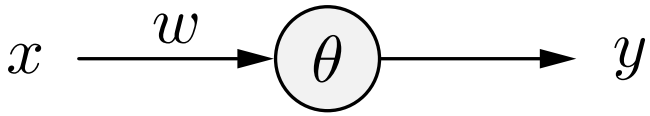
Training Threshold Logic Units

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

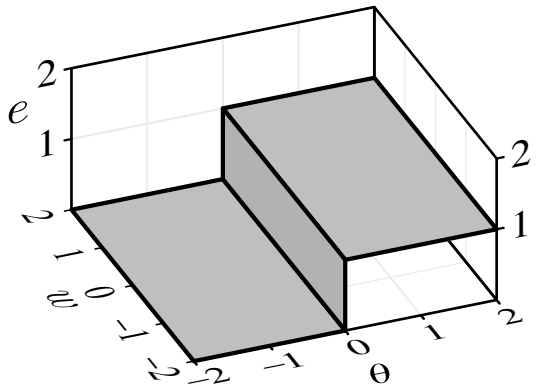
Training Threshold Logic Units

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

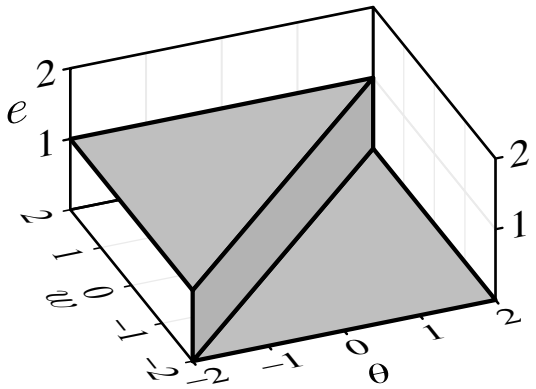


x	y
0	1
1	0

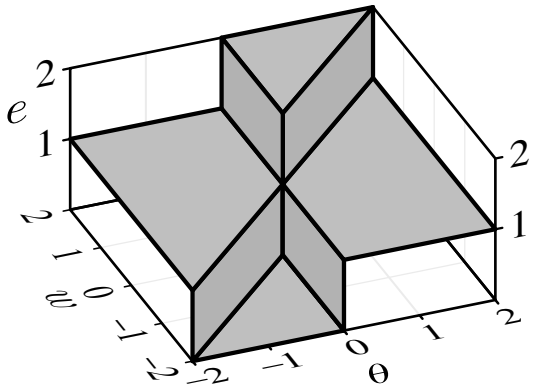
Output error as a function of weight and threshold.



error for $x = 0$



error for $x = 1$

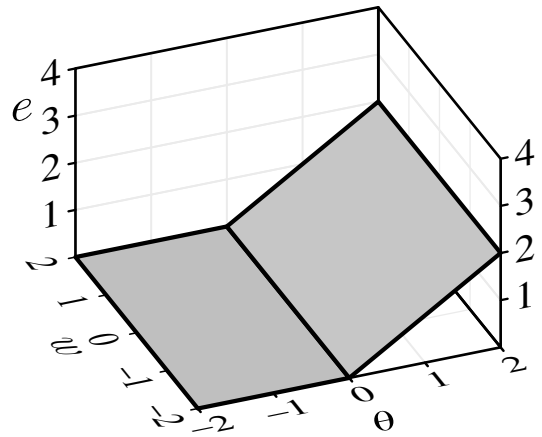


sum of errors

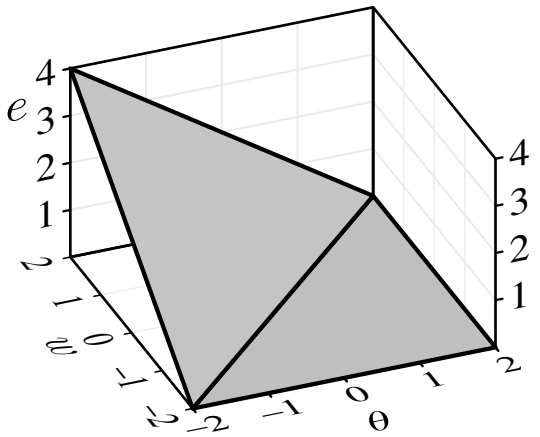
Training Threshold Logic Units

- 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
  (that is, consider "how wrong" the relation of weighted sum and threshold is).

