system.

$$\Delta W^{(k)} = -\eta \left.\frac{\partial E}{\partial W}\right|_{W=W^{(k)}} + \alpha\left(W^{(k)} - W^{(k-1)}\right)$$

with $0 < \alpha < 1$ called *moment parameter* (usually $\alpha = 0.9$).

- Alternatively, global minimization techniques can be used, such as *simulated annealing*, *genetic algorithms*, or *Reactive Tabu Search*.

## Drawback 3

If, adopting the descent along the gradient, one arrives in an area where the activation functions saturate (i.e., $E(g)$ is almost constant), then:

- the derivative of $g$ is almost zero,
- the correction is very small at each iteration,
- we will have a drastic reduction of the learning speed.

- This can be partially avoided if we choose an initial configuration with very small weights: at least initially the activation functions do not saturate.

## The *Backpropagation* algorithm

- It is an optimized technique for network training
- Motivation: the calculation of the derivative of $E$ with respect to $W$ carried out by the difference quotient, is computationally expensive: its complexity is $O(W^2)$ with $W$ number of weights of the network.
    - In fact, it is necessary to calculate for each weight the difference quotient

$$\frac{E(W+h)-E(W)}{h}$$

    - whose calculation has complexity $O(W)$
    - it is necessary to evaluate the whole network to have $E(W)$, therefore it is necessary to evaluate $W$ times the difference quotient

## Backpropagation (2)

- It’s a method for training feed forward neural networks.
- It is based on the gradient descent method.
- Optimizes the calculation of the derivative, arriving at a total complexity of $O(W)$, with $W$ number of weights.

## General scheme

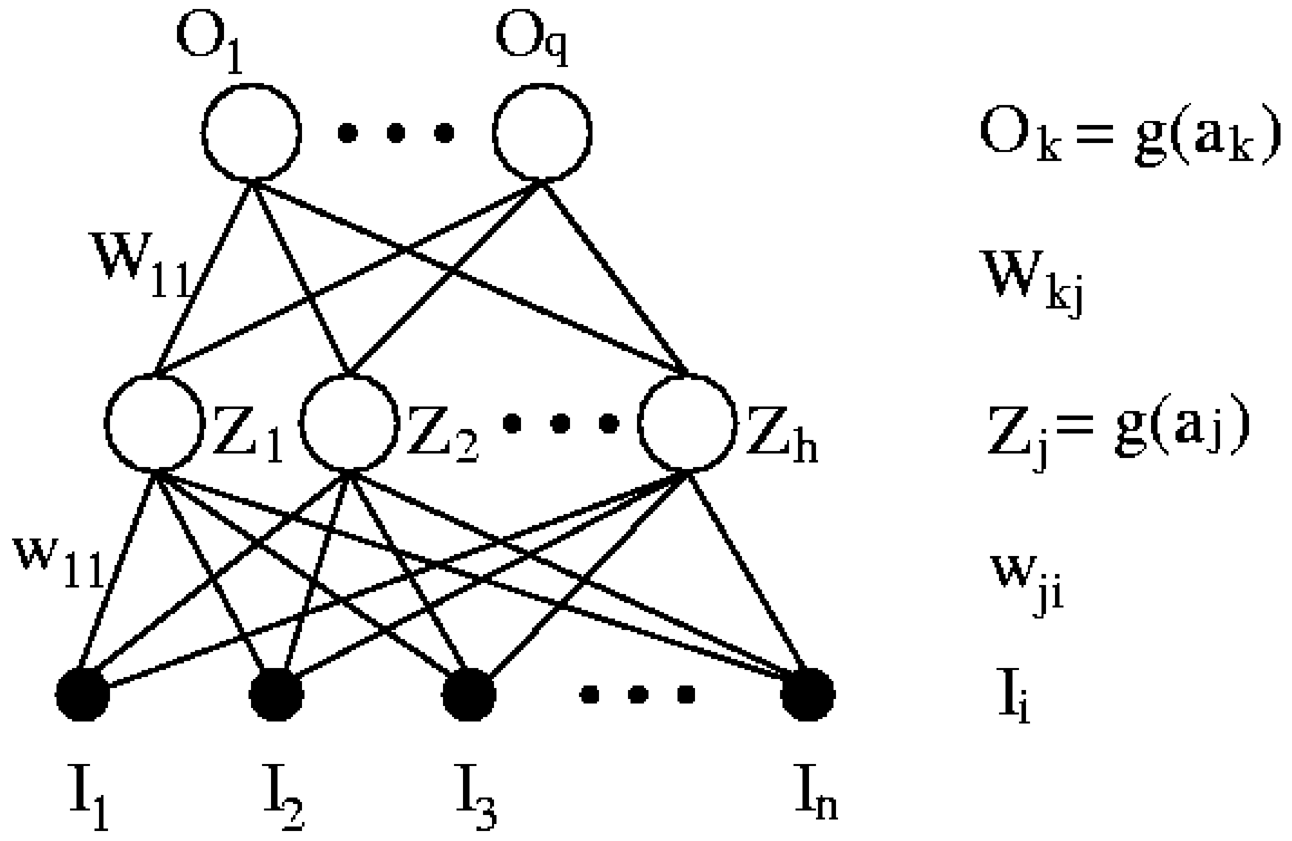
- *Back propagation* involves two phases, one *forward* phase and one *backward* phase:

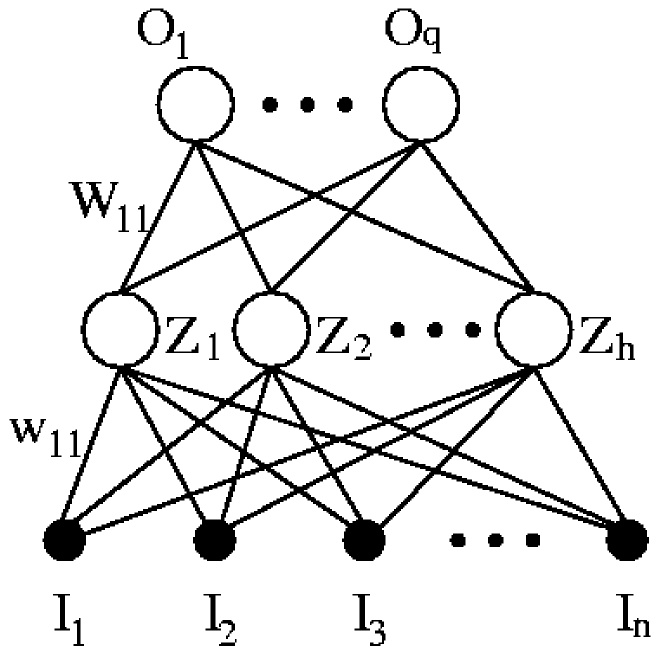
- *Forward* phase:
  - present an example to the network;
  - determine the output and calculate the error.

- *Backward* phase:
  - the error is propagated back in the network, progressively adjusting the weights.

## Formalization

- Consider a 2-layer *feed forward* network at $n$ input, $h$ neurons in the hidden layer and $q$ output
- $g$ activation function that can be derived (e.g. logistics)



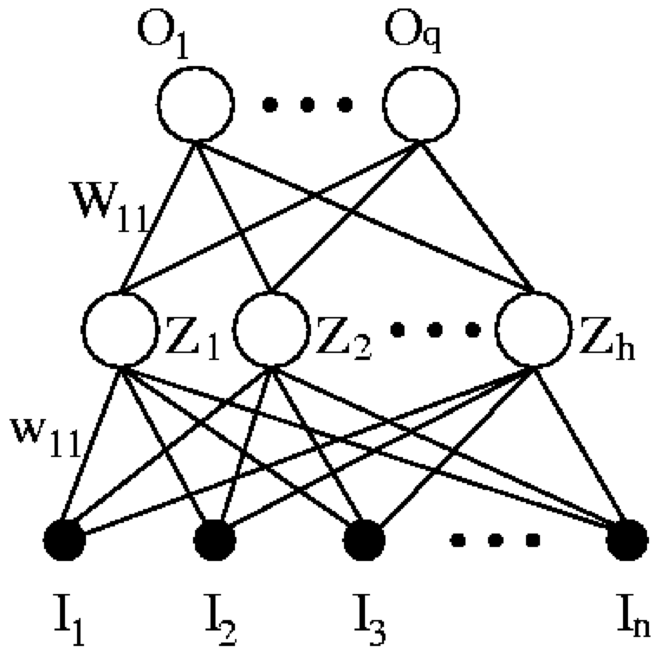


- $I_i$: network input.
- $w_{ji}$: weight of the connection from the *i*-th input to the *j*-th neuron of the hidden layer.
- $a_j$: weighted sum of the inputs of the *j*-th neuron of the hidden layer

$$a_j = \sum_{i=1}^{n} w_{ji} I_i$$

- $Z_j$: output of the *j*-th neuron of the hidden layer

$$Z_j = g(a_j)$$



- $W_{kj}$: connection weight from the $j$-th neuron of the hidden layer to the $k$-th neuron of the output layer.
- $a_k$ : weighted sum of inputs of the $k$-th neuron of the output layer

$$a_k = \sum_{j=1}^{h} W_{kj} Z_j$$

- $O_k$: output of the $k$-th neuron of the output layer

$$O_k = g(a_k)$$

- Considering an *online* approach for training, the error function (mean quadratic error) becomes:

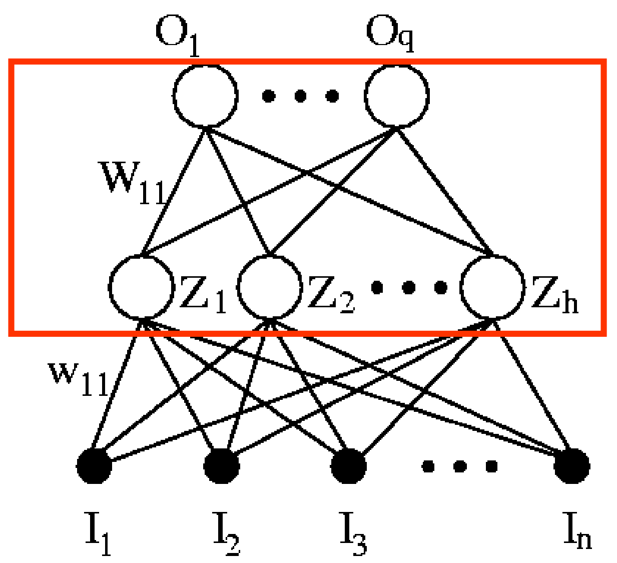
$$\begin{aligned} E(W) &= \frac{1}{2}\sum_i (y_i - O_i)^2 \\ &= \frac{1}{2}\sum_i \left[ y_i - g\left(\sum_j W_{kj}\, g\left(\sum_i w_{ji} x_i\right)\right)\right]^2 \end{aligned}$$

where $(x_i, y_i)$ represents a pattern of the training set, and $O_i$ are the outputs of the network with inputs $x_i$ (i.e. $\mathrm{I}_\mathrm{i}$ in the figure)

## Derivative calculation

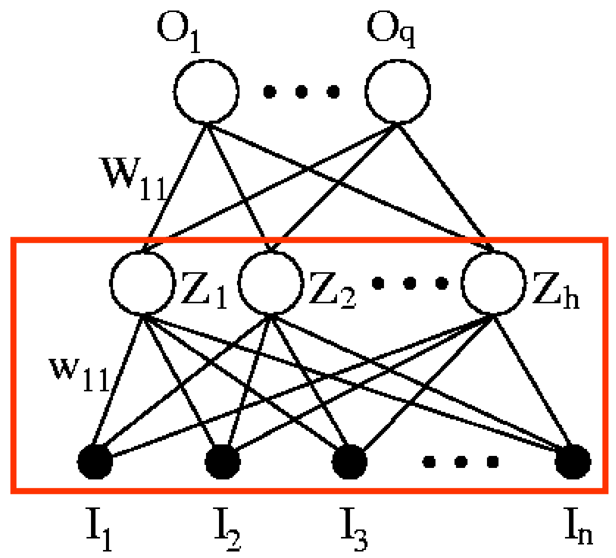
- We calculate the derivative of $E$ with respect to weights, in two separate steps:

Step 1



Weights of the connection between the hidden layer and the output

Step 2



Weights of the connection between the input and the hidden layer

## Connections Hidden Layer → Output

- For every single weight $w_{lm}$ we can apply the chain rule

$$\frac{\partial E}{\partial w_{lm}} = \frac{\partial E}{\partial a_l} \frac{\partial a_l}{\partial w_{lm}}$$

- For connections between the hidden layer and the output, we have that

$$\frac{\partial E}{\partial W_{kj}} = \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial W_{kj}}$$

with
$$\frac{\partial E}{\partial a_k} = \frac{\partial E}{\partial O_k}\frac{\partial O_k}{\partial a_k}$$
$$= (y_k - O_k)g'(a_k)$$

and
$$\frac{\partial a_k}{\partial W_{kj}} = Z_j$$

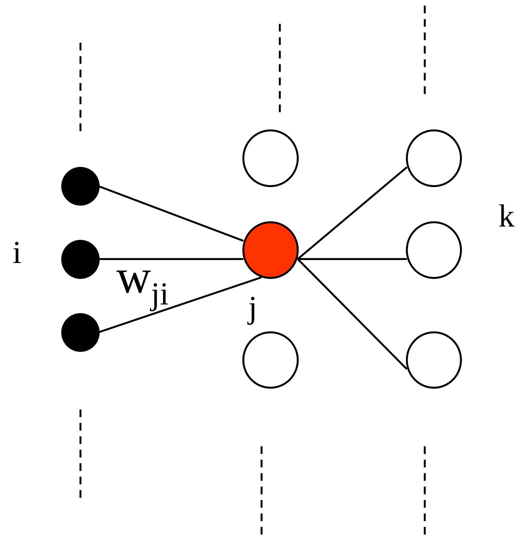
Defining:
$$\delta_k = \frac{\partial E}{\partial a_k} = (y_k - O_k)g'(a_k)$$

we get
$$\frac{\partial E}{\partial W_{kj}} = \delta_k Z_j$$

## Connections Input → hidden layer

Applying the chain rule:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}}$$



We get

$$\delta_j = \frac{\partial E}{\partial a_j} = \sum_k \frac{\partial E}{\partial a_k}\frac{\partial a_k}{\partial z_j}\frac{\partial z_j}{\partial a_j} = \sum_k \delta_k \frac{\partial a_k}{\partial z_j}\frac{\partial z_j}{\partial a_j} =$$

$$= \sum_k \delta_k w_{kj}\frac{\partial z_j}{\partial a_j} = \sum_k \delta_k w_{kj} g'(a_j) = g'(a_j)\sum_k \delta_k w_{kj}$$

- The derivative of $a_j$ with respect to $w_{ji}$ is

$$\frac{\partial a_j}{\partial w_{ji}} = I_i$$

- To recap, by defining

$$\delta_j = g'(a_j) \sum_k \delta_k w_{kj}$$

- We get

$$\frac{\partial E}{\partial w_{ji}} = \delta_j I_i$$

## Generalizing ...

- The derivative in the two steps is calculated according to a similar formula. And generalizing, we have that

$$\Delta W_{pq} = -\eta \delta_{output} \times V_{input}$$

where:

- *input* and *output* represent the origin neuron ($q$) and the target neuron ($p$) of the connection
- $V$ represents the input value of the neuron
- $\delta$ represents the "error" that is propagated backwards, varies from layer to layer but maintains a formula similar to

$$\delta_j = g'\left(a_j\right) \sum_k \delta_k w_{kj}$$

## Summarizing:

- *Forward* phase:
  - Input $x_i$ are fed to the network;
  - $Z$ and $O$ are calculated and stored (in general, they are the activations of the hidden and output units)
  - The complexity is $O(W)$

- *Backward* phase:
  - $\delta_k$ is calculated for output units
  - propagates $\delta_k$ backwards to calculate $\delta_j$ (for each hidden unit)
  - The complexity is $O(W)$

- Everything is repeated for each pattern.

## Considerations

- The complexity of the calculation of the derivative is $O(W)$: there is therefore a drastic reduction of the complexity compared to the use of the difference quotient

- The weight update rule is a *local* rule:
  - to calculate the variation of a weight, the algorithm needs only the information present at the two ends of the connection;
  - this allows a high parallelization of the neural network training process.

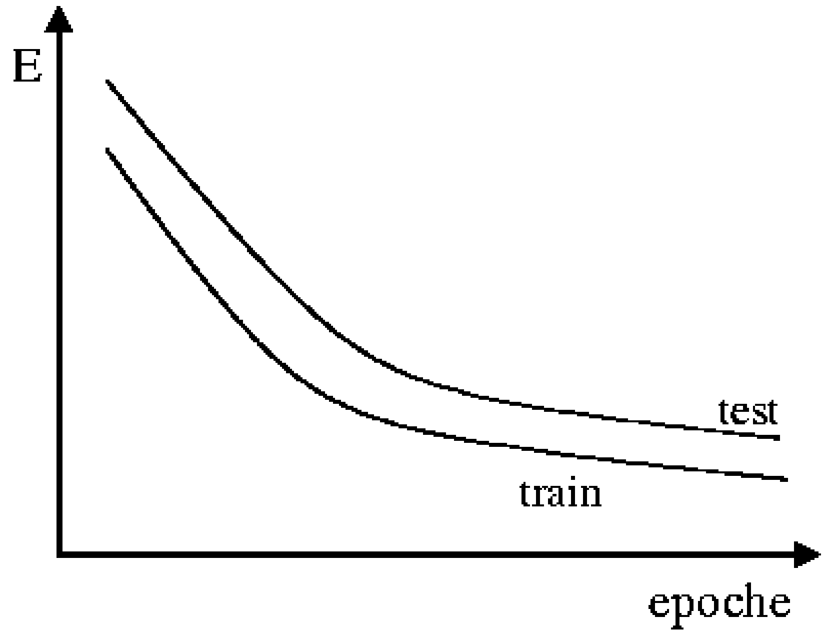
## Generalization

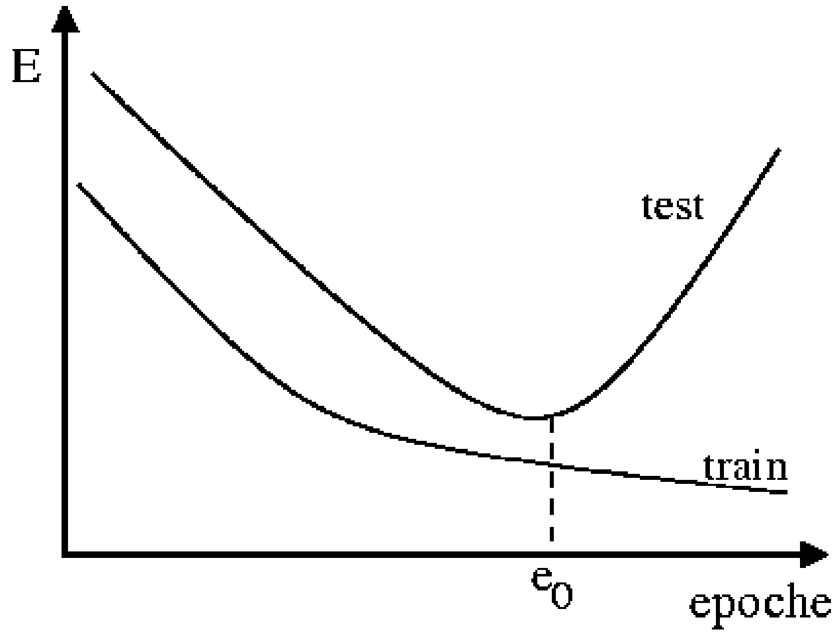
- *Generalization capacity of a network*

  Ability to correctly evaluate and recognize examples of the problem under consideration not present in the training set.

- This ability depends by the topology of the network (number of layers, neurons per layer, etc.) and the choice of the training set:
  - the larger and more representative of the phenomenon in question, the greater the capacity of generalization.

