



Neuronale Netze

Prof. Dr. Rudolf Kruse

kruse@iws.cs.uni-magdeburg.de

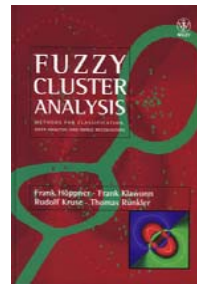
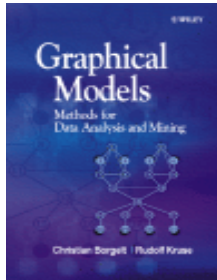
Arbeitsgruppe Computational Intelligence
Institut für Wissens- und Sprachverarbeitung
Fakultät für Informatik
Otto-von-Guericke Universität Magdeburg



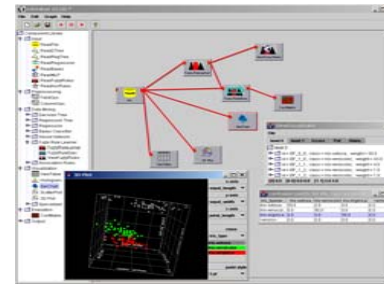
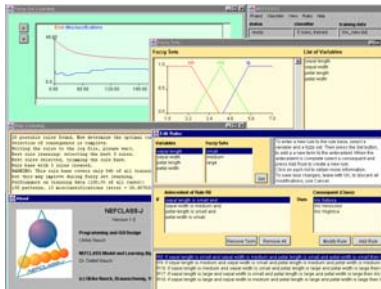
- Vorlesung: Neuronale Netze
 - Termin: Donnerstags, 09:15–10:45, G29-307
 - Dozent: Prof. Dr. Rudolf Kruse
 - Sprechstunde: Mittwochs, 11–12 Uhr, G29-008
 - bevorzugt erreichbar per email: kruse@iws.cs.uni-magdeburg.de
- Übungen: Neuronale Netze
 - Übungsleiter: Georg Ruß
 - Termine:
 - Montags, 13:15–14:45, 22A-218
 - Montags, 15:00–16:30, 22A-111
- Literatur
 - C. Borgelt, F. Klawonn, R. Kruse, D. Nauck:
Neuro-Fuzzy-Systeme, 3. Auflage, Vieweg, 2003

Geschäftsmodell des „Lehrstuhl Kruse“

Entwicklung neuer Methoden in Forschungsprojekten (Neuro-Fuzzy, Bayes,...)



Entwicklung frei verfügbarer Analyse-Software (z.B. „Information Miner auf CEBIT“)



Nutzung in Industrieprojekten, derzeit:

- Teilebedarfsplanung mit Markov-Networks (Volkswagen)
- Information Mining (BMW, Daimler Chrysler)
- Bayes-Methoden im Finanzwesen (Deutscher Giro und Sparkassenverband)
- Analyse der Kundenzufriedenheit (British Telecom)

CI Core Technologies

- Neural Nets (NN)
- Fuzzy Logic (FL)
- Probabilistic Reasoning (PR)
- Genetic Algorithms (GA)
- Hybrid Systems

Related Technologies

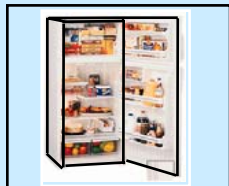
- Statistics (Stat.)
- Artificial Intelligence (AI):
 - Case-Based Reasoning (CBR)
 - Rule-Based Expert Systems (RBR)
 - Machine Learning (Induction Trees)
 - Bayesian Belief Networks (BBN)

Applications

- Classification
 - Monitoring/Anomaly Detection
 - Diagnostics
 - Prognostics
 - Configuration/Initialization
- Prediction
 - Quality Assessment
 - Equipment Life Estimation
- Scheduling
 - Time/Resource Assignments
- Control
 - Machine/Process Control
 - Process Initialization
 - Supervisory Control
- DSS/Auto-Decisioning
 - Cost/Risk Analysis
 - Revenue Optimization

Broad technology base and wide range of application tasks

Computational Intelligence / Soft Computing Applications



Appliances

- Preferred Service Contracts (Stat.)
- Call Center Support (CBR)



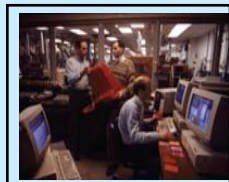
Capital Services

- Mortgage Collateral Evaluation (Fusion/FL/CBR)



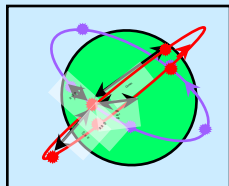
Financial Assurance

- GEFA LTC Preferred Customer (Stat./NN)
- GEFA Fixed Life Digital Underwriter (Stat, CBR, FL, GA)



Plastics

- Automated Color Matching (CBR)



LM Fed. Systems

- Scheduling Maintenance for Constellation of Satellites (GA)



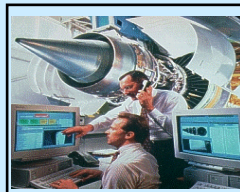
LM ORSS

- Vessel Management Syst. (AI/GA)



Medical Systems

- SPT Auto Analysis for MRI (FL)
- Reverse Engineering of Picker (FL)
- FE Analysis tool (FL)
- X-Ray error Logs Analysis (CBR)



Aircraft Engines

- Center for Remote Diagn. (CBR)
- Customer Response Center (CBR)
- Anomaly Detection (FL/Stat.)
- IMATE - Maintenance Advisor (NN/FL)
- Resolver Drift - Sensor Fusion (FL)



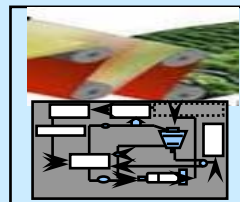
Transportation Systems

- Log from Transportation DB (CBR)
- Prototype Train Handling Cntrl. (FL/GA)
- Prototype Trend Analysis (Stat.)
- Embedded/Remote Diagnostics (BBN)



Power Gen. Systems

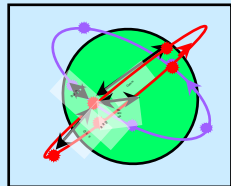
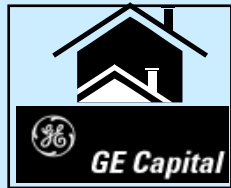
- Remote Anomaly Detection (Stat.)
- Embedded/Remote Diagnostics (BBN)
- Call Center Problem/Solution (CBR)



Industrial Systems

- Paper Web Breakage Prediction (NN/Stat./Induction)
- Control Mixing of Cement (FL/GA)

Enabling Technologies



Statistics

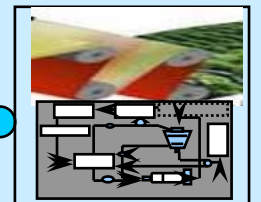
Statistical Models (CART, MARS)
Physics-Based Models
Data Mining