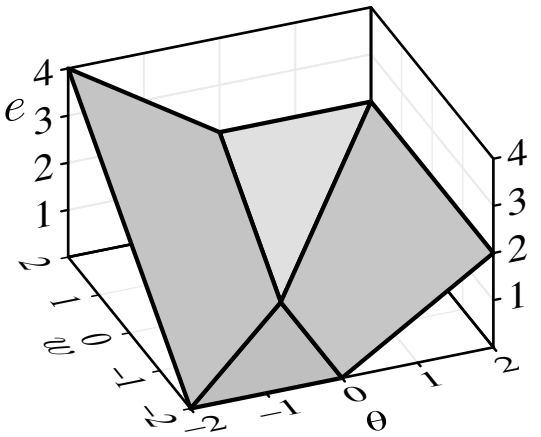
Modified output error as a function of weight and threshold.



error for $x = 0$



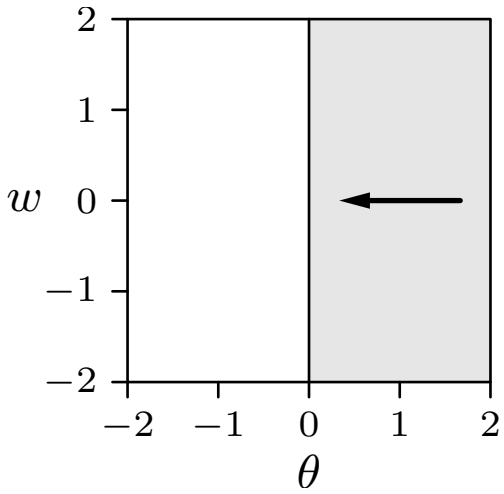
error for $x = 1$



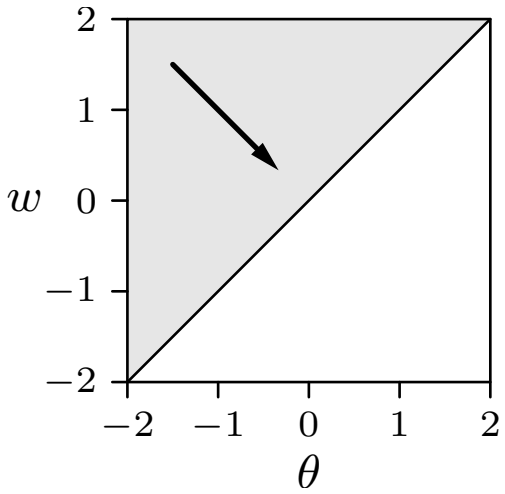
sum of errors

Training Threshold Logic Units

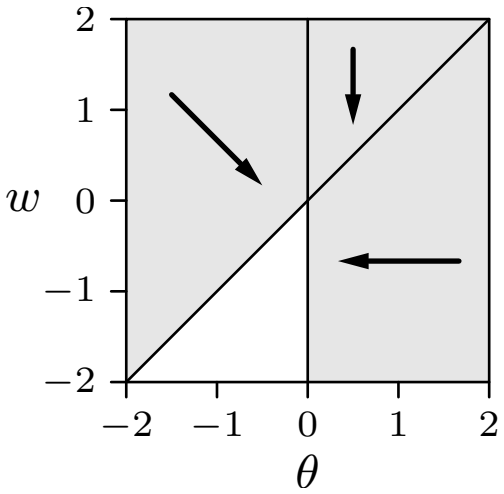
Schemata of resulting directions of parameter changes.



changes for $x = 0$



changes for $x = 1$



sum of changes

- Start at a random point.
- Iteratively adapt parameters
  according to the direction corresponding to the current point.
- Stop if the error vanishes.

Training Threshold Logic Units: Delta Rule

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

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

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**, that is, after a traversal of all training patterns.

