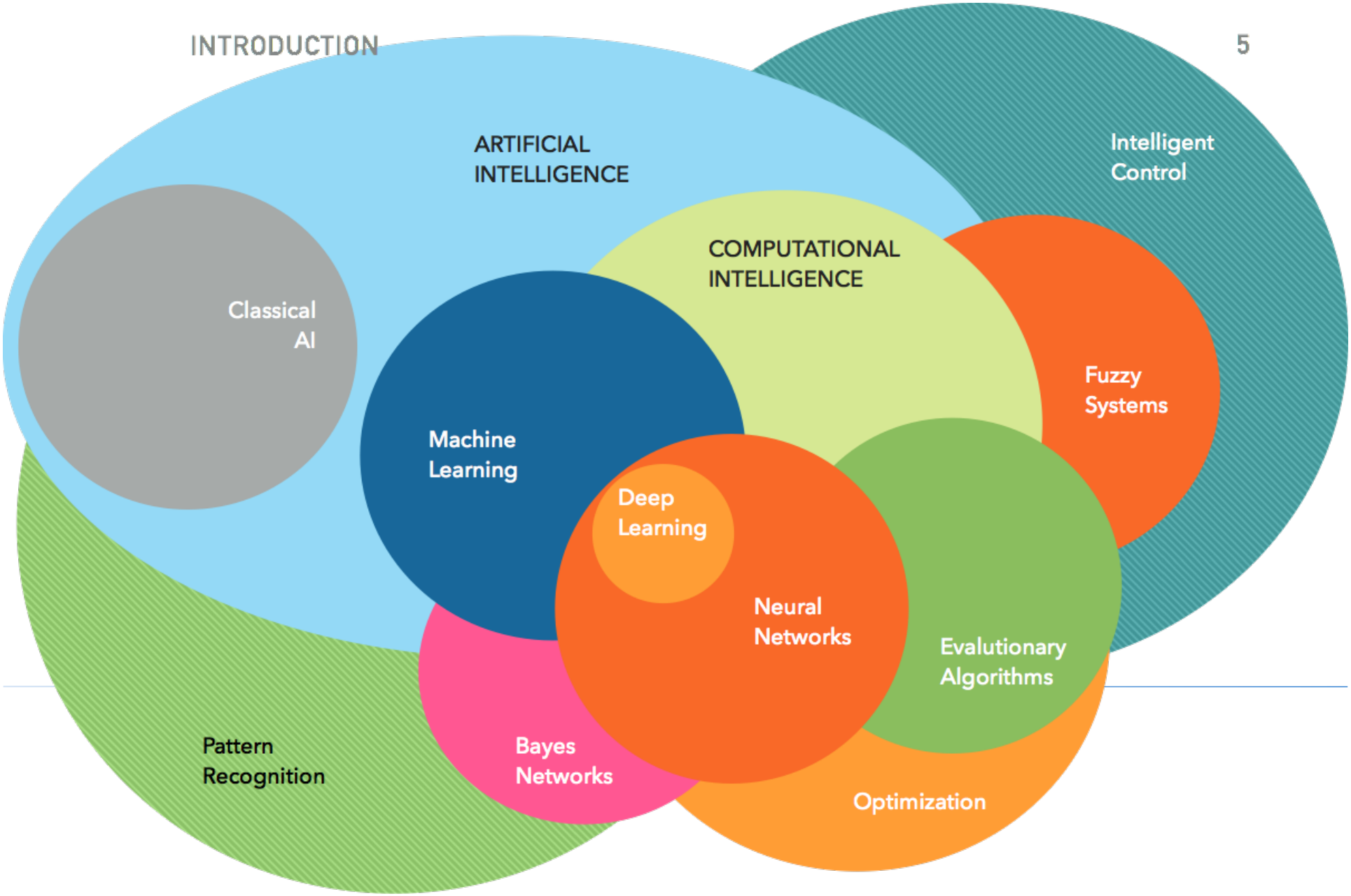
EE496 : COMPUTATIONAL INTELLINGENCE
NN01 : NEURAL NETWORKS INTRODUCTION

UGUR HALICI
METU: Department of Electrical and Electronics Engineering (EEE)
METU-Hacettepe U: Neuroscience and Neurotechnology (NSNT)


ARTIFICIAL
INTELLIGENCE
Intelligent
Control
Classical
AI
COMPUTATIONAL
INTELLIGENCE
Fuzzy
Systems
Machine
Learning
Deep
Learning
Neural
Networks
Evalutionary
Algorithms
Pattern
Recognition
Bayes
Networks
Optimization

Conventional computers vs. The Brain

	Computer	Brain
processing units	1 CPU (10^9 transistors)	10^{11} neurons
storage capacity	10^9 Bytes RAM, 10^{10} Bytes nonvolatile memory	10^{11} neurons, 10^{14} synapses
processing speed	10^{-8} sec.	10^{-3} sec.
neural updates per sec.	10^5	10^{14}