Soft Computing

Prob. Reasoning (BBN)
Neural Computing
Feedforward N Nets
Fuzzy Computing
Fuzzy Rules
Fuzzy Control
Fuzzy Clustering
Evolutionary Computing
Genetic Algorithms

AI

Rule Based Reasoning
Case Based Reasoning
Induction Trees
Fusion



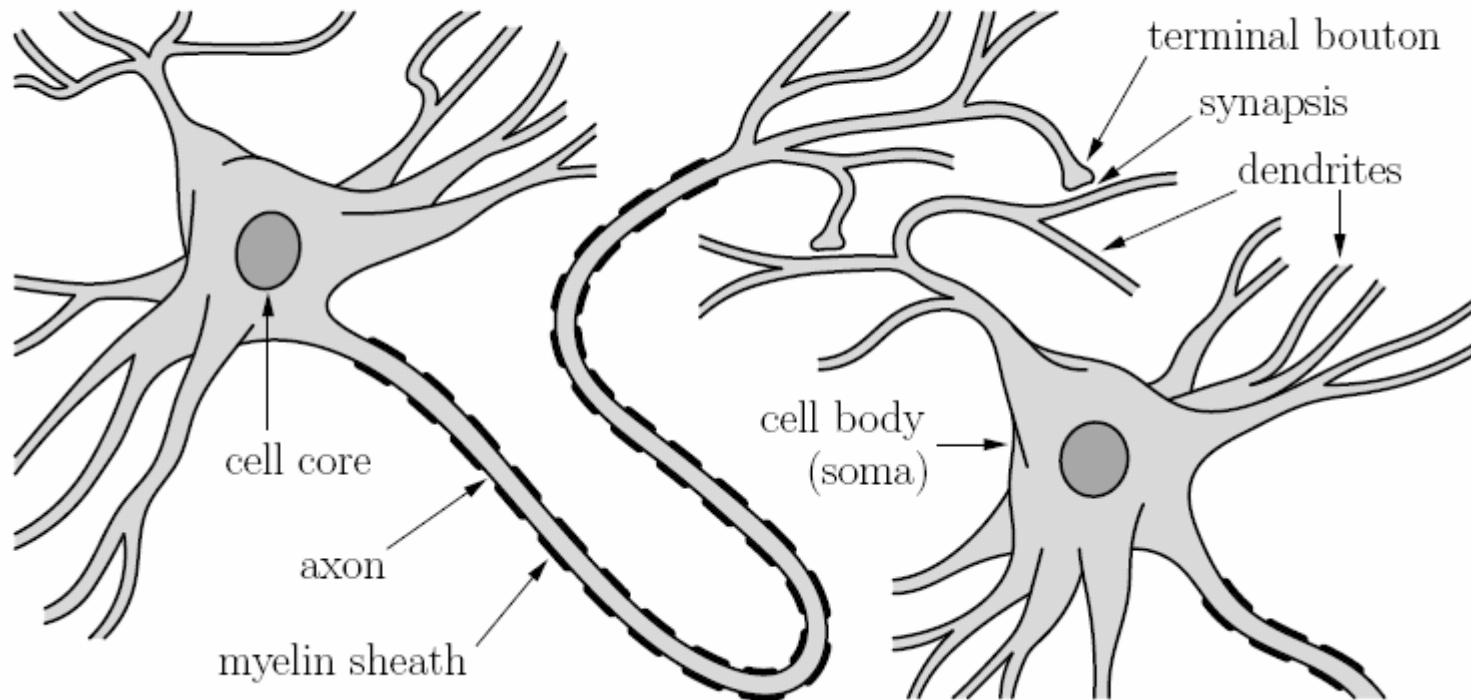
Chapter 1:

Introduction

Artificial Neural Networks

Biological Background

Structure of a prototypical biological neuron



Biological Background

(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 boutons, it triggers the release of neurotransmitters.

Konventionelle Rechner vs. Gehirn

	Computer	Human Brain
Computational units	1 CPU, 10^5 gates	10^{11} neurons
Storage units	10^9 bits RAM, 10^{10} bits disk	10^{11} neurons, 10^{14} synapses
Cycle time	10^{-8} sec.	10^{-3} sec.
Bandwith	10^9 bits/sec	10^{14} bits/sec
Neuro updates/sec.	10^5	10^{14}

Konventionelle Rechner vs. Gehirn

Beachte: **Hirnschaltzeit langsam (10^{-3} s), aber Updates erfolgen parallel. Dagegen braucht die serielle Simulation auf einem Rechner mehrere hundert Zyklen für 1 Update.**

Vorteile natürlicher neuronaler Netze:

- **Hohe Verarbeitungsgeschwindigkeit durch massive Parallelität.**
- **Funktionstüchtig selbst bei Ausfall von Teilen des Netzes (Fehlertoleranz).**
- **Langsamer Funktionsausfall bei fortschreitenden Ausfällen von Neuronen. (Graceful Degradation).**
- **Gut geeignet für induktives Lernen.**

Es erscheint daher sinnvoll, diese Vorteile natürlicher neuronaler Netze künstlich nachzuahmen.

Motivation: Why (Artificial) Neural Networks?

- **(Neuro-)Biology / (Neuro-)Physiology / Psychology:**
 - Exploit similarity to real (biological) neural networks.
 - Build models to understand nerve and brain operation by simulation.
- **Computer Science / Engineering / Economics**
 - Mimic certain cognitive capabilities of human beings.
 - Solve learning/adaptation, prediction, and optimization problems.
- **Physics / Chemistry**
 - Use neural network models to describe physical phenomena.
 - Special case: spin glasses (alloys of magnetic and non-magnetic metals).

Motivation: Why Neural Networks?

Physical-Symbol System Hypothesis [Newell and Simon 1976]

A physical-symbol system has the necessary and sufficient means for general intelligent action.

Neural networks process simple signals, not symbols.

So why study neural networks in Artificial Intelligence?

- Symbol-based representations work well for inference tasks, but fairly bad for perception tasks.
- Symbol-based expert systems tend to get slower with growing knowledge, human experts tend to get faster.
- Neural networks allow for highly parallel information processing.
- There are several successful applications in industry and finance.