- Measurement of generalization capacity:
  - it would be necessary to extract other examples from the problem and use them to test the network: unfortunately this may results too expensive;
  - the alternative that is adopted in most cases is to divide the available set of examples into two parts and
    - use a part to train, and
    - use the other part to test the network.
  - By calculating the error on the training set and on the test set along time, it’s possible to decide when the *training* can stop





The network generalizes well and one can get a low error

After a number of $e_0$ epochs, the error on the test set starts to grow: an *over-training* situation occurs.

This means that the network is starting to lose the generalization ability and it is therefore appropriate to stop the training

- *Overtraining*

  situation in which the network has learned the example patterns so well that it has lost the flexibility to recognize other patterns, that is, the network has “memorized" the patterns of the training set, without having learned anything.

- Note: Network training is not equivalent to minimizing error on the training set (overtraining must be taken into account).
- In other words, it is not always said that the minimum of the error corresponds to the best training for the neural network.

## Training - Testing set subdivision

- Typically, the set of available data is limited:
  - it is not viable to build test sets continuously by repeating experimental trials;
  - it is necessary to properly exploit the given data set.

- There are several methods to split the training set from the testing set:
  - *Resubstitution* method;
  - *Cross-Validation* methods.

## The *Resubstitution* method

- The idea in this method is to train and test the classifier with the same set, the entire set of available data.

- This approach clearly provides an optimistic estimate of the error, in the sense that it is a lower bound.

## *Cross Validation* methods

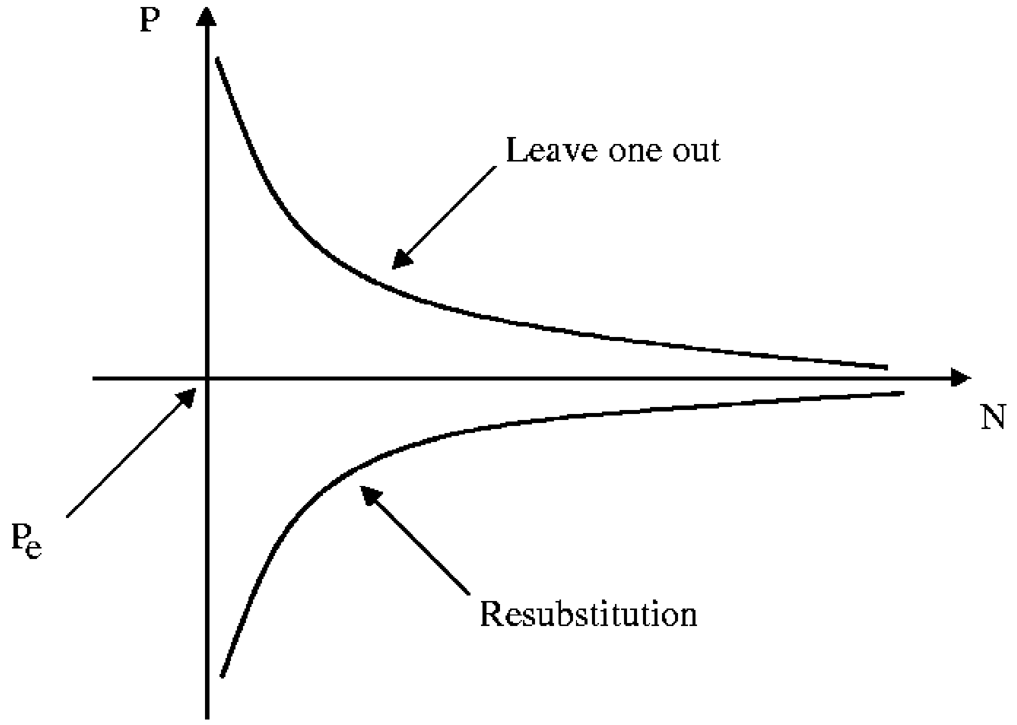
- The methods belonging to the *Cross-Validation* class (there are many variations) are adopting two disjoint sets to train and to test a classifier.

- Variants:
  - *Holdout*
  - *Averaged Holdout*
  - *Leave One-Out*
  - *Leave K-Out*

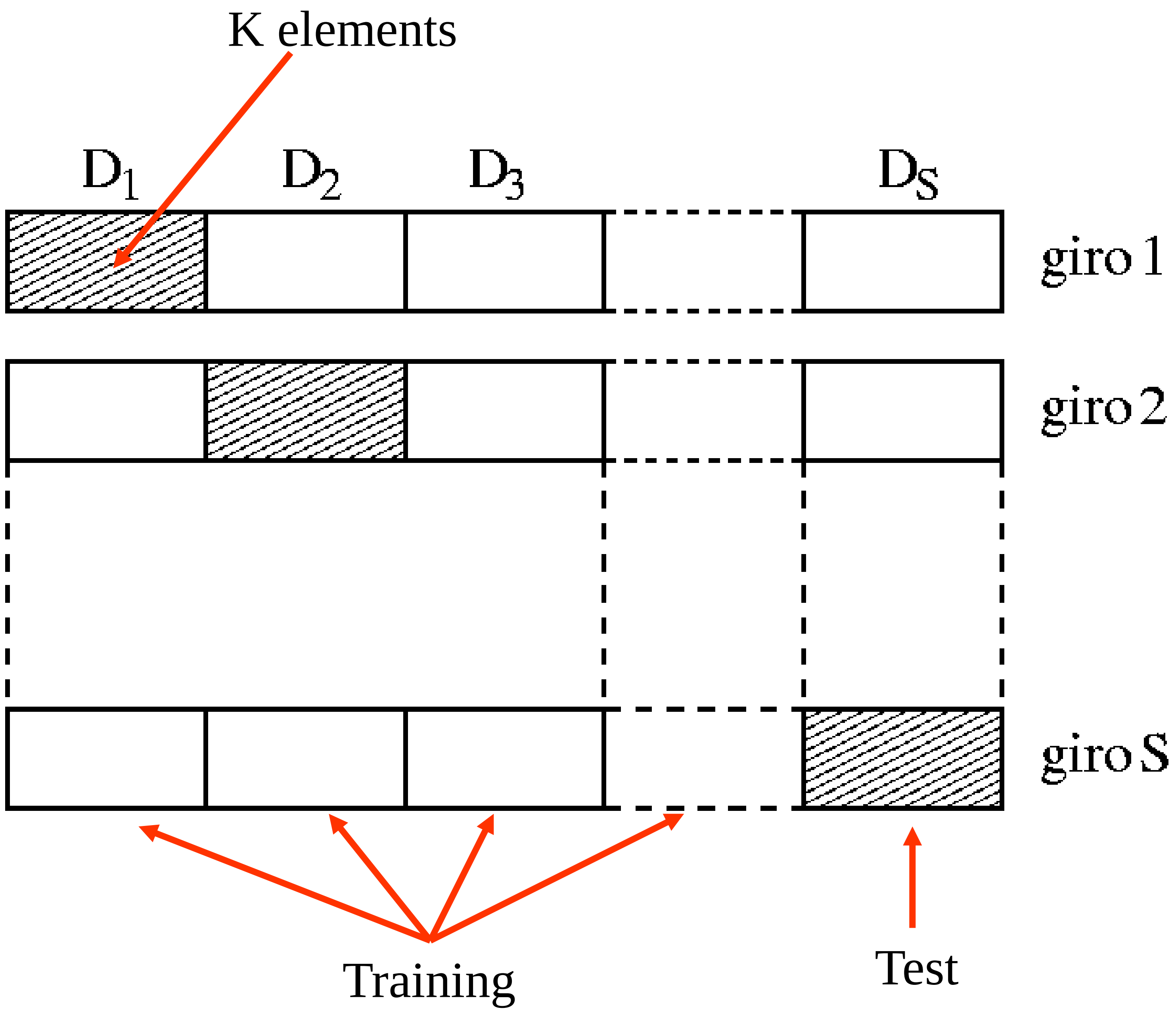
- *Holdout*
  - The whole data set is randomly partitioned into two disjoint subsets of equal size;
  - one of the two subsets is used as a *training set* and the other as a *test set*;
  - this method provides an upper bound of the error.

- *Averaged Holdout*
  - to make the result less dependent on the chosen partition, one can average the calculated results over multiple holdout partitions;
  - partitions are built randomly, or exhaustively;
  - this method provides an upper bound of the error.

- *Leave One-Out*
  - given a data set of cardinality N, this method uses N-1 samples to train the classifier, while the only excluded data sample is used to test it;
  - an average is made over all the trials;
  - this method provides an upper bound of the error.



- *Leave K-Out*:
  - this technique is a generalization of the previous technique;
  - the idea is to divide the data set into distinct and random segments;
  - the classifier is trained using S-1 segments, while it is tested using the remaining segment;
  - this operation is carried out S times, varying in turn the segment in the Test Set;
  - finally the error is averaged over the S results.


K elements
$D_1$
$D_2$
$D_3$
$D_S$
giro 1
giro 2
giro S
Training
Test

## When neural networks are appropriate

- If the instances of the problem are given in pairs (attributes - class)
  - Typically, a preprocessing step is required, e.g., the input real values should be scaled in the range [0-1], while input discrete values should be converted to Booleans.
- If the examples in the training set are numerous.
- If long training times are acceptable.
- If it is not important that the discriminant function is expressive for a human subject.