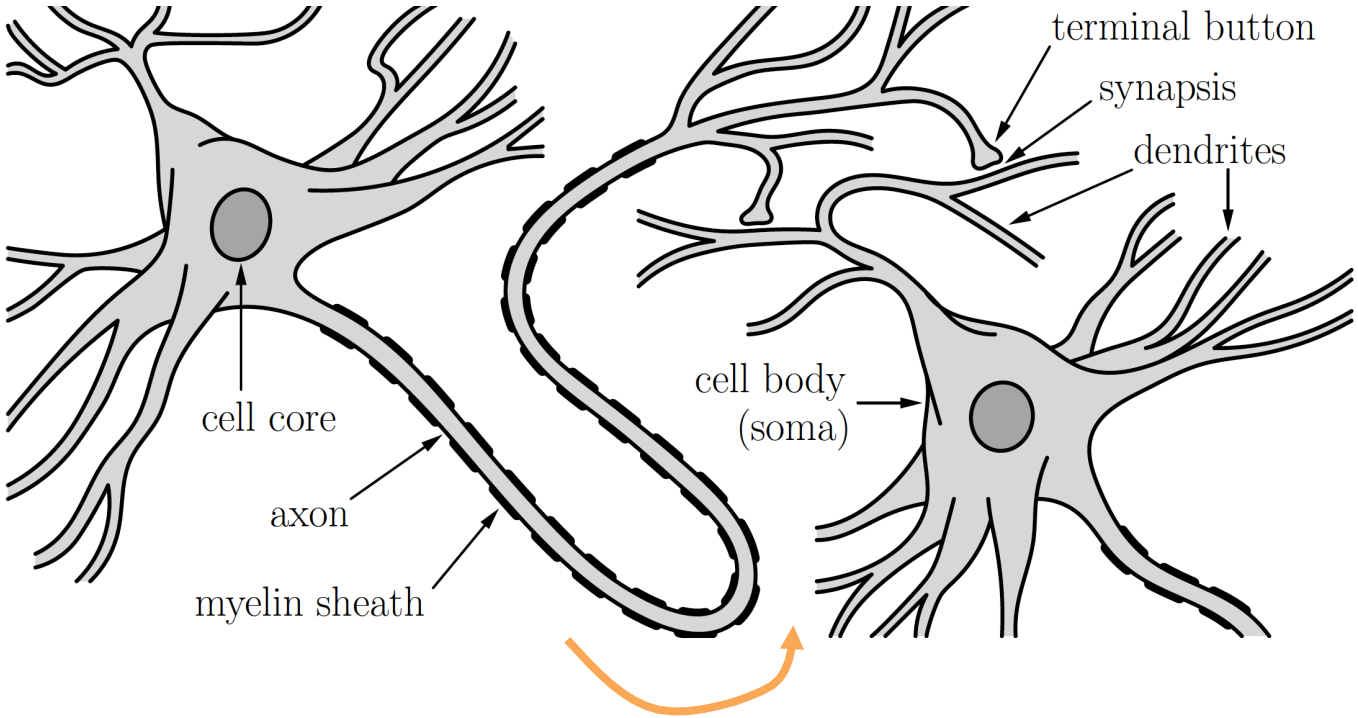
Conventional computers vs. The Brain

- Note: the switching times of the human brain are quite slow, being only 10^{-3} sec., but updates are processed in parallel. In contrast, serial PC simulations take several hundreds of processing cycles for one update.
- Advantages of neural networks:
  - great processing speed by making massively use of parallel processing
  - even after partial failure the network is still in service (fault tolerant)
  - with increasing amount of failing neurons just slow failure of entire system
  - well-suited for inductive learning , which is a process where the learner discovers rules by observing examples.
- Thus it seems promising to emulate these advantages by using artificial neural networks.

Biological Background

action pottentials arriving to synapses affects soma potential

terminal buttons are connected to other neurons through synapses



impulses carried away from cell body through axon

Conventional computers vs. The Brain

(Very) simplified description of neural information processing

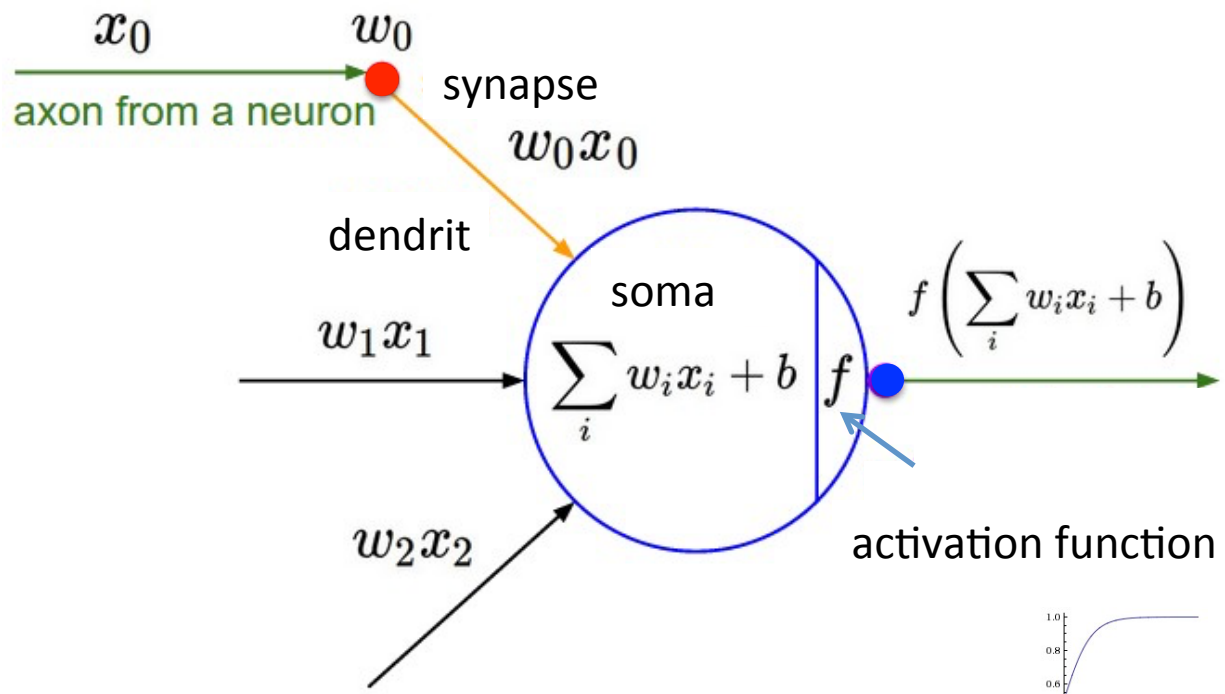
- Axon terminal releases chemicals, called **neurotransmitters.**
- These act on the membrane of the receptor dendrite to change its polarization. (The inside is usually 70mV more negative than the outside.)
- Decrease in potential difference: **excitatory** synapse