Types of Neural Networks

- **Introduction**
Motivation, Biological Background
- **Threshold Logic Units**
Definition, Geometric Interpretation, Limitations, Networks of TLUs, Training
- **General Neural Networks**
Structure, Operation, Training
- **Multilayer Perceptrons**
Definition, Function Approximation, Gradient Descent, Backpropagation, Variants, Sensitivity Analysis
- **Radial Basis Function Networks**
Definition, Function Approximation, Initialization, Training, Generalized Version
- **Self-Organizing Maps**
Definition, Learning Vector Quantization, Neighborhood of Output Neurons
- **Hopfield Networks**
Definition, Convergence, Associative Memory, Solving Optimization Problems
- **Recurrent Neural Networks**
Differential Equations, Vector Networks, Backpropagation through Time

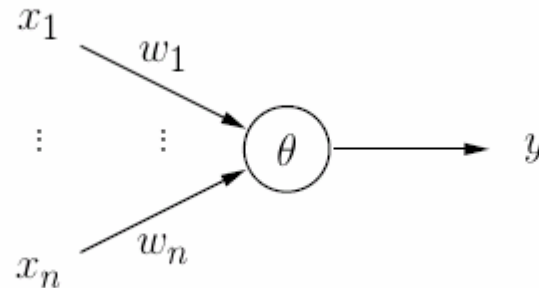
Chapter 2:

Threshold Logic Units (Perceptrons)

Threshold Logic Units

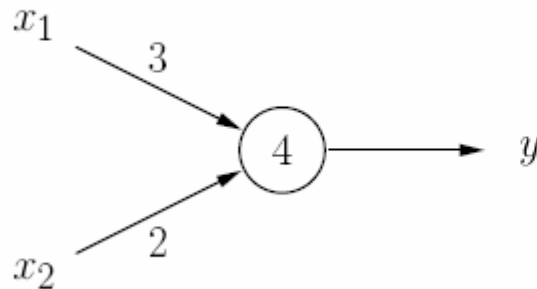
A **Threshold Logic Unit (TLU)** is a processing unit for numbers with n inputs x_1, \dots, x_n and one output y . The unit has a **threshold** θ and each input x_i is associated with a **weight** w_i . A threshold logic unit computes the function

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



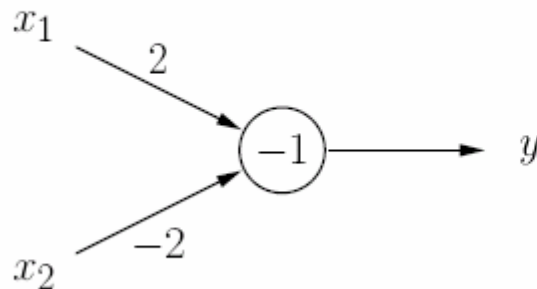
Threshold Logic Units: Examples

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

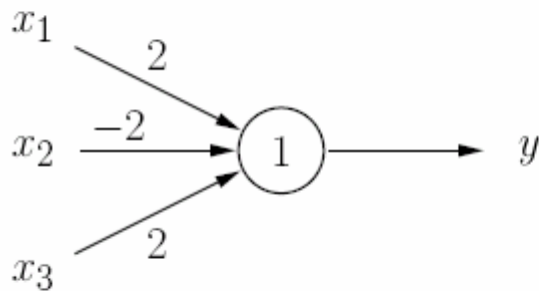
Threshold logic unit for the implication $x_2 \rightarrow x_1$.



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 $(x_1 \wedge \overline{x_2}) \vee (x_1 \wedge x_3) \vee (\overline{x_2} \wedge x_3)$.



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

Review of line representations

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

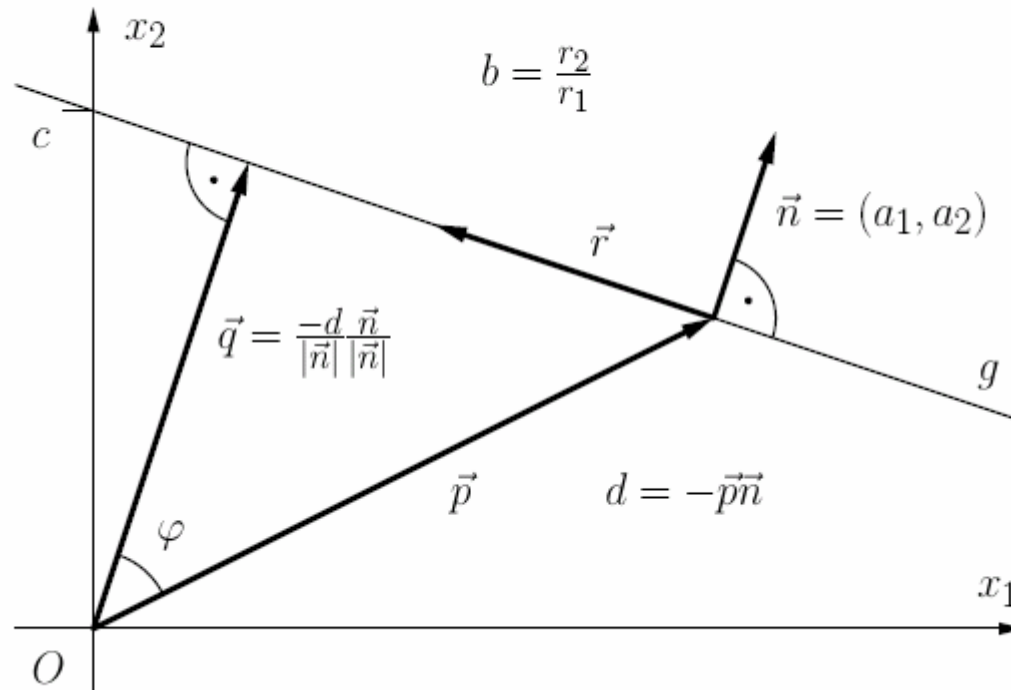
Explicit Form:	$g \equiv x_2 = bx_1 + c$
Implicit Form:	$g \equiv a_1x_1 + a_2x_2 + d = 0$
Point-Direction Form:	$g \equiv \vec{x} = \vec{p} + k\vec{r}$
Normal Form:	$g \equiv (\vec{x} - \vec{p})\vec{n} = 0$