Training Threshold Logic Units: Delta Rule

```
procedure online_training (var w⃗, var θ, L, η);
var y, e;                                  (* output, sum of errors *)
begin
  repeat                                   (* training loop *)
    e := 0;                                (* initialize the error sum *)
    for all (x⃗, o) ∈ L do begin            (* traverse the patterns *)
      if (w⃗^⊤ x⃗ ≥ 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 *)
        w⃗ := w⃗ + η(o − y)x⃗;               (* 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

```
procedure batch_training (var w⃗, var θ, L, η);
var y, e, θ_c, w⃗_c;                          (* output, sum of errors, sums of changes *)
begin
  repeat                                     (* training loop *)
    e := 0; θ_c := 0; w⃗_c := 0⃗;              (* initializations *)
    for all (x⃗, o) ∈ L do begin              (* traverse the patterns *)
      if (w⃗^⊤ x⃗ ≥ 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_c  := θ_c  − η(o − y);              (* sum the changes of the *)
        w⃗_c := w⃗_c + η(o − y)x⃗;             (* threshold and the weights *)
        e    := e    +  |o − y|;             (* sum the errors *)
      end;
    end;
    θ  := θ  + θ_c;                          (* adapt the threshold *)
    w⃗ := w⃗ + w⃗_c;                          (* and the weights *)
  until (e ≤ 0);                             (* repeat the computations *)
end;                                         (* until the error vanishes *)
```

Training Threshold Logic Units: Online