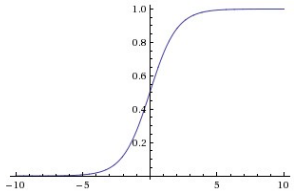
  Increase in potential difference: **inhibitory** synapse
- If there is enough net excitatory input, the axon is depolarized.
- The resulting **action potential** travels along the axon.

  (Speed depends on the degree to which the axon is covered with myelin.)
- When the action potential reaches the terminal buttons,

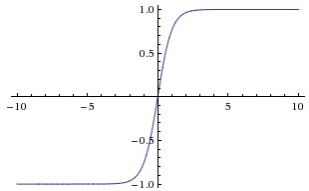
  it triggers the release of neurotransmitters.

Artificial Neuron (Mc Culloch Pitts)





sigmoid



tanh

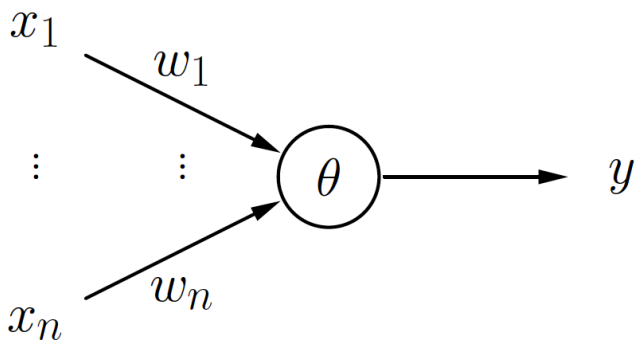
threshold function: is the sigmoid in which the slope at the origin goes to infinity.

Threshold Logic Units

A **Threshold Logic Unit (TLU)** is a processing unit for numbers with n inputs $x_1, \ldots, x_n$ and one output y. The unit has a threshold θ and each input x_i is associated with a weight w_i.

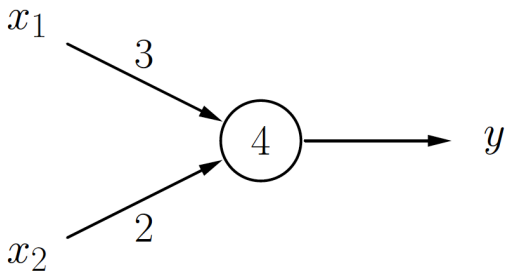
A threshold logic unit computes the function

$$y = \begin{cases} 1, & \text{if} \quad \vec{x}\vec{w} = \sum_{i=1}^{n} w_i x_i \geq \theta, \\ 0, & \text{otherwise.} \end{cases}$$



Threshold Logic Units: Examples

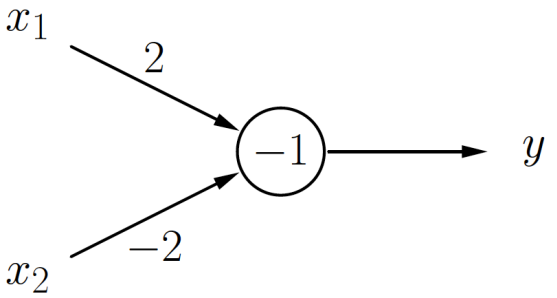
Threshold logic unit for the conjunction $x_1 \wedge x_2$.



x_1	x_2	$3x_1 + 2x_2$	y
0	0	0	0
1	0	3	0
0	1	2	0
1	1	5	1

Exercise: Let the weights be the same for both inputs, say it is 1, and then determine the threshold for "AND" and "OR" operations. Also implement "NOT" operation.