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

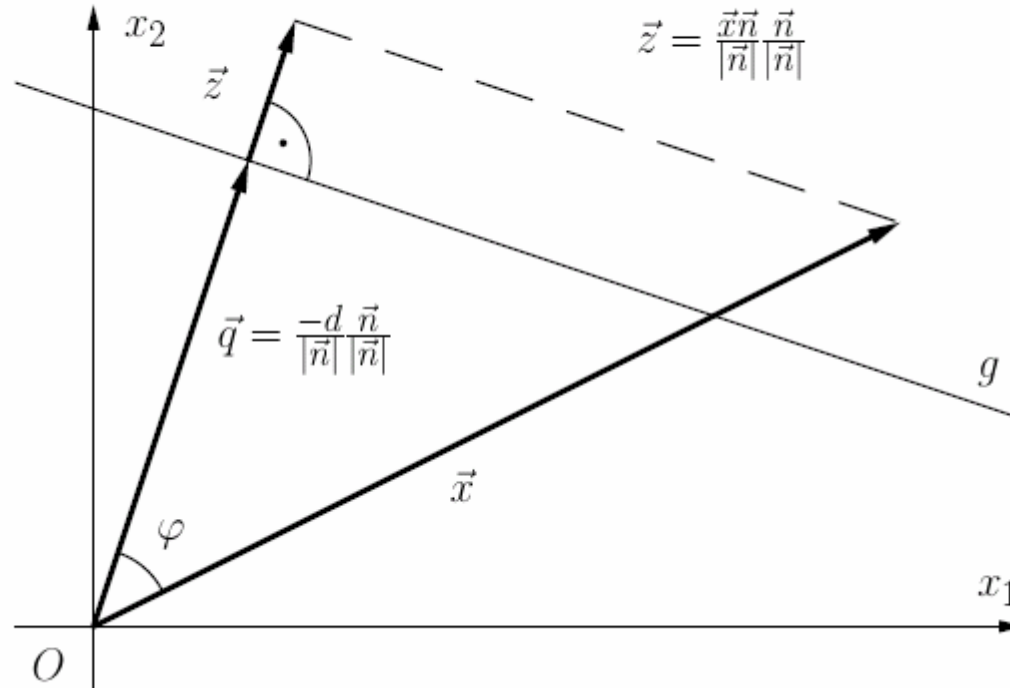
Threshold Logic Units: Geometric Interpretation

A straight line and its defining parameters.



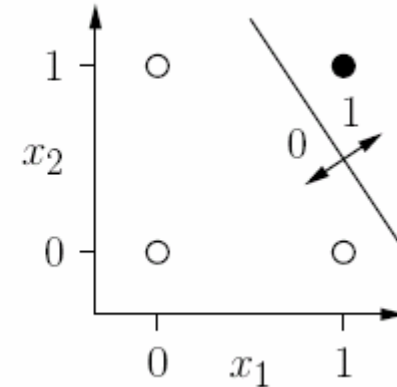
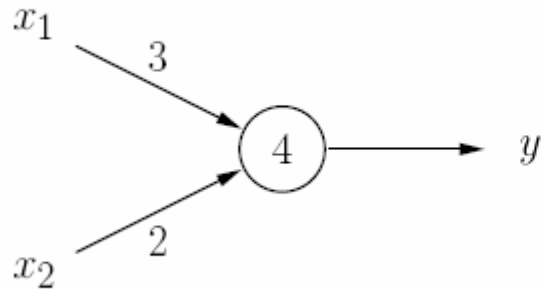
Threshold Logic Units: Geometric Interpretation

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

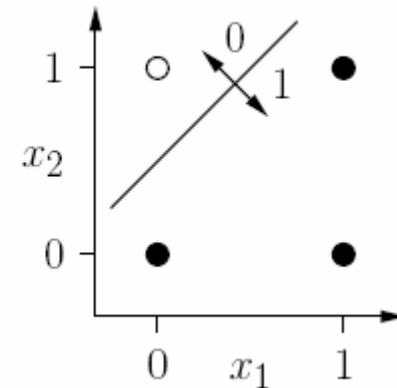
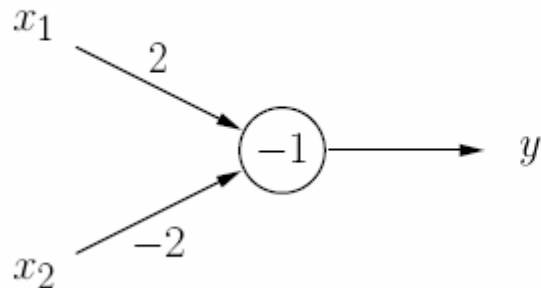


Threshold Logic Units: Geometric Interpretation

Threshold logic unit for $x_1 \wedge x_2$.

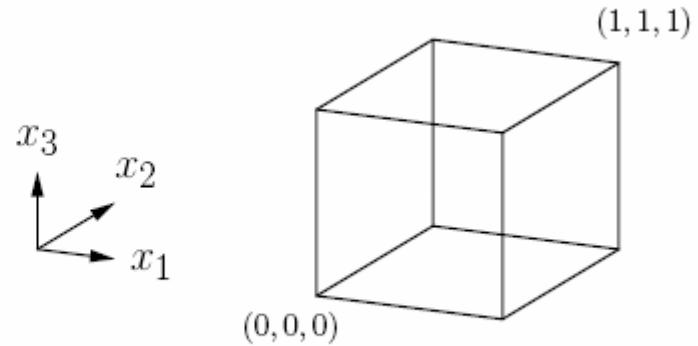


A threshold logic unit for $x_2 \rightarrow x_1$.

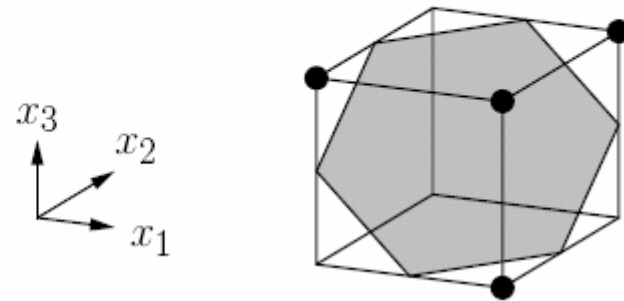
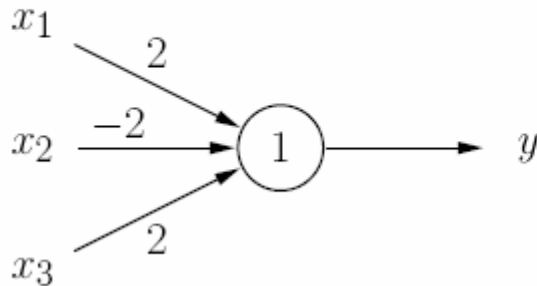


Threshold Logic Units: Geometric Interpretation

Visualization of 3-dimensional Boolean functions:



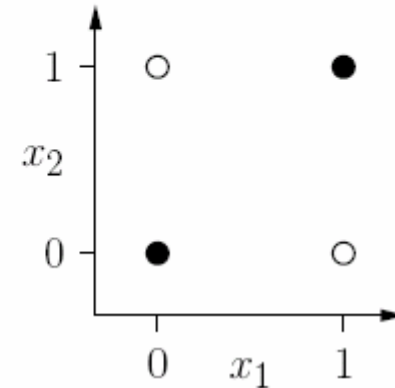
Threshold logic unit for $(x_1 \wedge \overline{x_2}) \vee (x_1 \wedge x_3) \vee (\overline{x_2} \wedge x_3)$.



Threshold Logic Units: Limitations

The bimplication problem $x_1 \leftrightarrow x_2$: There is no separating line.