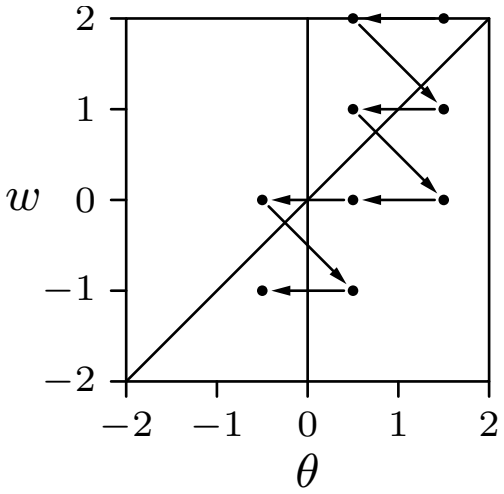
epoch	x	o	$\vec{x}\vec{w}$	y	e	$\Delta\theta$	Δw	θ	w
								1.5	2
1	0	1	−1.5	0	1	−1	0	0.5	2
	1	0	1.5	1	−1	1	−1	1.5	1
2	0	1	−1.5	0	1	−1	0	0.5	1
	1	0	0.5	1	−1	1	−1	1.5	0
3	0	1	−1.5	0	1	−1	0	0.5	0
	1	0	0.5	0	0	0	0	0.5	0
4	0	1	−0.5	0	1	−1	0	−0.5	0
	1	0	0.5	1	−1	1	−1	0.5	−1
5	0	1	−0.5	0	1	−1	0	−0.5	−1
	1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

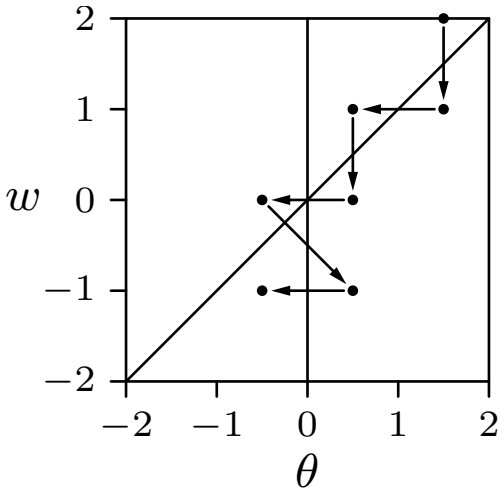
epoch	x	o	$\vec{x}\vec{w}$	y	e	$\Delta\theta$	Δw	θ	w
								1.5	2
1	0	1	−1.5	0	1	−1	0		
	1	0	0.5	1	−1	1	−1	1.5	1
2	0	1	−1.5	0	1	−1	0		
	1	0	−0.5	0	0	0	0	0.5	1
3	0	1	−0.5	0	1	−1	0		
	1	0	0.5	1	−1	1	−1	0.5	0
4	0	1	−0.5	0	1	−1	0		
	1	0	−0.5	0	0	0	0	−0.5	0
5	0	1	0.5	1	0	0	0		
	1	0	0.5	1	−1	1	−1	0.5	−1
6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

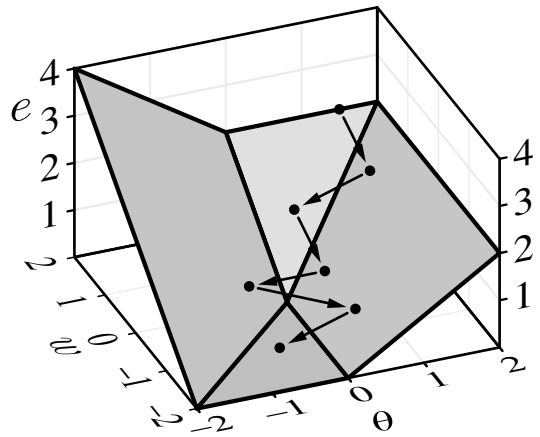
Example training procedure: Online and batch training.



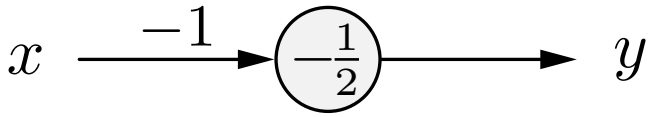
Online Training

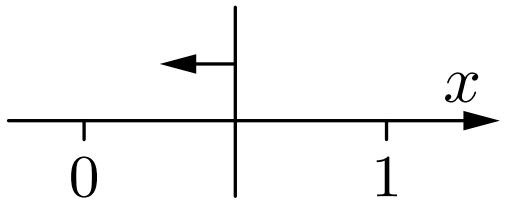


Batch Training



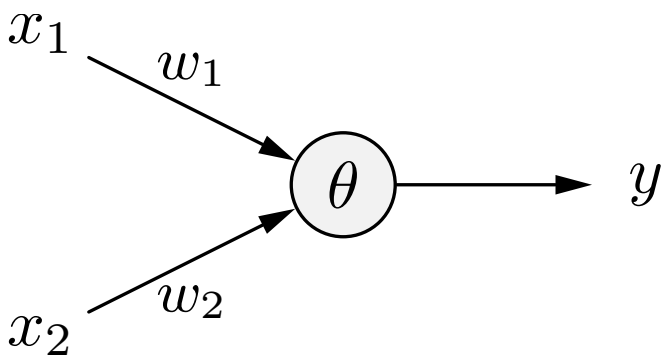
Batch Training



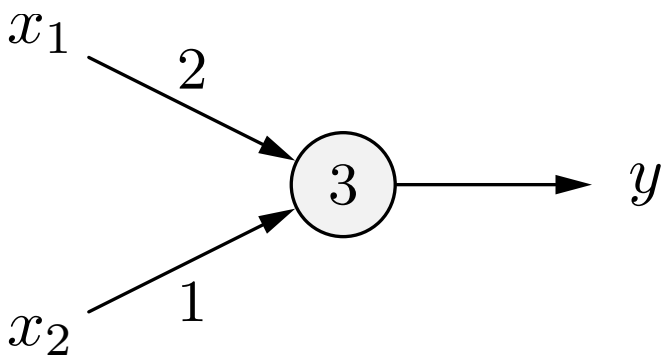


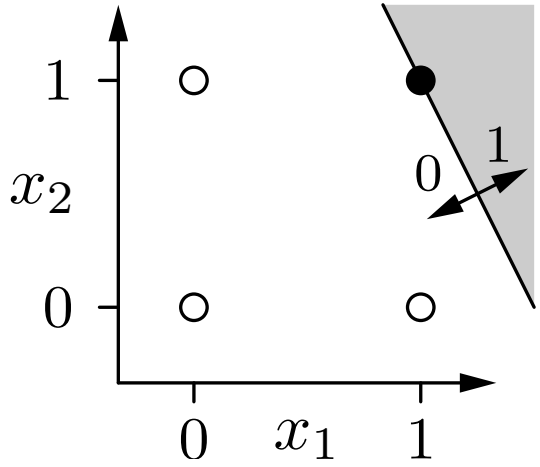
Training Threshold Logic Units: Conjunction

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 $L = \{(\vec{x}_1, o_1), \ldots (\vec{x}_m, o_m)\}$ 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, that is, if $\vec{w} \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$ exist such that

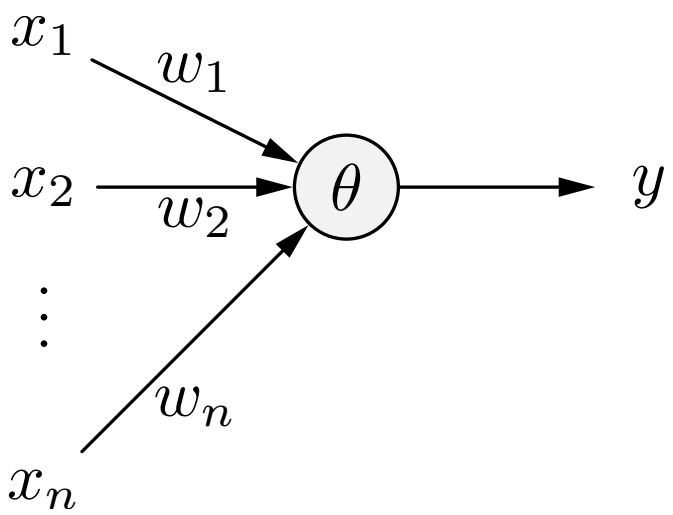