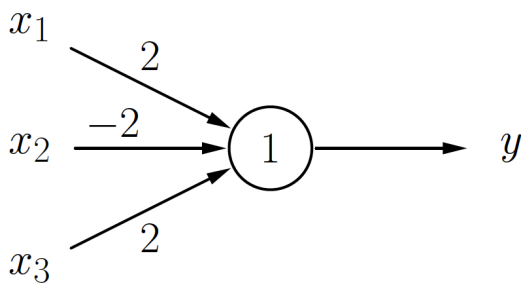
Threshold logic unit for the implication $x_2 \rightarrow x_1$.



x_1	x_2	$2x_1 - 2x_2$	y
0	0	0	1
1	0	2	1
0	1	-2	0
1	1	0	1

Threshold Logic Units: Examples

Threshold logic unit for (x1 ∧ x2) ∨ (x1 ∧ x3) ∨ (x2 ∧ x3).



x_1	x_2	x_3	$\sum_i w_i x_i$	y
0	0	0	0	0
1	0	0	2	1
0	1	0	−2	0
1	1	0	0	0
0	0	1	2	1
1	0	1	4	1
0	1	1	0	0
1	1	1	2	1

Threshold Logic Units: Geometric Interpretation

Straight lines are usually represented in one of the following forms:

Explicit Form: $g \equiv x_2 = bx_1 + c$

Implicit Form: $g \equiv a_1x_1 + a_2x_2 + d = 0$

Point-Direction Form: $g \equiv \vec{\mathbf{x}} = \vec{\mathbf{p}} + k\vec{\mathbf{r}}$

Normal Form: $g \equiv (\vec{\mathbf{x}} - \vec{\mathbf{p}})\vec{\mathbf{n}} = \mathbf{0}$

with the parameters:

b : Gradient of the line

c: Section of the x_2 axis

$\vec{p}$: Vector of a point of the line (base vector)

$\vec{r}$: Direction vector of the line

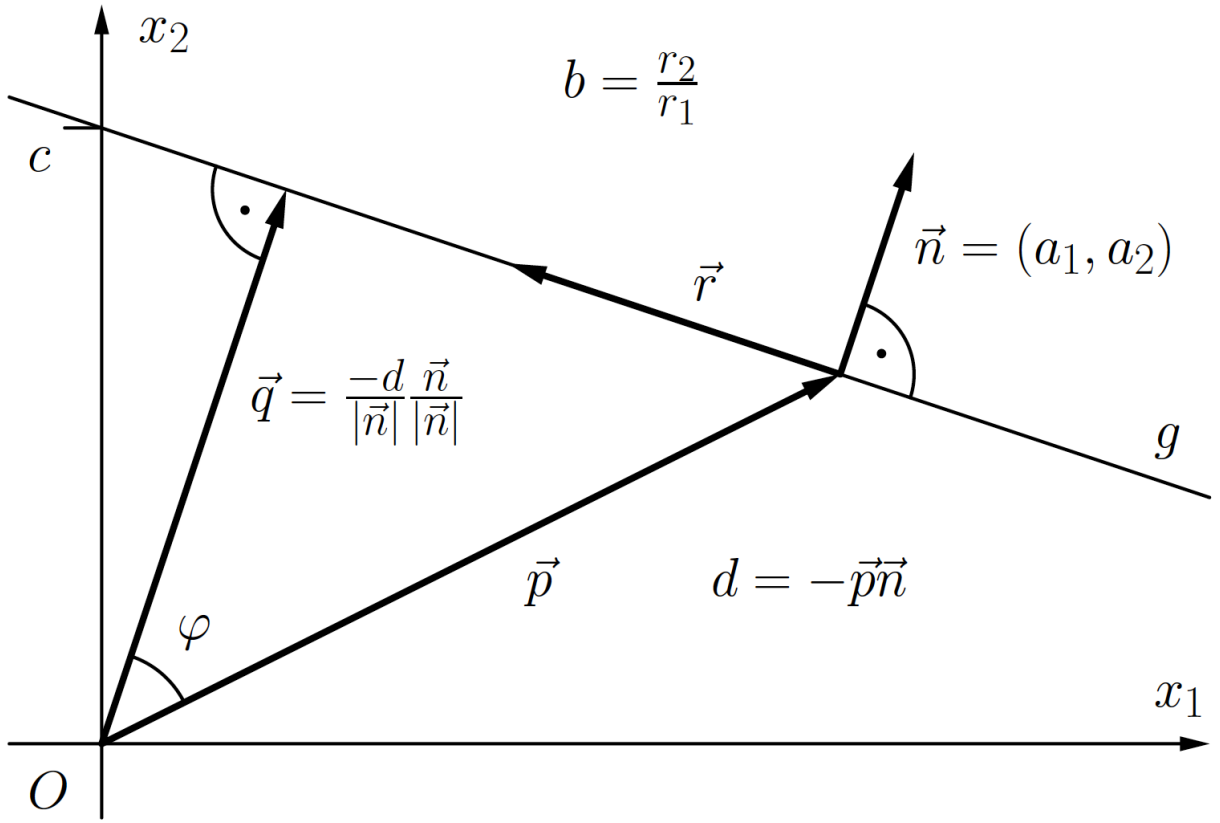
$\vec{n}$: Normal vector of the line

Threshold Logic Units: Geometric Interpretation

A straight line and its defining parameters:

$x_2 = bx_1 + c$

$a_1x_1 + a_2x_2 + d = 0$

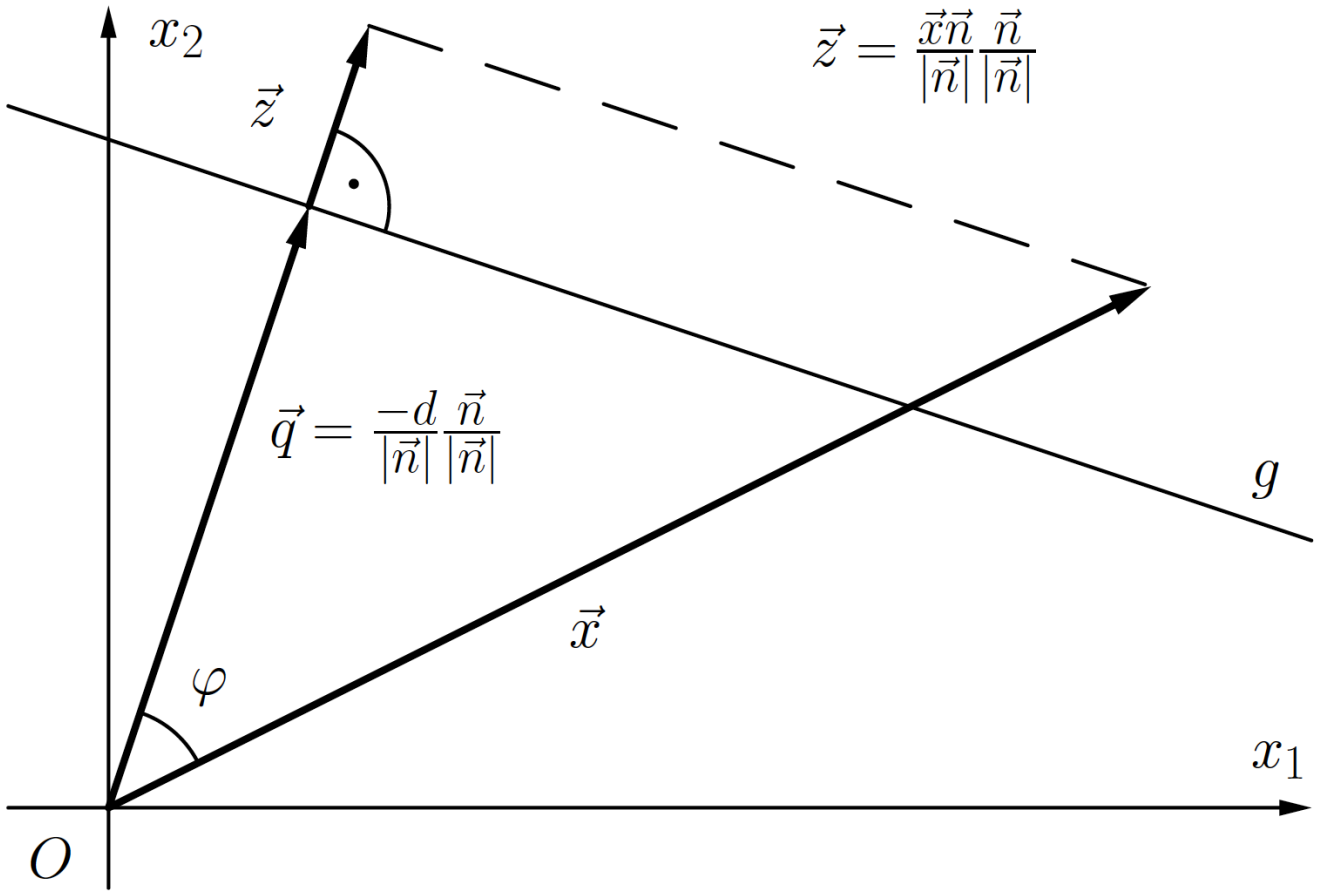


Threshold Logic Units: Geometric Interpretation

How to determine the side on which a point $\vec{\mathbf{x}}$ lies.

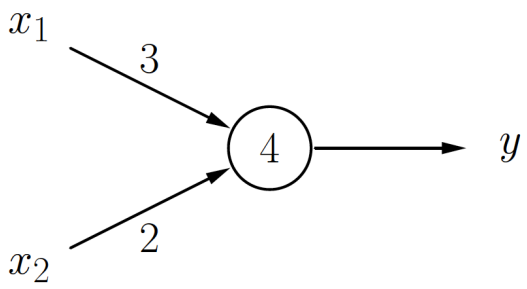
$x_2 = bx_1 + c$

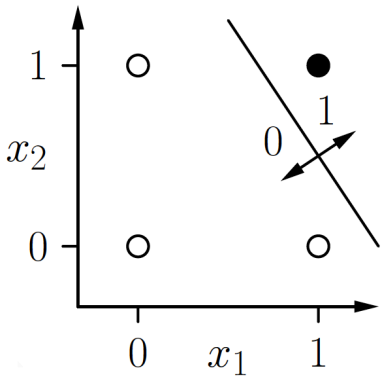
$a_1x_1 + a_2x_2 + d = 0$



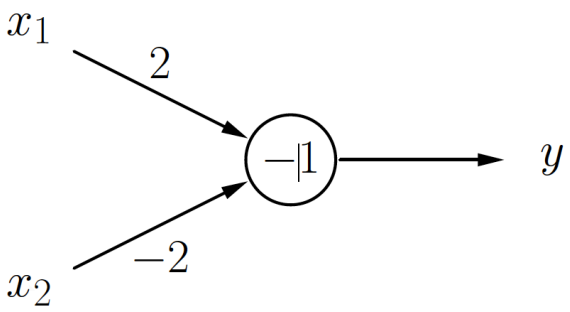
Threshold Logic Units: Geometric Interpretation

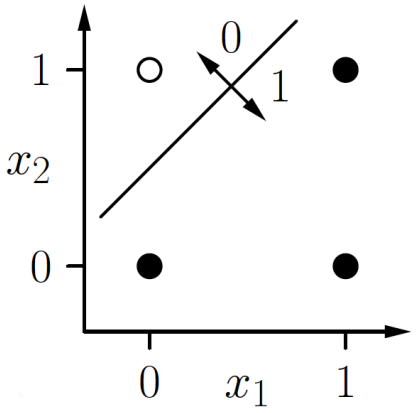
Threshold logic unit for $x_1 \wedge x_2$.