## Advantages and disadvantages

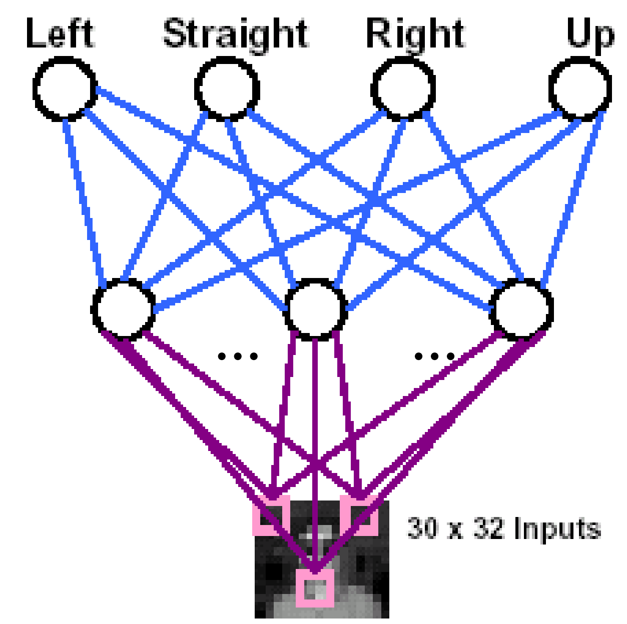
- Advantages:
  - It is an inherently parallel algorithm, ideal for multiprocessor hardware
  - represents a very powerful classification system, used in countless contexts

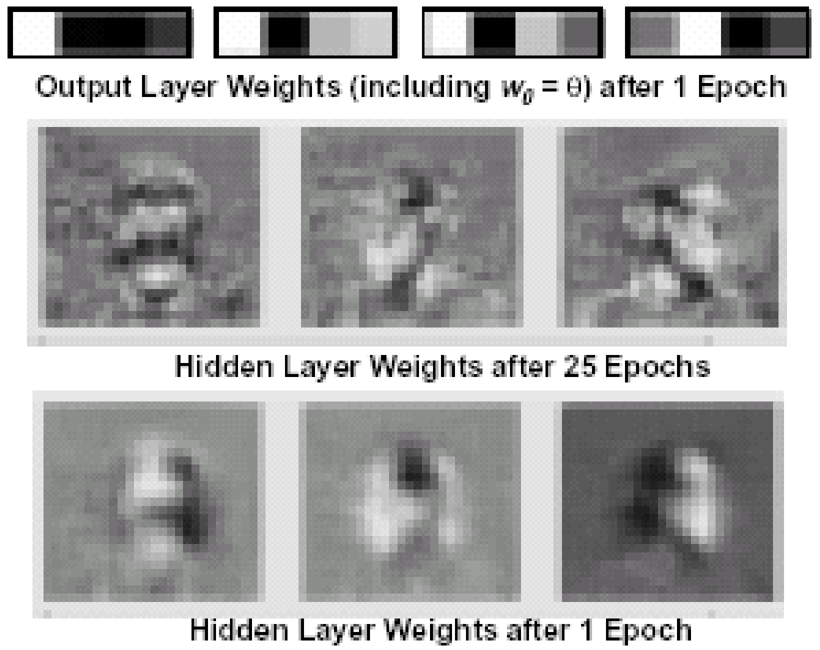
- Disadvantages:
  - determining network topology is an art
  - generalization vs. memorization: with too many neurons, the network tends to memorize patterns and can no longer generalize
  - networks require long and costly training phase

## Examples of applications in Vision

- Face and character recognition
- Edge detection
- Driving vehicles
- ... *many more*

## Example: Neural Nets for Face Recognition




Output Layer Weights (including $w_0 = \theta$) after 1 Epoch

Hidden Layer Weights after 25 Epochs

Hidden Layer Weights after 1 Epoch

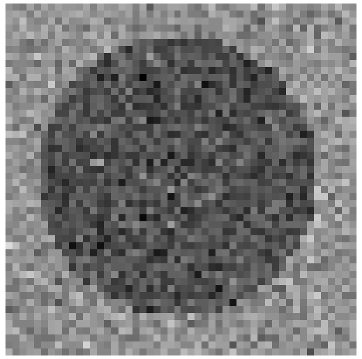
- 90% Accurate Learning Head Pose, Recognizing 1-of-20 Faces
- http://www.cs.cmu.edu/~tom/faces.html

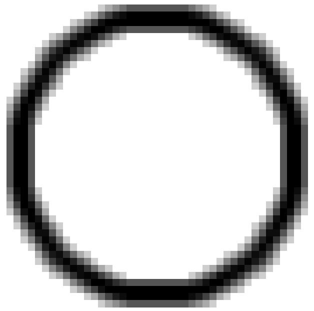
CIS 830: Advanced Topics in Artificial Intelligence

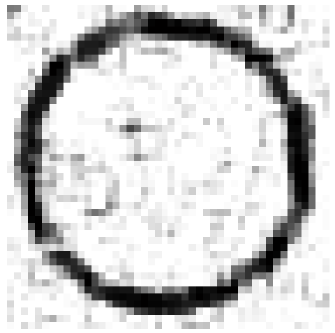
Kansas State University
Department of Computing and Information Sciences