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



Formal proof by *reductio ad absurdum*:

$$\text{since } (0, 0) \mapsto 1: \quad 0 \geq \theta, \quad (1)$$

$$\text{since } (1, 0) \mapsto 0: \quad w_1 < \theta, \quad (2)$$

$$\text{since } (0, 1) \mapsto 0: \quad w_2 < \theta, \quad (3)$$

$$\text{since } (1, 1) \mapsto 1: \quad w_1 + w_2 \geq \theta. \quad (4)$$

(2) and (3): $w_1 + w_2 < 2\theta$. With (4): $2\theta > \theta$, or $\theta > 0$. Contradiction to (1).

Threshold Logic Units: Limitations

Total number and number of linearly separable Boolean functions.
([Widner 1960] as cited in [Zell 1994])

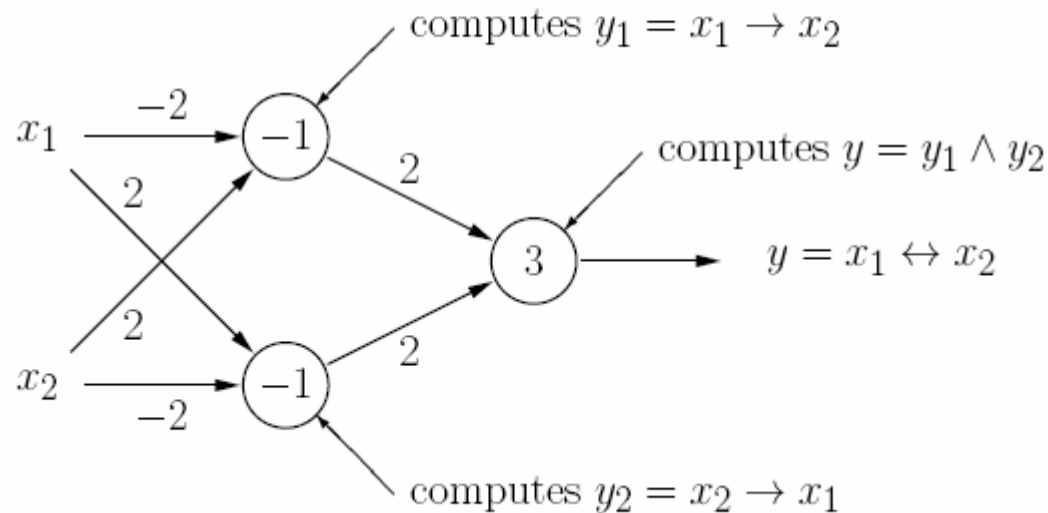
inputs	Boolean functions	linearly separable functions
1	4	4
2	16	14
3	256	104
4	65536	1774
5	$4.3 \cdot 10^9$	94572
6	$1.8 \cdot 10^{19}$	$5.0 \cdot 10^6$

- 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

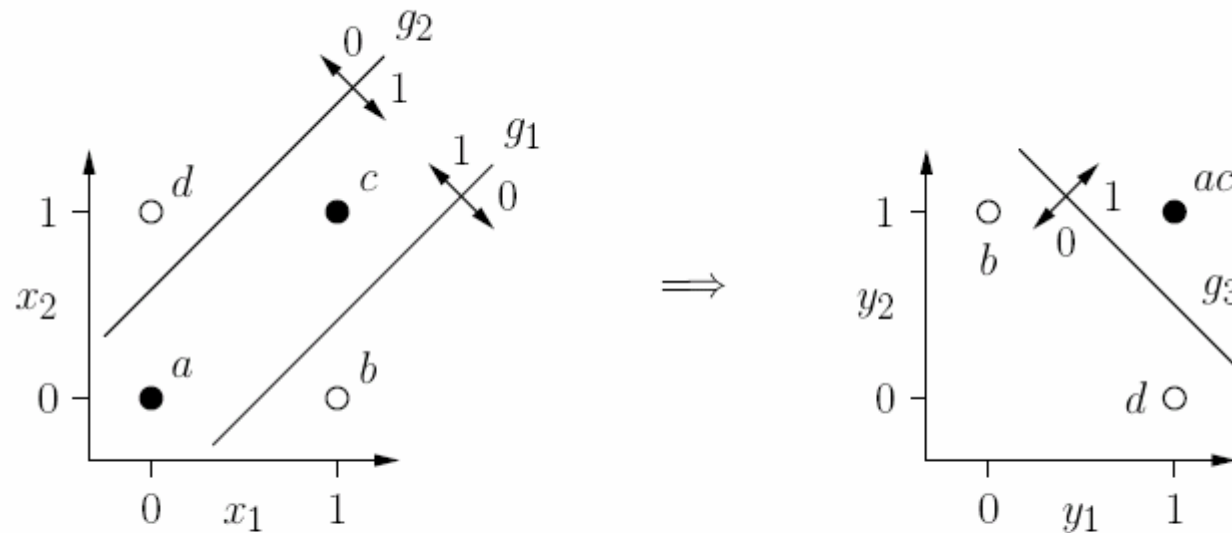
Solving the biimplication problem with a network.

Idea: Logical Decomposition $x_1 \leftrightarrow x_2 \equiv (x_1 \rightarrow x_2) \wedge (x_2 \rightarrow x_1)$



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.

Representing Arbitrary Boolean Functions

Let $y = f(x_1, \dots, x_n)$ be a Boolean function of n variables.

(i) Represent $f(x_1, \dots, x_n)$ in disjunctive normal form. That is, determine $D_f = K_1 \vee \dots \vee K_m$, where all K_j are conjunctions of n literals, i.e., $K_j = l_{j1} \wedge \dots \wedge l_{jn}$ with $l_{ji} = x_i$ (positive literal) or $l_{ji} = \neg x_i$ (negative literal).

(ii) Create a neuron for each conjunction K_j of the disjunctive normal form (having n inputs — one input for each variable), where

$$w_{ji} = \begin{cases} 2, & \text{if } l_{ji} = x_i, \\ -2, & \text{if } l_{ji} = \neg x_i, \end{cases} \quad \text{and} \quad \theta_j = n - 1 + \frac{1}{2} \sum_{i=1}^n w_{ji}.$$

(iii) Create an output neuron (having m inputs — one input for each neuron that was created in step (ii)), where

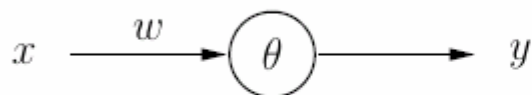
$$w_{(n+1)k} = 2, \quad k = 1, \dots, m, \quad \text{and} \quad \theta_{n+1} = 1.$$