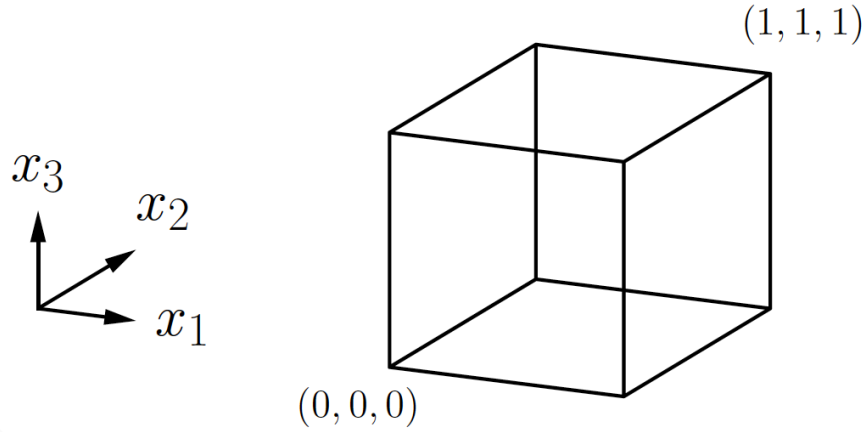
A threshold logic unit for x2 → x1.



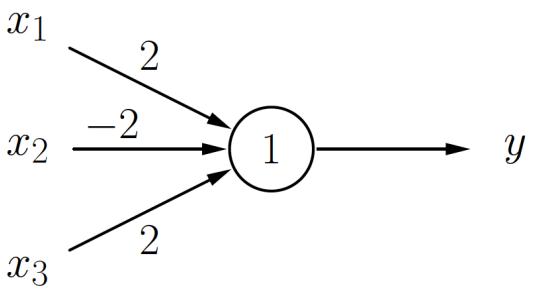


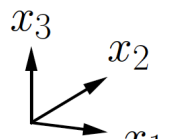
Threshold Logic Units: Geometric Interpretation

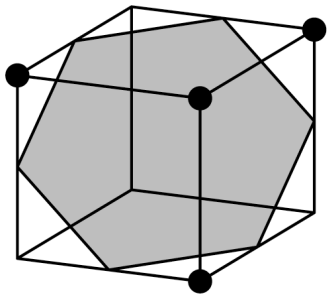
Visualization of 3-dimensional Boolean functions:



Threshold logic unit for (x1 ∧ x2) ∨ (x1 ∧ x3) ∨ (x2 ∧ x3).







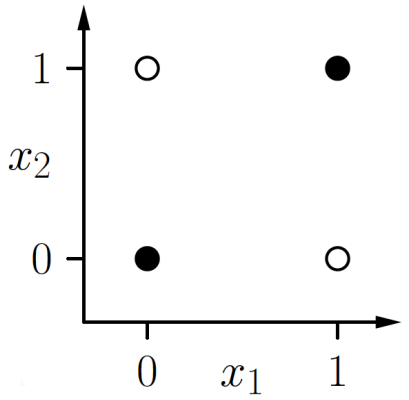
Threshold Logic Units: linear separability

- We call two sets of points in an n-dimensional space linearly separable, if they can separated by an (n-1)-dimensional hyperplane. One of these sets may contain points lying on the hyperplane, too.
- A boolean function is called linearly separable, if the set of points of 0 and the set of points of 1 are linearly separable.

Example : The biimplication problem $x_1 \leftrightarrow x_2$ (i.e. : $x_1x_2 \vee x_1'x_2'$)

There is no separating line.

x_1	x_2	y
0	0	1
1	0	0
0	1	0
1	1	1



Threshold Logic Units: Limitations

Total number and number of linearly separable Boolean functions.

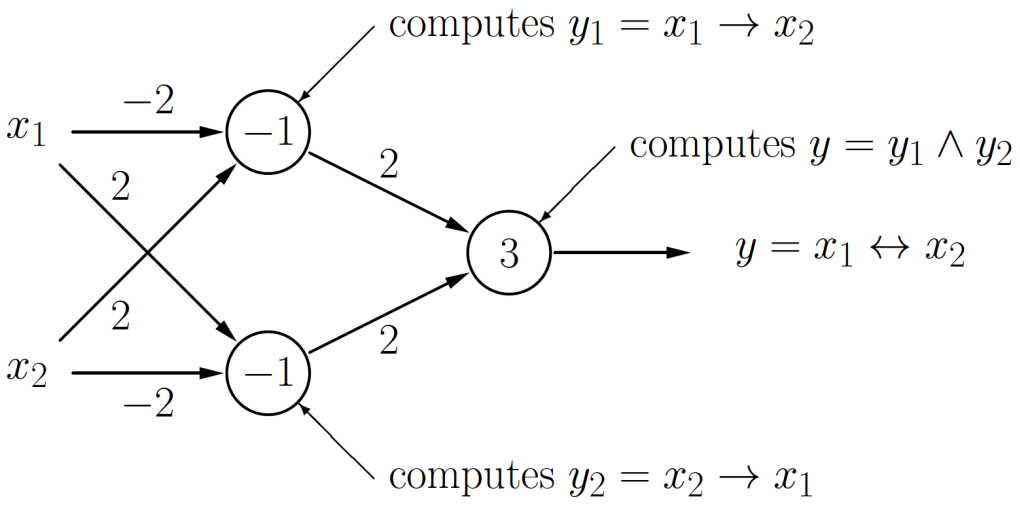
inputs	Boolean functions	linearly separable functions
1	4	4
2	16	14
3	256	104
4	65536	1774
5	$4.3 \cdot 10^9$	94572
6	$1.8 \cdot 10^{19}$	$5.0 \cdot 10^6$