## Edge detection: training

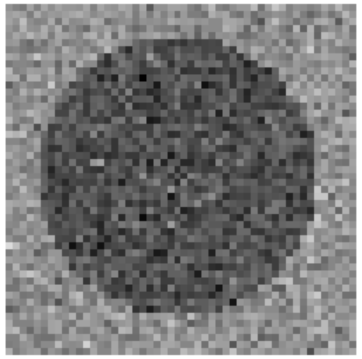
Training set 1

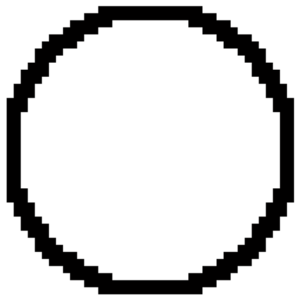

Input

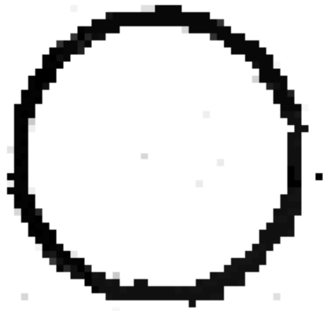

Desired output


Fitted output

Training set 2

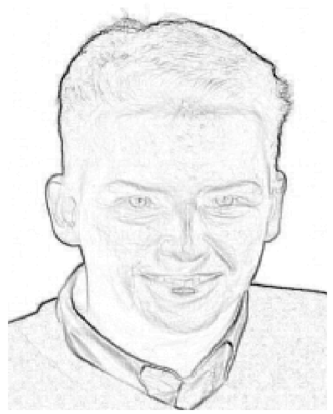

Input

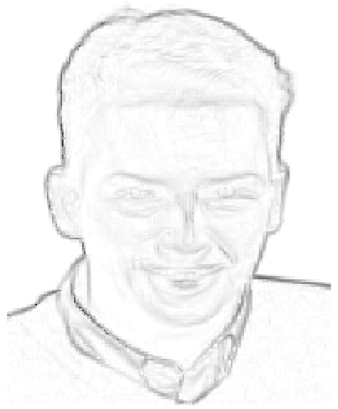

Desired output

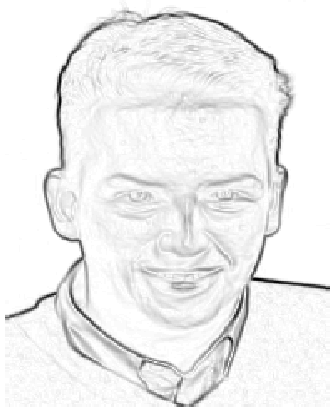

Fitted output

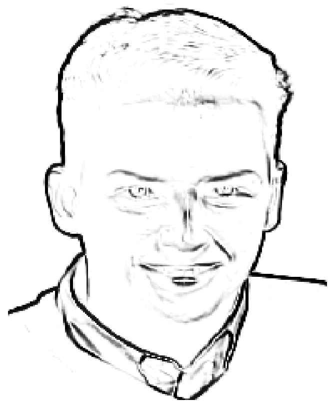
# Edge detection: results

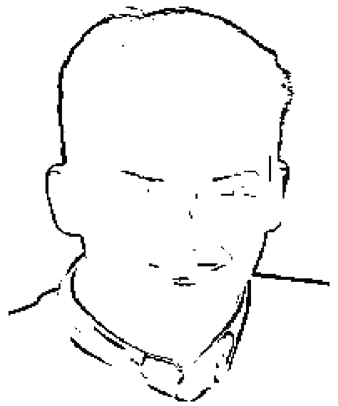
Original image


Dilation-erosion edges


Fitted single sigmoid to 3x3 fields edges

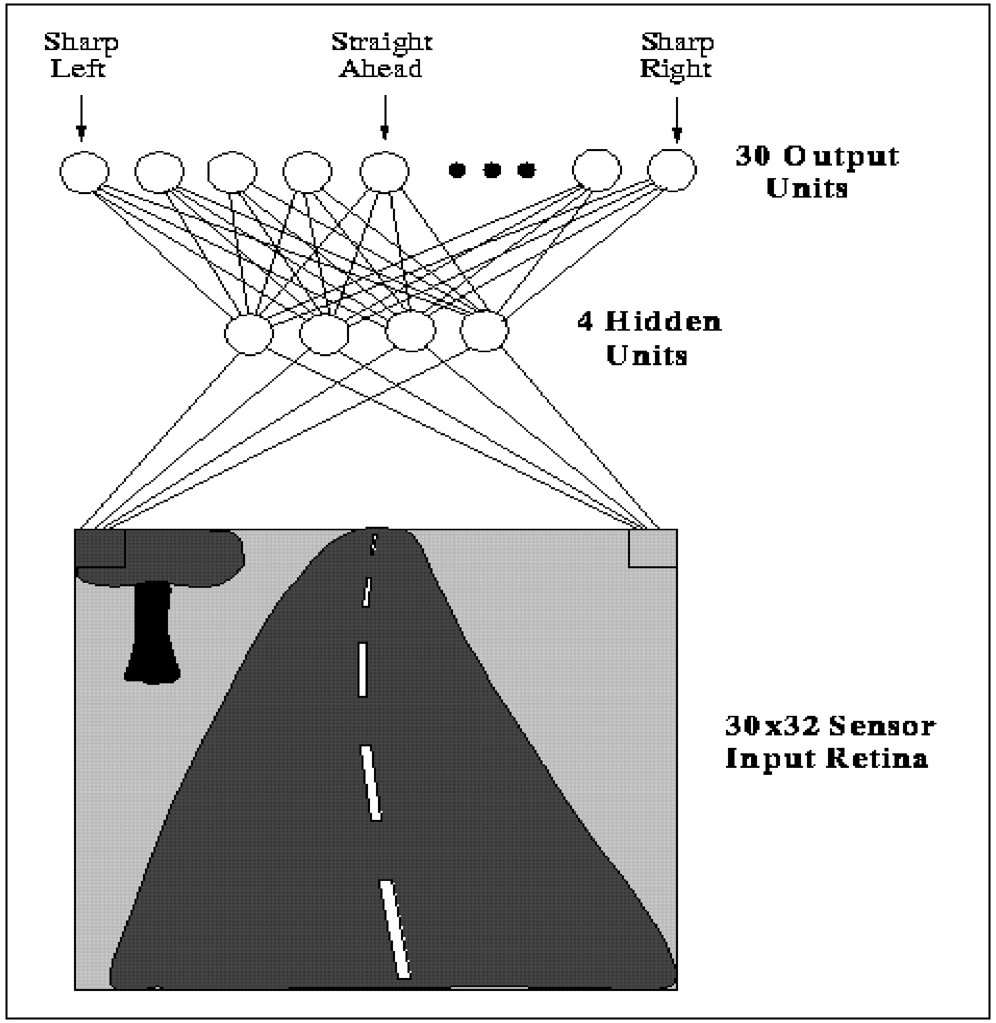

Fitted plane edges


FF ANN, training set 1


FF ANN, training set 2

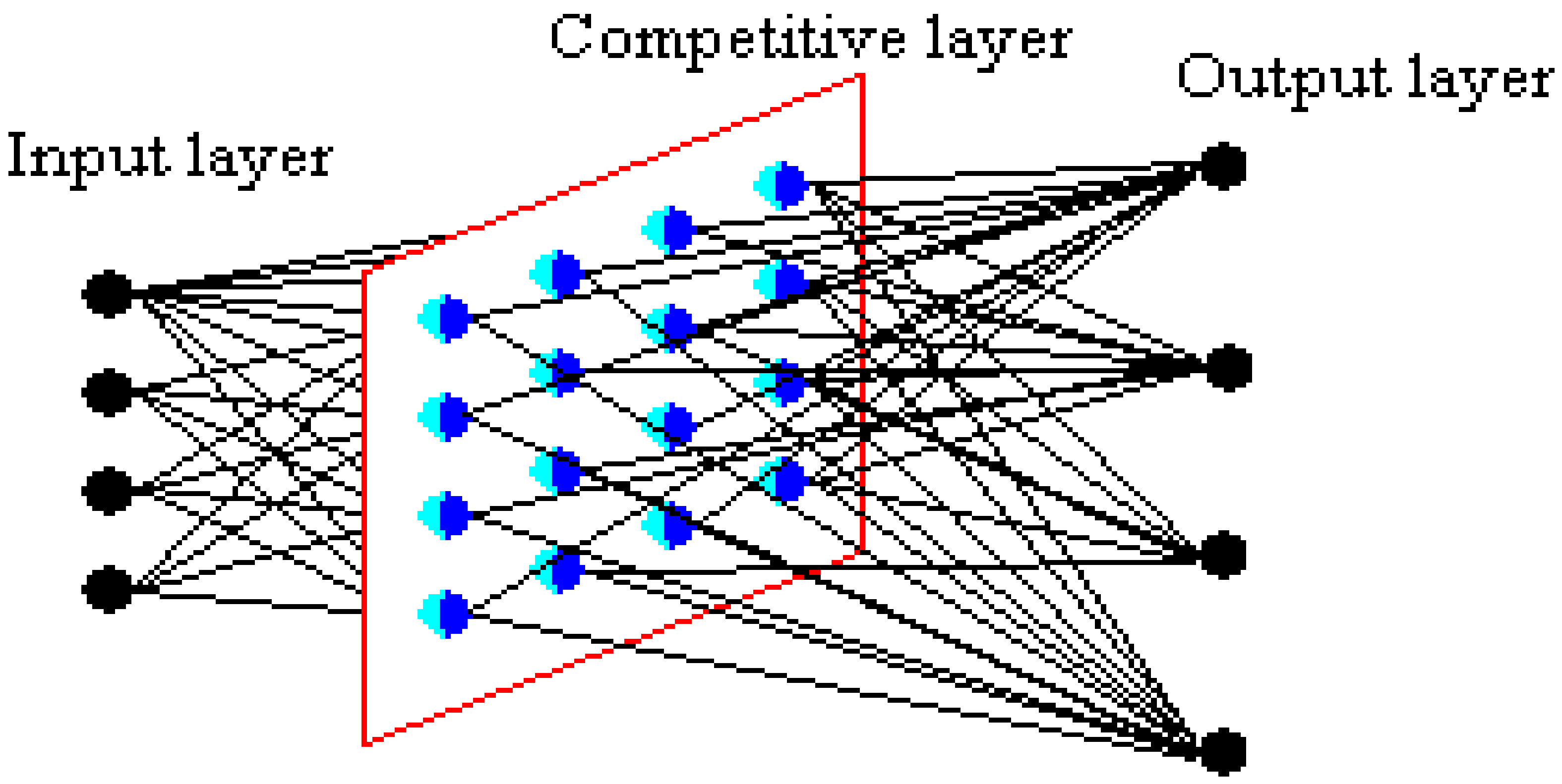
Truck driving

(Pomerlau)



## Kohonen networks

Kohonen networks implement a self-organizing feature map



## Self-organizing feature map

- The experience of the senses is multidimensional

- Example: sound is characterized by timbre, tone, intensity, noise, etc.

- The brain maps the external multidimensional representation of the world (with its spatial relationships) into an internal 1D or 2D representation (feature map). Examples:
    - *tonotopic map*: sound frequencies are mapped in regions of the cortex spatially ordered with respect to frequency;
    - *retinotopic map*: the visual field is mapped in the visual cortex (occipital lobe) with a resolution that grows from the center to the outside of the visual field, maintaining proximity relationships

## Mathematical model

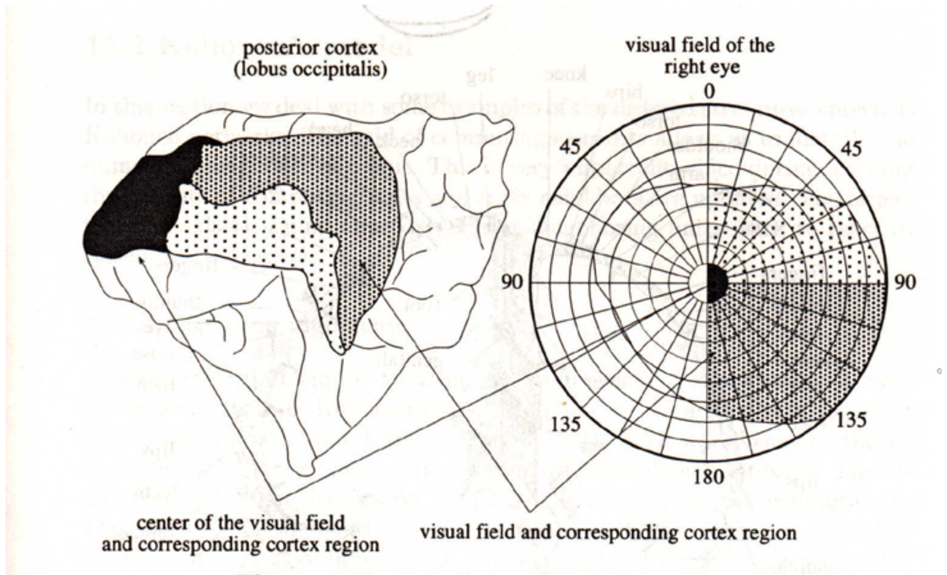
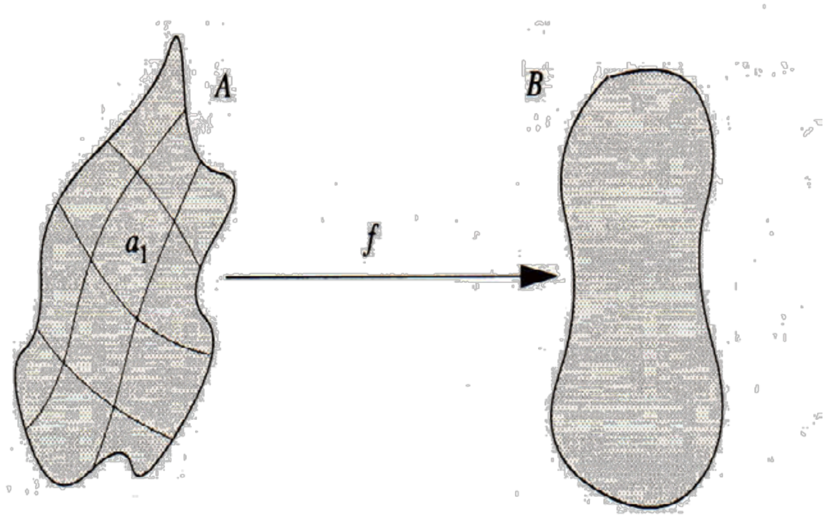

posterior cortex
(lobus occipitalis)
visual field of the
right eye
0
45
45
90
90
135
135
180
center of the visual field
and corresponding cortex region
visual field and corresponding cortex region

**Fig. 15.2.** Mapping of the visual field on the cortex



- The Kohonen network determines a function *f* from input space A to output space B.
- The domain of *f* is estimated by the Kohonen network so that, when a vector is chosen from a predetermined zone, only one neuron in the network fires.

## Competitive training

- The input layer receives an input multidimensional pattern, the vector $\mathbf{x} = (x_1, \ldots, x_n)$
- The *competitive layer* can be one-dimensional, two-dimensional, or n-dimensional (typically 2D).
- Each neuron in the competitive layer receives a weighted sum of the inputs of the input layer

$$a_i = g\left(\sum_j w_{ij} x_j\right) = g\left(\sum_j w_{i,\cdot} x_j\right)$$

  - where $w_{i,\cdot}$ is the $i$-th line of $w$, containing the weights of the connections from the neuron $i$ at the input layer. The activation function $g$ is the identity.
- $w_{i,\cdot}$ and $\mathbf{x}$ must be normalized.