Training Threshold Logic Units

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

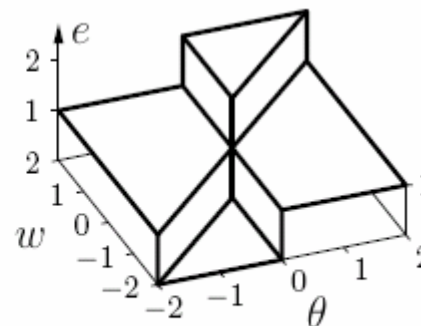
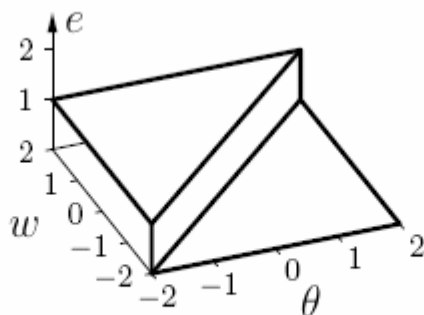
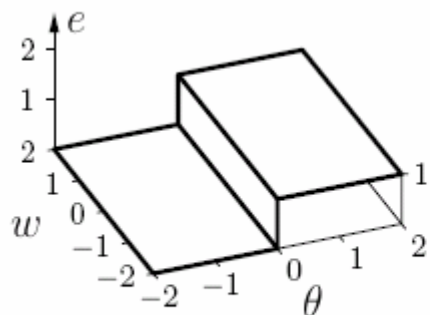
Training Threshold Logic Units

Single input threshold logic unit for the negation $\neg x$.



x	y
0	1
1	0

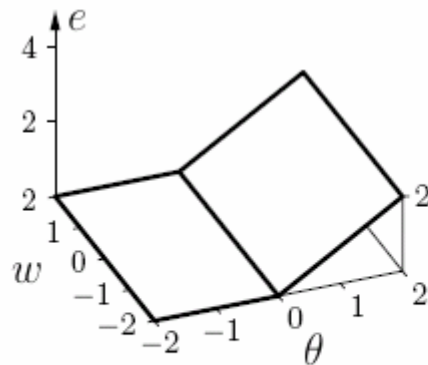
Output error as a function of weight and threshold.



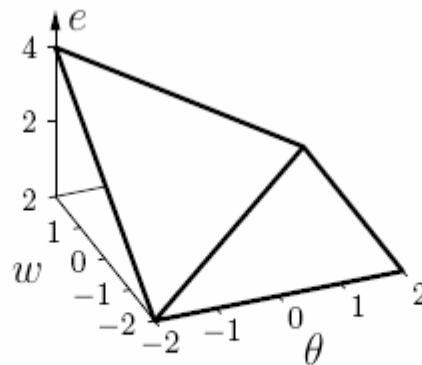
Training Threshold Logic Units

- The error function cannot be used directly, because it consists of plateaus.
- Solution: If the computed output is wrong, take into account, how far the weighted sum is from the threshold.

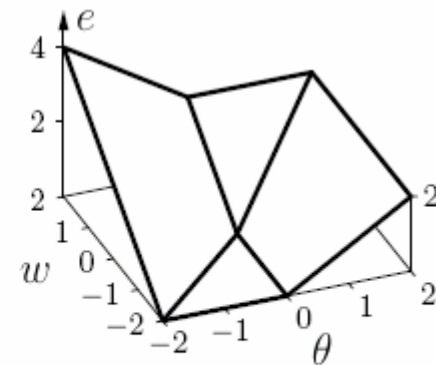
Modified output error as a function of weight and threshold.



error for $x = 0$



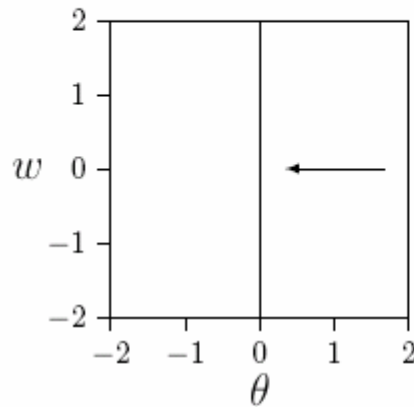
error for $x = 1$



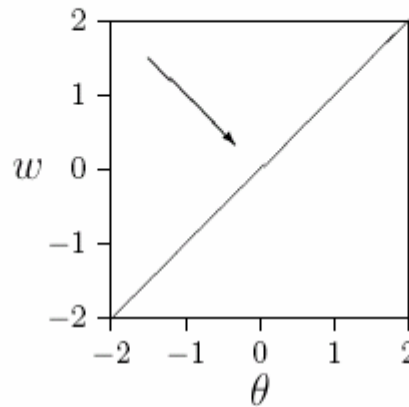
sum of errors

Training Threshold Logic Units

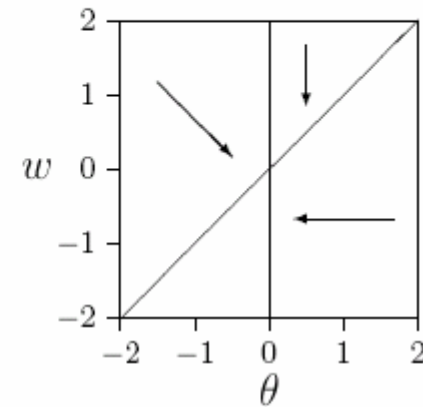
Schemata of resulting directions of parameter changes.



changes for $x = 0$



changes for $x = 1$

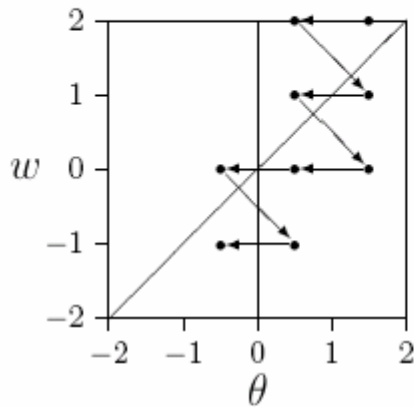


sum of changes

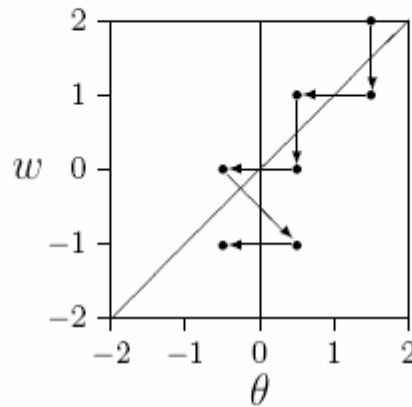
- Start at random point.
- Iteratively adapt parameters according to the direction corresponding to the current point.

Training Threshold Logic Units

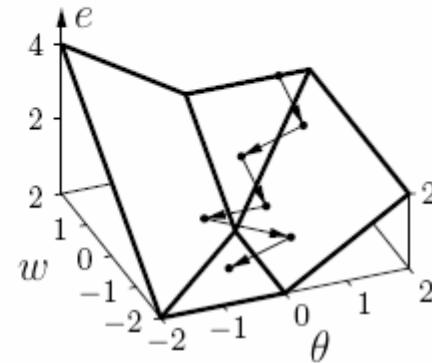
Example training procedure: Online and batch training.