- For many inputs a threshold logic unit can compute almost no functions.
- Networks of threshold logic units are needed to overcome the limitations.

Networks of Threshold Logic Units

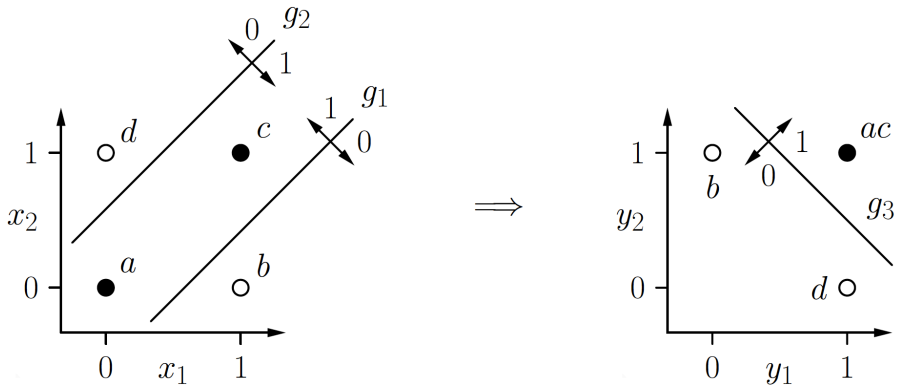
Solving the biimplication with a network.

Idea: logical decomposition x1 ↔ x2 ≡ (x1 → x2) ⋀ (x2 → x1)



Networks of Threshold Logic Units

Solving the biimplication problem: Geometric interpretation



- The first layer computes new Boolean coordinates for the points.
- After the coordinate transformation the problem is linearly separable.
- Because we are able to implement each of AND, OR and NOT logic functions using single threshold units, we are able to implement any logic function as a neural network, using a combination of AND, OR, NOT operations on variables.

Training Threshold Logic Units : Delta Rule

- Geometric interpretation provides a way to construct threshold logic units with 2 and 3 inputs, but:
  - Not an automatic method (human visualization needed).
  - Not feasible for more than 3 inputs.
- General idea of automatic training:
  - Start with random values for weights and threshold.
  - Determine the error of the output for a set of training patterns.
  - Error is a function of the weights and the threshold:

    $e = e(w_1, \ldots, w_n, \theta)$.
  - Adapt weights and threshold so that the error gets smaller.
  - Iterate adaptation until the error vanishes.

Training Threshold Logic Units : Delta Rule

- Formal Training Rule: Let $\vec{x} = (x_1, \ldots, x_n)$ be an input vector of a threshold logic unit,
  - o the desired output for this input vector and
  - y the actual output of the threshold logic unit.
- If $y \neq o$, then the threshold θ and the weight vector $\vec{w} = (w_1, \ldots, w_n)$ are

$$\theta^{(\text{new})} = \theta^{(\text{old})} + \Delta\theta \quad \text{with} \quad \Delta\theta = -\eta(o - y),$$
$$\forall i \in \{1, \ldots, n\}: \quad w_i^{(\text{new})} = w_i^{(\text{old})} + \Delta w_i \quad \text{with} \quad \Delta w_i = \eta(o - y)x_i,$$

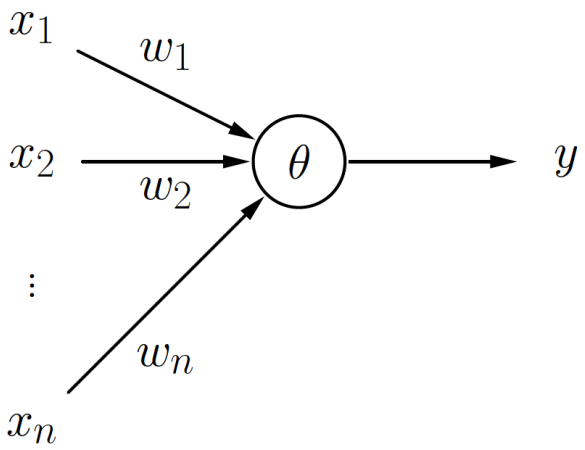
where η is a parameter that is called **learning rate**.