$$\begin{aligned} \forall (\vec{x}, 0) \in L_0 : \quad & \vec{w}^\top \vec{x} < \theta \qquad \text{and} \\ \forall (\vec{x}, 1) \in L_1 : \quad & \vec{w}^\top \vec{x} \geq \theta, \end{aligned}$$

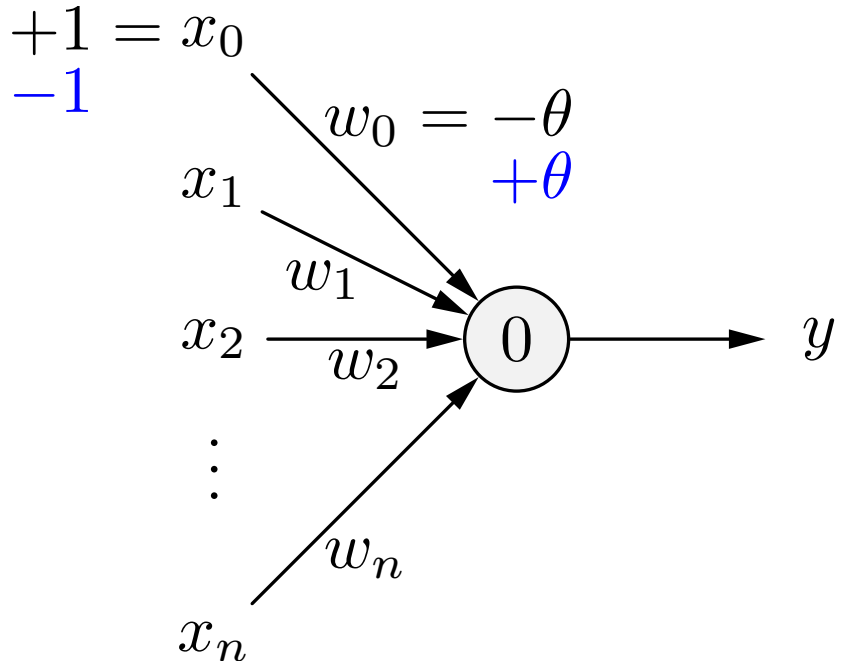
then online as well as batch training terminate.

- 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 (oscillation, repeated computation of same non-solving $\vec{w}$ and θ).

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

Formal Training Rule (with threshold turned into a weight):
Let $\vec{x} = (x_0 = 1, x_1, \ldots, x_n)^\top$ be an (extended) 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 (extended) weight vector $\vec{w} = (w_0 = -\theta, w_1, \ldots, w_n)^\top$ is adapted as follows in order to reduce the error:

$$\forall i \in \{0, \ldots, n\}: \quad w_i^{(\text{new})} = w_i^{(\text{old})} + \Delta w_i \qquad \text{with} \qquad \Delta w_i = \eta(o - y)x_i,$$

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].

- Note that with extended input and weight vectors, there is only one update rule (no distinction of threshold and weights).
- Note also that the (extended) input vector may be $\vec{x} = (x_0 = -1, x_1, \ldots, x_n)^\top$ and the corresponding (extended) weight vector $\vec{w} = (w_0 = +\theta, w_1, \ldots, w_n)^\top$.

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 easy and 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(s),
  - the problem can usually be solved with several different functions
    computed by the neurons of the first layer(s) (non-unique solution).
- When this situation became clear,
  neural networks were first seen as a "research dead end".