- Each neuron of the competitive layer is associated with a set of other neurons to form a set of "neighborhoods".

- After receiving a certain input, some neurons will be excited enough to fire.

- The neuron whose response is maximum will have the ability to adjust the weights of the neurons in its neighborhood, including itself.

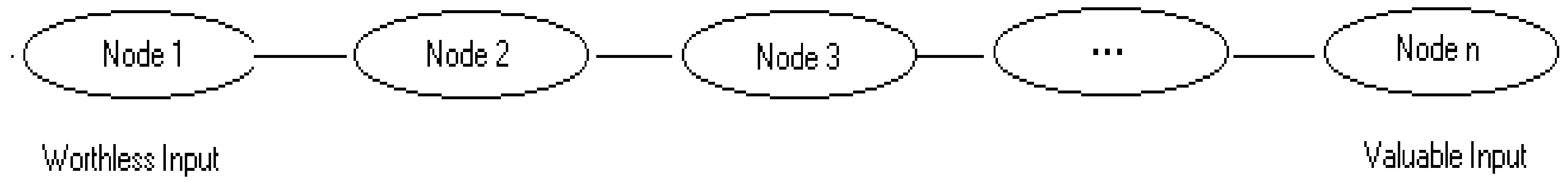
- Let $k$ be the winning neuron; for each neuron $i$ in its neighborhood

$$w_{i,.} \Leftarrow w_{i,.} + \eta \, (x - w_{i,.})$$

## Output level

- The organization of the output level depends strictly on the application.
- For proper operation of the Kohonen network it is not necessary that it is present.
- If the nodes are located along a single dimension, the output can be interpreted as a continuous

Organization of nodes along a single axis:



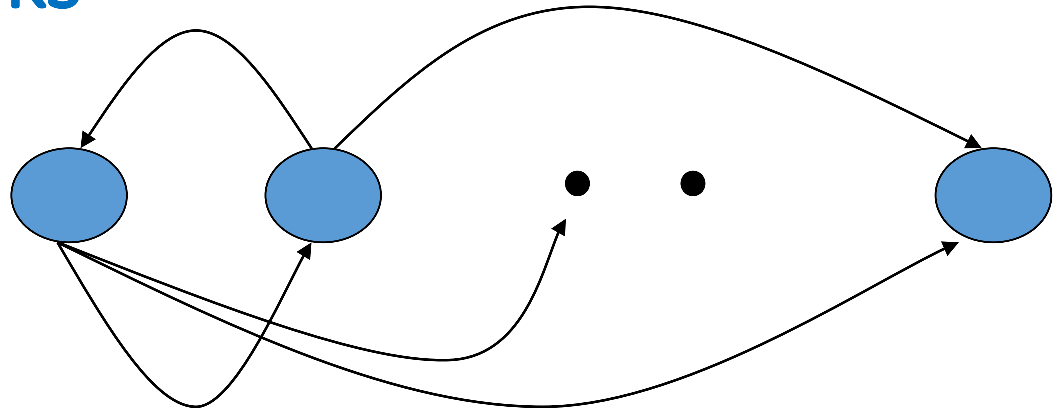
## Advantages

- Great for clustering problems
- Can reduce the computational complexity of a problem
- Very sensitive to frequent inputs
- Can create new ways to discover data bindings
- No supervision required

## Disadvantages

- The system is a black box
- Network convergence is not guaranteed for networks having dimensions larger than two
- Many problems are difficult to represent with an SOM
- A large training set is required
- The associations determined by an SOM are not always easily interpretable

## Hopfield Networks



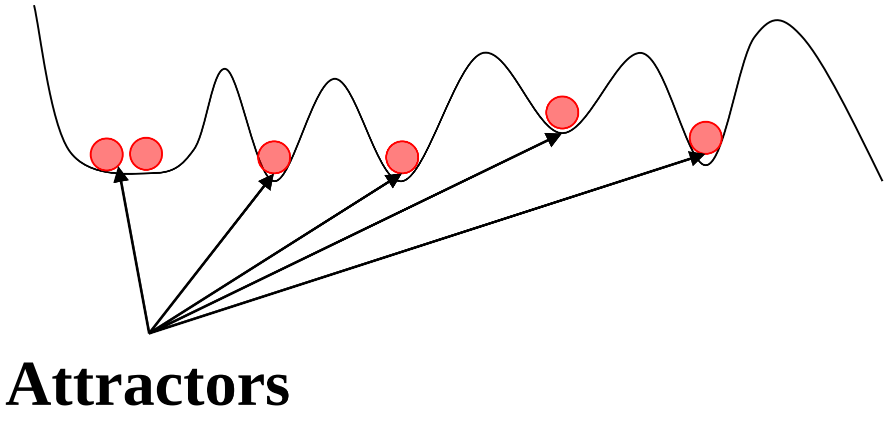
- Recurring network (all connected with all), one layer.
- Symmetric connections: $w_{ij} = w_{ji}$, $w_{ii} = 0$
- Threshold activation function:
  $x_i(t+1) = a_i = \text{step}( \sum_{i \neq j} w_{ij}\ x_j(t) )$
- At any given time, the activation values of all neurons define the state of the network $x(t)=(x_1(t) \ldots x_N(t))$
- As long as neurons apply the update rule, the state of the network evolves.

# Attractors

- In analogy to physical systems, an energy function can be defined for the grid:

$$E(x) = -\tfrac{1}{2} \sum_{i \neq j} w_{ij}\, x_i x_j$$

- It is shown that updating the state of a neuron can only decrease energy (Lyapunov function).
- Then, the grid evolves towards a state of minimum energy, so/called attractor.
- Attractors are stored patterns that network retrieves when it is initialized "near" the attractor



Note: with random asynchronous update one only has fixed points, otherwise one could have cycles

## Hebbian Learning

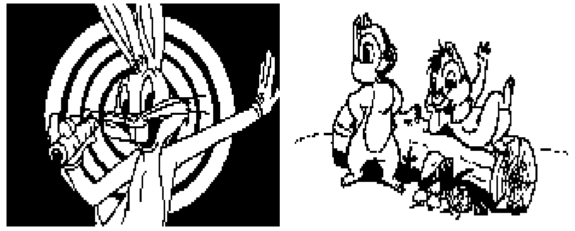
- The weights of the connections determine the attractors.
- How to fix weights so that the network stores a given set of patterns $A^1 \dots A^M$ ?
- Hebbian Rule:

$$w_{ij} = \sum_m (2A_i^m - 1)(2A_j^m - 1)$$

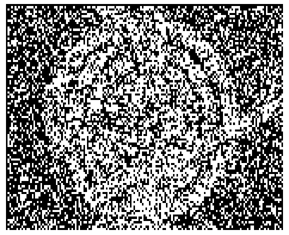
- The connections between neurons that are simultaneously active for a given pattern are reinforced, the others are weakened.
- In the most favorable case, the Hopfield network is able to learn a number of patterns equal to 10% of the number of neurons.

## Examples

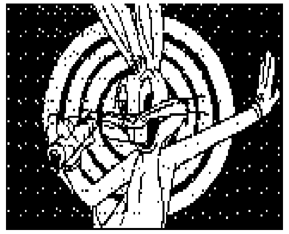
**Training**
2 images stored on the network



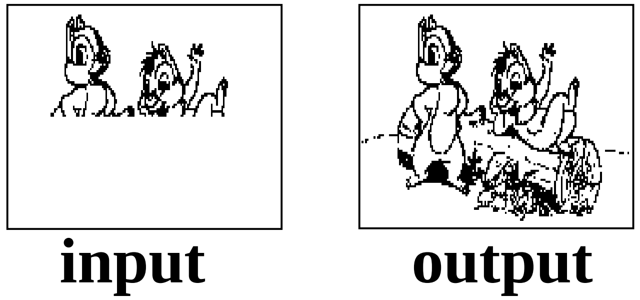
**Testing**
The network starts with this image



**Stable state (attractor)**
Reconstructed image



**Other examples:**

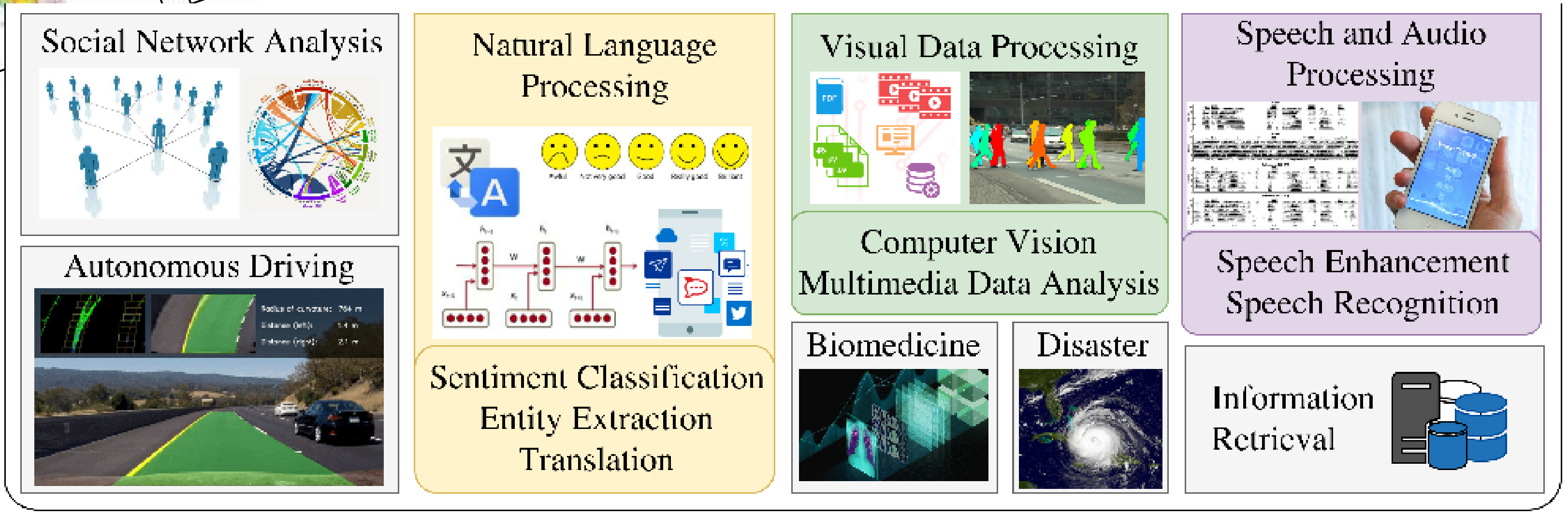


Source:

http://www2.psy.uq.edu.au/~brainwav/Manual/Hopfield.html

## Deep learning applications

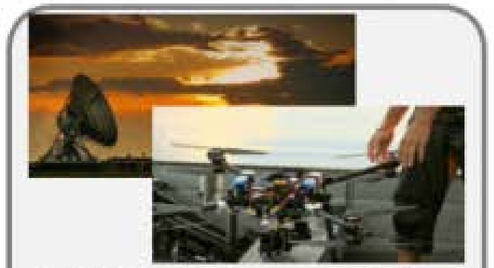
Image Classification
Object Detection
COMPUTER VISION
Voice Recognition
Language Translation
SPEECH & AUDIO
Recommendation Engines
Sentiment Analysis
NATURAL LANGUAGE PROCESSING
Translate