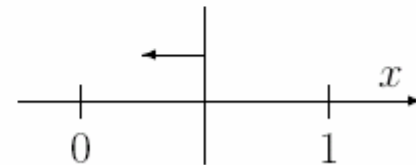
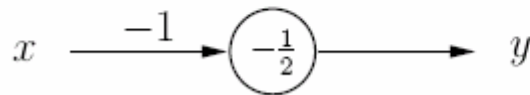
Online-Lernen



Batch-Lernen



Batch-Lernen



Training Threshold Logic Units: Delta Rule

Formal Training Rule: Let $\vec{x} = (x_1, \dots, x_n)$ 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 $\vec{w} = (w_1, \dots, w_n)$ are adapted as follows in order to reduce the error:

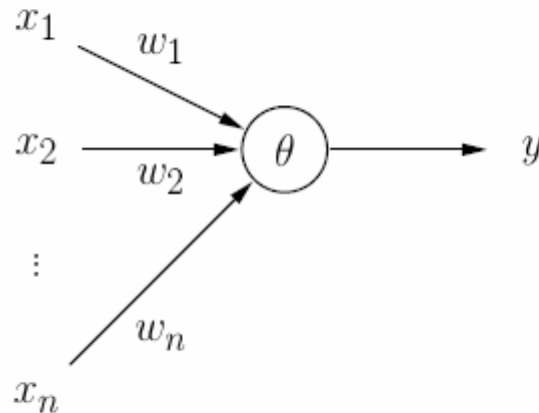
$$\begin{aligned} \theta^{(\text{new})} &= \theta^{(\text{old})} + \Delta\theta & \text{with } \Delta\theta &= -\eta(o - y), \\ \forall i \in \{1, \dots, n\} : w_i^{(\text{new})} &= w_i^{(\text{old})} + \Delta w_i & \text{with } \Delta w_i &= \eta(o - y)x_i, \end{aligned}$$

where η is a parameter that is called **learning rate**. It determines the severity of the weight changes. This procedure is called **Delta Rule** or **Widrow–Hoff Procedure**. [Widrow and Hoff 1960].

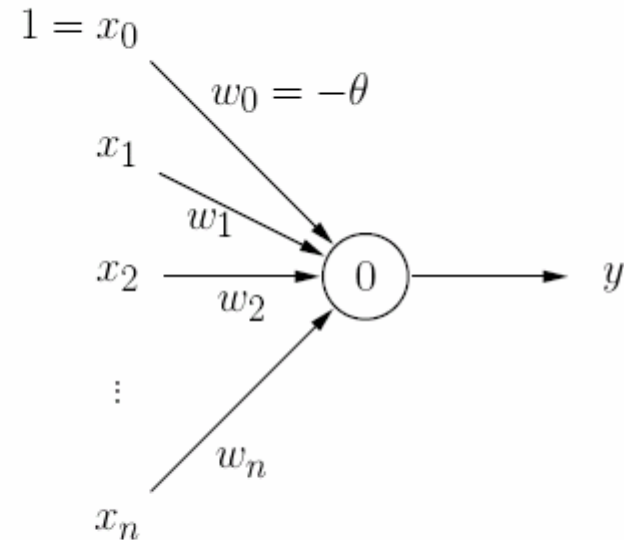
- **Online Training:** Adapt parameters after each training pattern.
- **Batch Training:** Adapt parameters only at the end of each **epoch**, i.e. after a traversal of all training patterns.

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

```
procedure online_training (var  $\vec{w}$ , var  $\theta$ ,  $L$ ,  $\eta$ );
var  $y$ ,  $e$ ;                                (* output, sum of errors *)
begin
  repeat
     $e := 0$ ;                                (* initialize the error sum *)
    for all  $(\vec{x}, o) \in L$  do begin        (* traverse the patterns *)
      if  $(\vec{w}\vec{x} \geq \theta)$  then  $y := 1$ ;  (* compute the output *)
      else  $y := 0$ ;                          (* of the threshold logic unit *)
      if  $(y \neq o)$  then begin              (* if the output is wrong *)
         $\theta := \theta - \eta(o - y)$ ;      (* adapt the threshold *)
         $\vec{w} := \vec{w} + \eta(o - y)\vec{x}$ ;  (* and the weights *)
         $e := e + |o - y|$ ;                (* sum the errors *)
      end;
    end;
  until  $(e \leq 0)$ ;                          (* repeat the computations *)
end;                                         (* until the error vanishes *)
```

Training Threshold Logic Units: Delta Rule

```
procedure batch_training (var  $\vec{w}$ , var  $\theta$ ,  $L$ ,  $\eta$ );  
var  $y$ ,  $e$ , (* output, sum of errors *)  
     $\theta_c$ ,  $\vec{w}_c$ ; (* summed changes *)  
begin  
  repeat  
     $e := 0$ ;  $\theta_c := 0$ ;  $\vec{w}_c := \vec{0}$ ; (* initializations *)  
    for all  $(\vec{x}, o) \in L$  do begin (* traverse the patterns *)  
      if  $(\vec{w}\vec{x} \geq \theta)$  then  $y := 1$ ; (* compute the output *)  
      else  $y := 0$ ; (* of the threshold logic unit *)  
      if  $(y \neq o)$  then begin (* if the output is wrong *)  
         $\theta_c := \theta_c - \eta(o - y)$ ; (* sum the changes of the *)  
         $\vec{w}_c := \vec{w}_c + \eta(o - y)\vec{x}$ ; (* threshold and the weights *)  
         $e := e + |o - y|$ ; (* sum the errors *)  
      end;  
    end;  
     $\theta := \theta + \theta_c$ ; (* adapt the threshold *)  
     $\vec{w} := \vec{w} + \vec{w}_c$ ; (* and the weights *)  
  until  $(e \leq 0)$ ; (* repeat the computations *)  
end; (* until the error vanishes *)
```