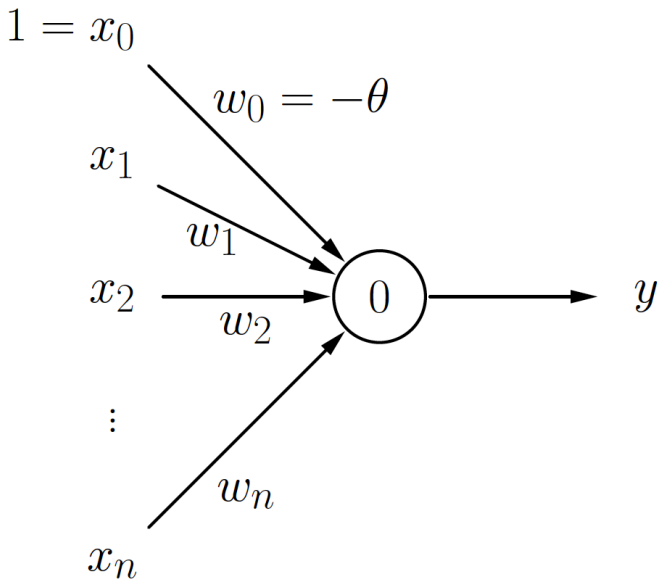
Training Threshold Logic Units : Delta Rule

- The given rule determines the severity of the weight changes. This procedure is called **Delta Rule** or **Widrow–Hoff Procedure** [Widrow and Hoff 1960].
  - **Online Training:** Adapt parameters after each training pattern.
  - **Batch Training:** Adapt parameters only at the end of each epoch, i.e. after a traversal of all training patterns.

Training Threshold Logic Units : Delta Rule

- Turning the threshold value into a weight:





$$\sum_{i=1}^{n} w_i x_i \geq \theta$$

$$\sum_{i=1}^{n} w_i x_i - \theta \geq 0$$

Training Threshold Logic Units : Delta Rule

```
(* L: traning set η: learning rate, →x: pattern, o: desired output
   y: actual output, e: error *)

procedure online training (var →w, var θ, L, η);
var y, e;                                          (* output, sum of errors *)
begin
    repeat
        e := 0;                                    (* initialize the error sum *)
        for all (→x, o) ∈ L do begin               (* traverse the patterns *)
            if (→w→x ≥ θ) then y := 1;             (* compute the output *)
                          else y := 0;             (* of the threshold logic unit *)
            if (y≠ o) then begin                   (* if the output is wrong *)
                θ := θ − η(o − y);                 (* adapt the threshold *)
                →w := →w + η(o − y)→x;             (* and the weights *)
                e := e + |o − y|;                  (* sum the errors *)
            end;
        end;
    until (e = 0);                                 (* repeat the computations *)
end;                                               (* until the error vanishes *)
```

Training Threshold Logic Units : Delta Rule

```
(* L: traning set, η: learning rate, →x: pattern, o: desired output *)

procedure batch training (var →w, var θ, L, η);
var y, e,                                          (* output, sum of errors *)
    θc, →wc;                                       (* summed changes *)
begin
    repeat
        e := 0; θc := 0; →wc := →0;                (* initializations *)
        for all (→x, o) ∈ L do begin               (* traverse the patterns *)
            if ( →w→x ≥ θ) then y := 1;            (* compute the output *)
                           else  y := 0;           (* of the threshold logic unit *)
            if (y≠ o) then begin                   (* if the output is wrong *)
                θc := θc − η(o − y);               (* sum the changes of the *)
                →wc := →wc + η(o − y)→x;           (* threshold and the weights *)
                e := e + |o − y|;                  (* sum the errors *)
            end;
        end;
        θ := θ + θc;                               (* adapt the threshold *)
        →w := →w + →wc;                            (* and the weights *)
    until (e = 0);                                 (* repeat the computations *)
end;                                               (* until the error vanishes *)
```

Training Threshold Logic Units : Convergence

Convergence Theorem: Let $L = \{(\vec{\mathbf{x}}_1, o_1), \ldots (\vec{\mathbf{x}}_m, o_m)\}$ be a set of training patterns, each consisting of an input vector $\vec{\mathbf{x}}_i \in R^n$ and a desired output $o_i \in \{0, 1\}$.

Furthermore, let $L_0 = \{(\vec{\mathbf{x}}, o) \in L \mid o = 0\}$ and $L_1 = \{(\vec{\mathbf{x}}, o) \in L \mid o = 1\}$. If L_0 and L_1 are linearly separable, i.e., if $\vec{\mathbf{w}} \in R_n$ and $\theta \in R$ exist, such that

$\forall(\vec{\mathbf{x}}, 0) \in L_0 : \vec{\mathbf{w}}\vec{\mathbf{x}} < \theta$ and

$\forall(\vec{x}, 1) \in L_1 : \vec{\mathbf{w}}\vec{\mathbf{x}} \geq \theta$,

then online as well as batch training terminate.

- The algorithms terminate only when the error vanishes.
- Therefore the resulting threshold and weights must solve the problem.
- For not linearly separable problems the algorithms do not terminate.

Training Threshold Logic Units : Convergence

- Single threshold logic units have strong limitations:

  They can only compute linearly separable functions.
- Networks of threshold logic units can compute arbitrary Boolean functions.
- Training single threshold logic units with the delta rule is fast and guaranteed to find a solution if one exists.
- Networks of threshold logic units cannot be trained in this way, because
  - there are no desired values for the neurons of the first layer,
  - the problem can usually be solved with different functions computed by the neurons of the first layer.
- When this situation became clear,

  neural networks were seen as a “research dead end”

  .. but it was NOT..