# Deep learning applications

## Aerospace, Defense and Communications

Communications devices, security


Multi-standard communications receivers, drone recognition

## Consumer Electronics and Digital Health

Voice assistants

Digital health

## Automotive

Voice control enabled Infotainment

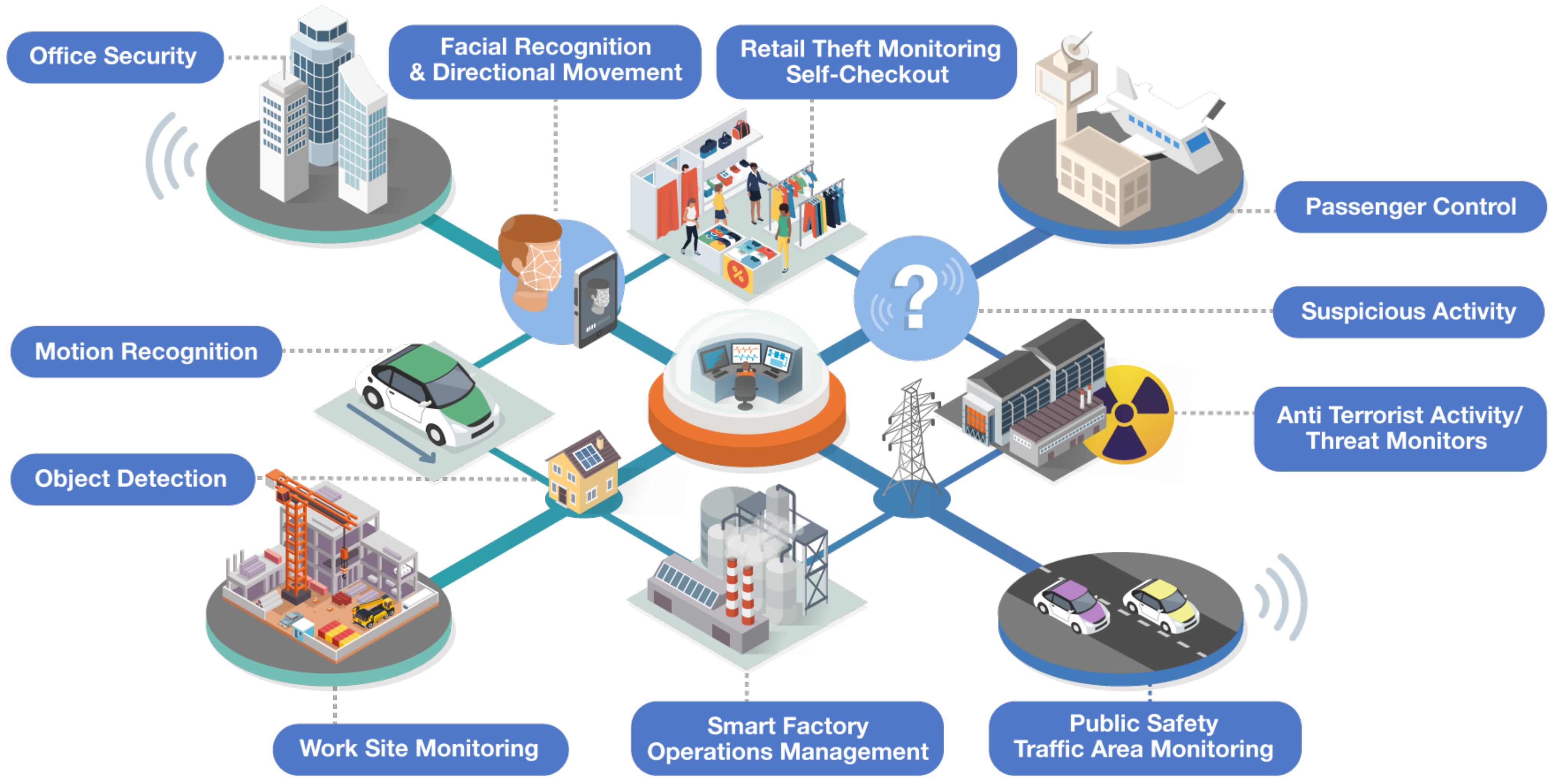
Sensor processing, automated driving

## Industrial Automation

Condition monitoring

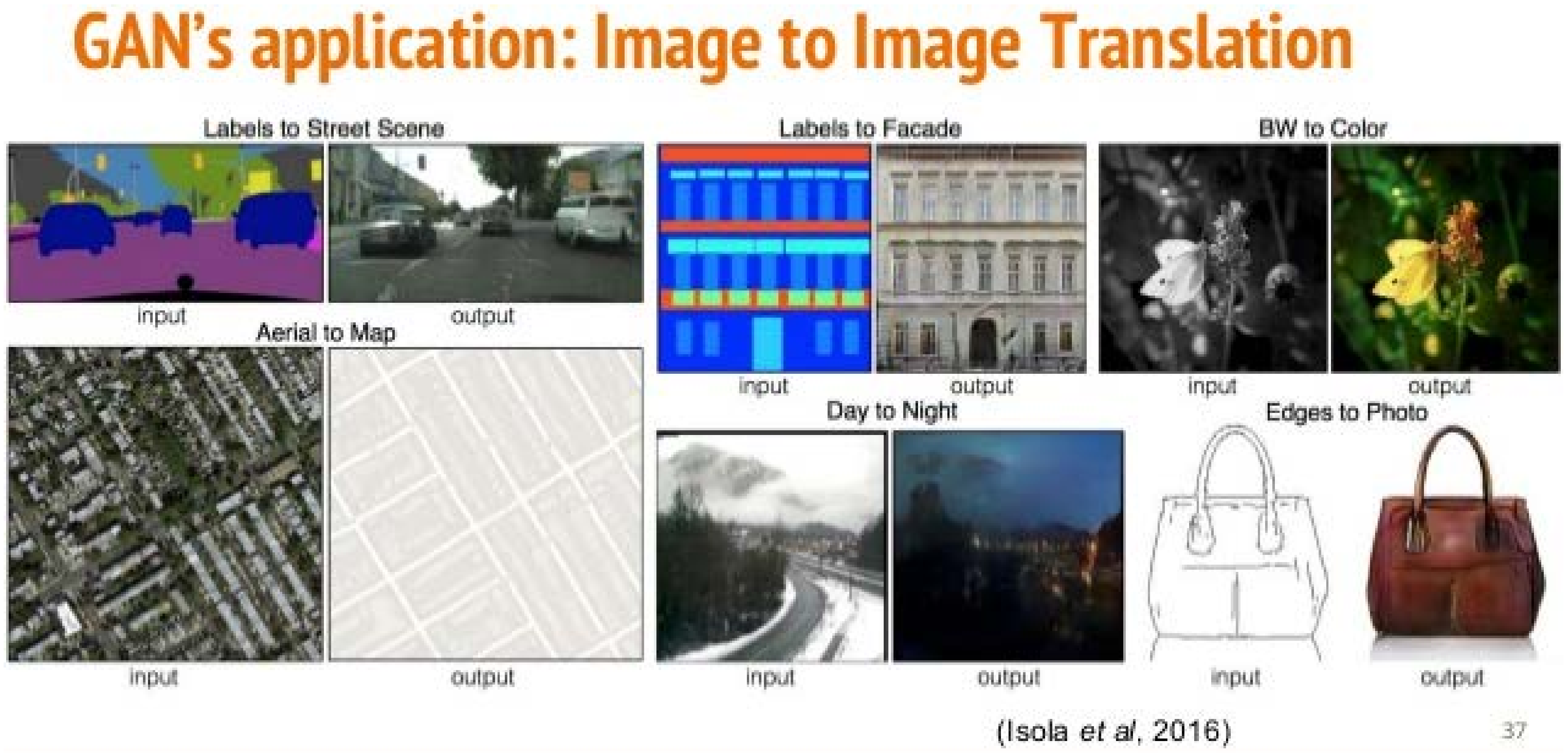
Predictive maintenance

# Deep learning applications

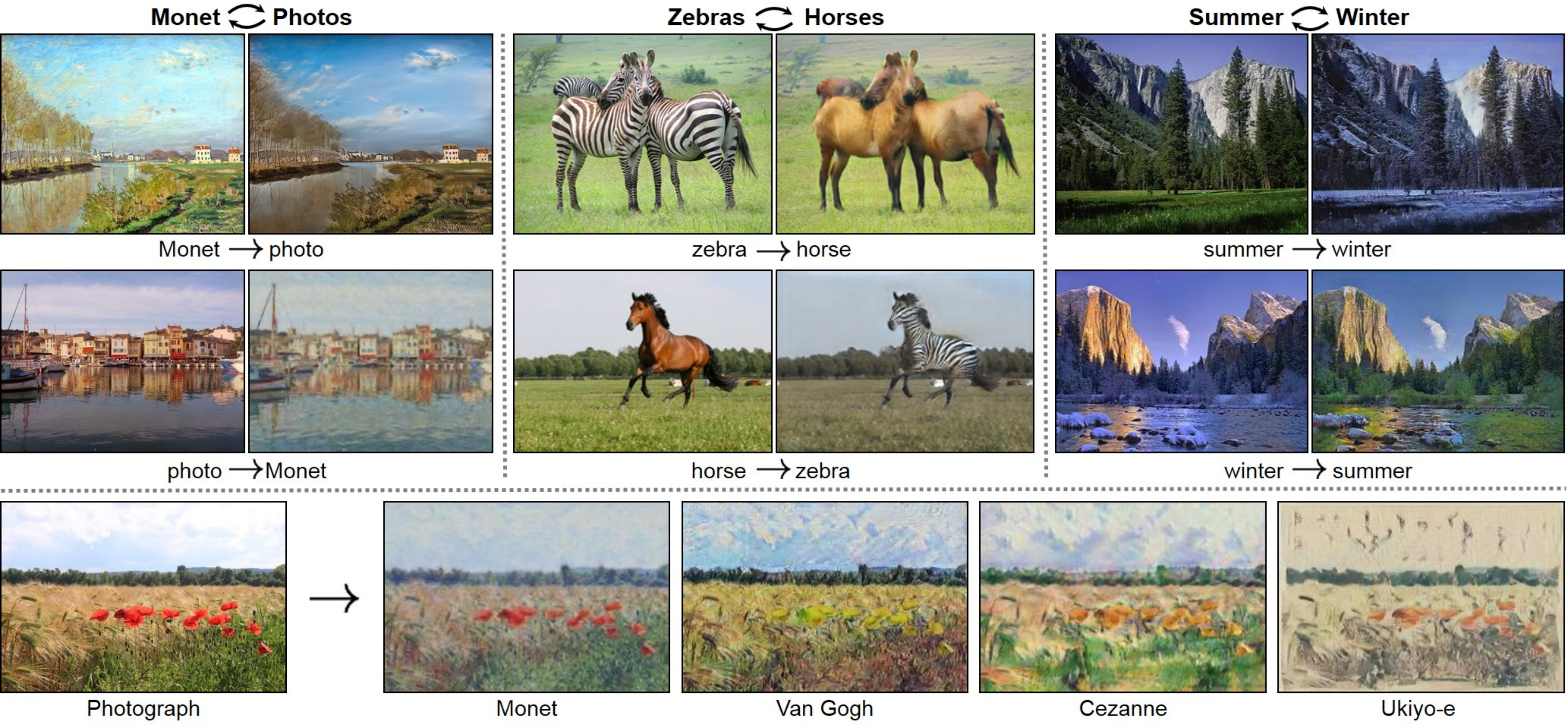

Office Security
Facial Recognition & Directional Movement
Retail Theft Monitoring Self-Checkout
Passenger Control
Suspicious Activity
Motion Recognition
Anti Terrorist Activity/ Threat Monitors