Training Threshold Logic Units: Online

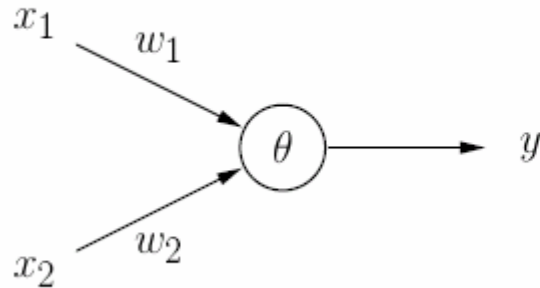
epoch	x	o	$\vec{x}\vec{w}$	y	e	$\Delta\theta$	Δw	θ	w
								1.5	2
1	0	1	-1.5	0	1	-1	0	0.5	2
	1	0	1.5	1	-1	1	-1	1.5	1
2	0	1	-1.5	0	1	-1	0	0.5	1
	1	0	0.5	1	-1	1	-1	1.5	0
3	0	1	-1.5	0	1	-1	0	0.5	0
	1	0	0.5	0	0	0	0	0.5	0
4	0	1	-0.5	0	1	-1	0	-0.5	0
	1	0	0.5	1	-1	1	-1	0.5	-1
5	0	1	-0.5	0	1	-1	0	-0.5	-1
	1	0	-0.5	0	0	0	0	-0.5	-1
6	0	1	0.5	1	0	0	0	-0.5	-1
	1	0	-0.5	0	0	0	0	-0.5	-1

Training Threshold Logic Units: Batch

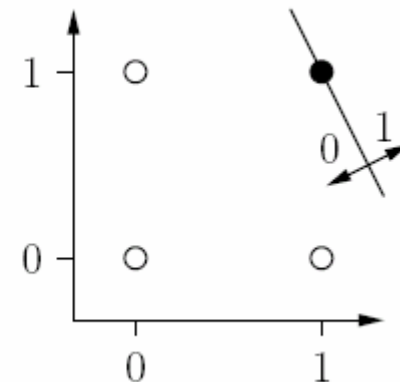
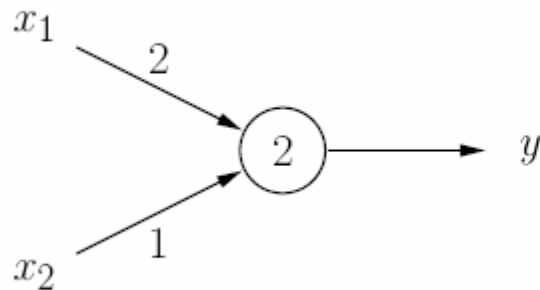
epoch	x	o	$\vec{x}\vec{w}$	y	e	$\Delta\theta$	Δw	θ	w
								1.5	2
1	0	1	-1.5	0	1	-1	0		
	1	0	0.5	1	-1	1	-1	1.5	1
2	0	1	-1.5	0	1	-1	0		
	1	0	-0.5	0	0	0	0	0.5	1
3	0	1	-0.5	0	1	-1	0		
	1	0	0.5	1	-1	1	-1	0.5	0
4	0	1	-0.5	0	1	-1	0		
	1	0	-0.5	0	0	0	0	-0.5	0
5	0	1	0.5	1	0	0	0		
	1	0	0.5	1	-1	1	-1	0.5	-1
6	0	1	-0.5	0	1	-1	0		
	1	0	-1.5	0	0	0	0	-0.5	-1
7	0	1	0.5	1	0	0	0		
	1	0	-0.5	0	0	0	0	-0.5	-1

Training Threshold Logic Units: Conjunction

Threshold logic unit with two inputs for the conjunction.



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



Training Threshold Logic Units: Conjunction

epoch	x_1	x_2	o	$\vec{x}\vec{w}$	y	e	$\Delta\theta$	Δw_1	Δw_2	θ	w_1	w_2
										0	0	0
1	0	0	0	0	1	-1	1	0	0	1	0	0
	0	1	0	-1	0	0	0	0	0	1	0	0
	1	0	0	-1	0	0	0	0	0	1	0	0
	1	1	1	-1	0	1	-1	1	1	0	1	1
2	0	0	0	0	1	-1	1	0	0	1	1	1
	0	1	0	0	1	-1	1	0	-1	2	1	0
	1	0	0	-1	0	0	0	0	0	2	1	0
	1	1	1	-1	0	1	-1	1	1	1	2	1
3	0	0	0	-1	0	0	0	0	0	1	2	1
	0	1	0	0	1	-1	1	0	-1	2	2	0
	1	0	0	0	1	-1	1	-1	0	3	1	0
	1	1	1	-2	0	1	-1	1	1	2	2	1
4	0	0	0	-2	0	0	0	0	0	2	2	1
	0	1	0	-1	0	0	0	0	0	2	2	1
	1	0	0	0	1	-1	1	-1	0	3	1	1
	1	1	1	-1	0	1	-1	1	1	2	2	2
5	0	0	0	-2	0	0	0	0	0	2	2	2
	0	1	0	0	1	-1	1	0	-1	3	2	1
	1	0	0	-1	0	0	0	0	0	3	2	1
	1	1	1	0	1	0	0	0	0	3	2	1
6	0	0	0	-3	0	0	0	0	0	3	2	1
	0	1	0	-2	0	0	0	0	0	3	2	1
	1	0	0	-1	0	0	0	0	0	3	2	1
	1	1	1	0	1	0	0	0	0	3	2	1

Training Threshold Logic Units: Biimplication

epoch	x_1	x_2	o	$\vec{x}\vec{w}$	y	e	$\Delta\theta$	Δw_1	Δw_2	θ	w_1	w_2
										0	0	0
1	0	0	1	0	1	0	0	0	0	0	0	0
	0	1	0	0	1	-1	1	0	-1	1	0	-1
	1	0	0	-1	0	0	0	0	0	1	0	-1
	1	1	1	-2	0	1	-1	1	1	0	1	0
2	0	0	1	0	1	0	0	0	0	0	1	0
	0	1	0	0	1	-1	1	0	-1	1	1	-1
	1	0	0	0	1	-1	1	-1	0	2	0	-1
	1	1	1	-3	0	1	-1	1	1	1	1	0
3	0	0	1	0	1	0	0	0	0	0	1	0
	0	1	0	0	1	-1	1	0	-1	1	1	-1
	1	0	0	0	1	-1	1	-1	0	2	0	-1
	1	1	1	-3	0	1	-1	1	1	1	1	0

Training Threshold Logic Units: Convergence

Convergence Theorem: Let $L = \{(\vec{x}_1, o_1), \dots, (\vec{x}_m, o_m)\}$ be a set of training patterns, each consisting of an input vector $\vec{x}_i \in \mathbb{R}^n$ and a desired output $o_i \in \{0, 1\}$. Furthermore, let $L_0 = \{(\vec{x}, o) \in L \mid o = 0\}$ and $L_1 = \{(\vec{x}, o) \in L \mid o = 1\}$. If L_0 and L_1 are linearly separable, i.e., if $\vec{w} \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$ exist, such that

$$\begin{aligned} \forall (\vec{x}, 0) \in L_0 : \quad \vec{w}\vec{x} < \theta & \quad \text{and} \\ \forall (\vec{x}, 1) \in L_1 : \quad \vec{w}\vec{x} \geq \theta, \end{aligned}$$

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 Networks of Threshold Logic Units

- 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, 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”.