Object Detection
Work Site Monitoring
Smart Factory Operations Management
Public Safety Traffic Area Monitoring

# Deep learning applications

## Deep learning applications



# Deep learning applications



# Deep learning applications - Deepfake

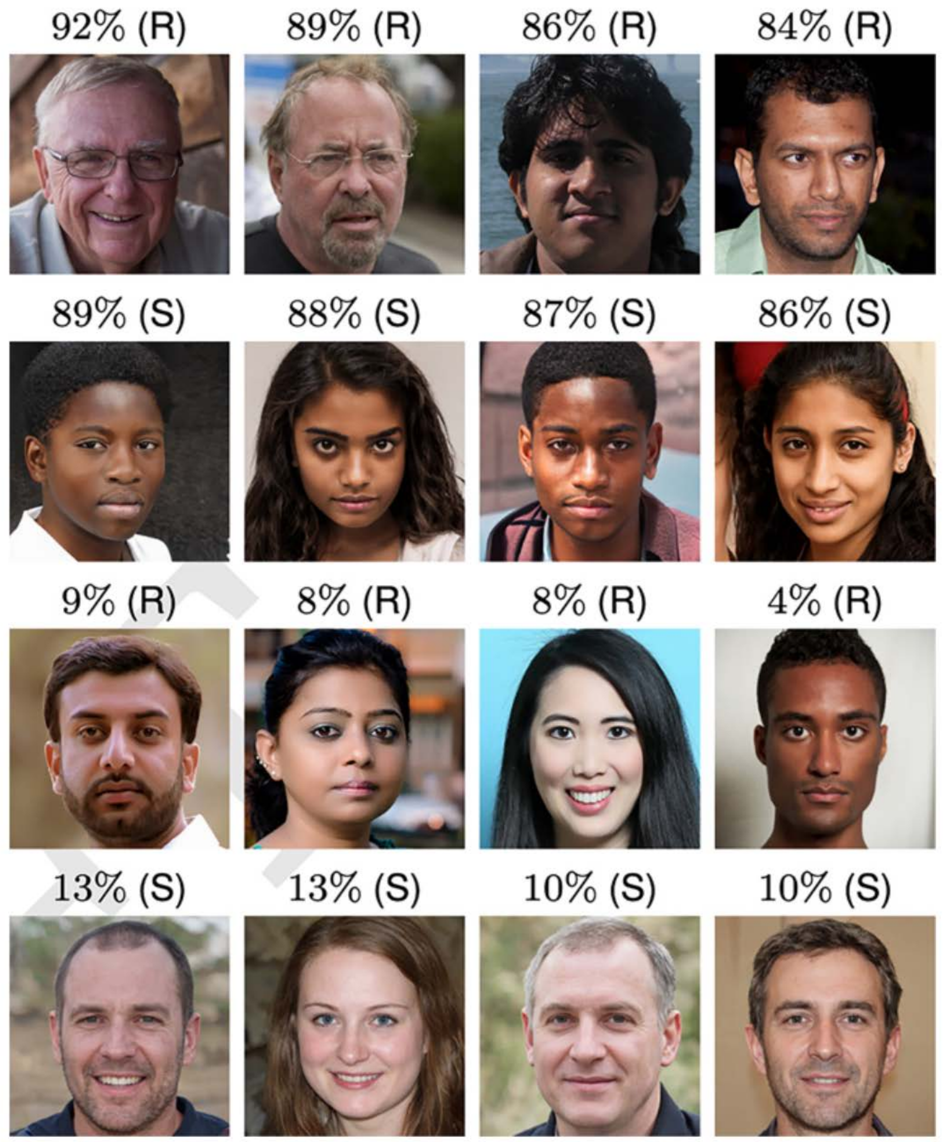


**Fig. 1.** The most (*Top* and *Upper Middle*) and least (*Bottom* and *Lower Middle*) accurately classified real (R) and synthetic (S) faces.

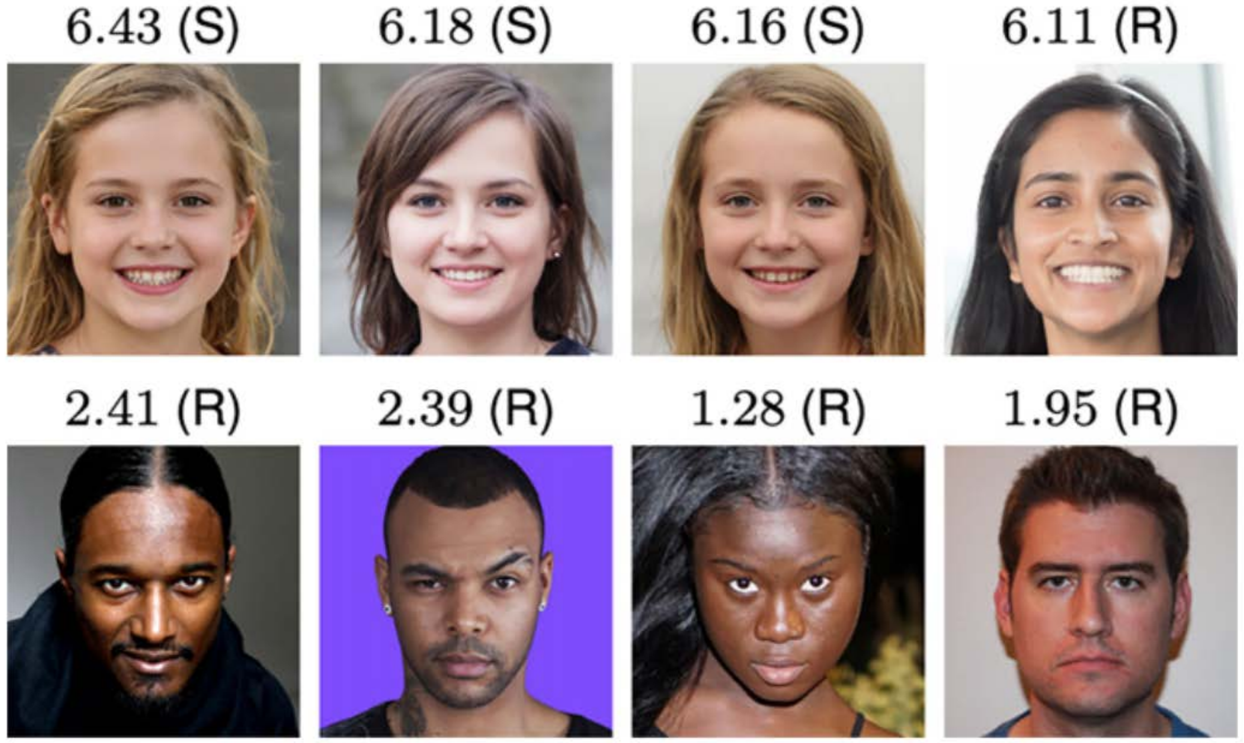


**Fig. 3.** The four most (*Top*) and four least (*Bottom*) trustworthy faces and their trustworthy rating on a scale of 1 (very untrustworthy) to 7 (very trustworthy). Synthetic faces (S) are, on average, more trustworthy than real faces (R).