

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.

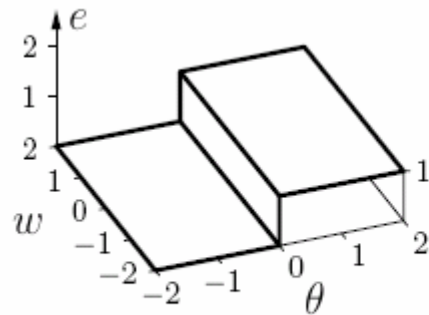
Training Threshold Logic Units

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

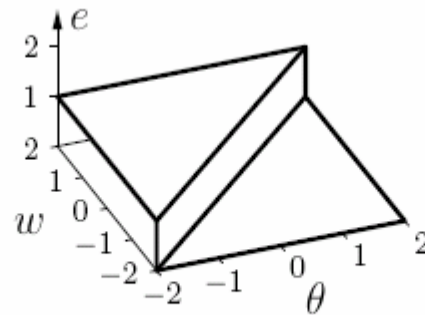


x	y
0	1
1	0

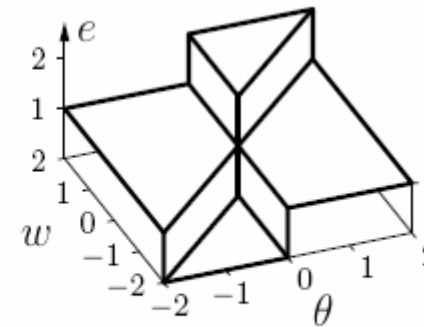
Output error as a function of weight and threshold.



error for $x = 0$



error for $x = 1$

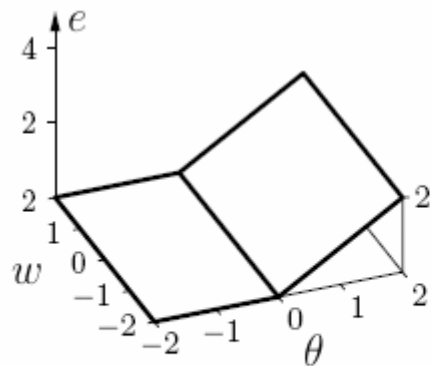


sum of errors

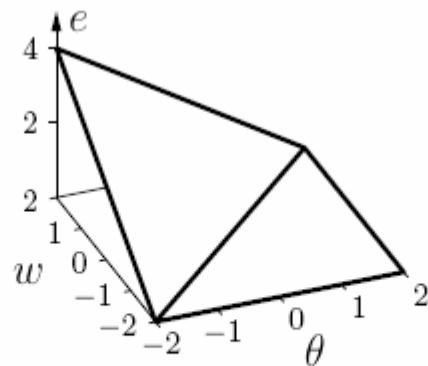
Training Threshold Logic Units

- 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.

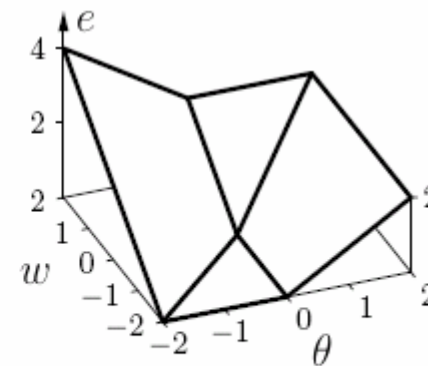
Modified output error as a function of weight and threshold.



error for $x = 0$



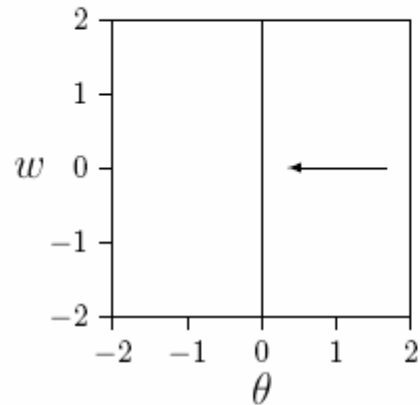
error for $x = 1$



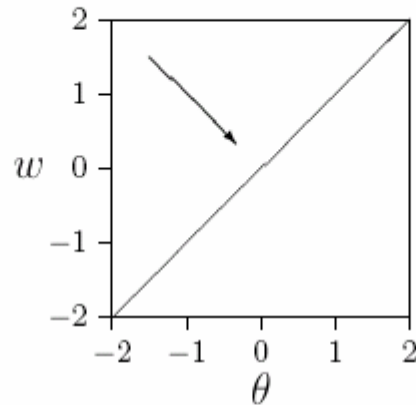
sum of errors

Training Threshold Logic Units

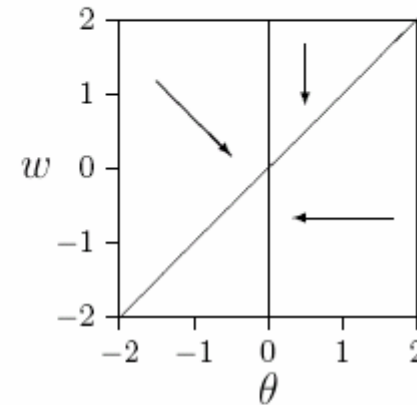
Schemata of resulting directions of parameter changes.



changes for $x = 0$



changes for $x = 1$

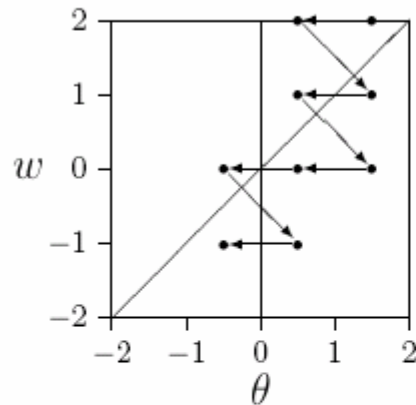


sum of changes

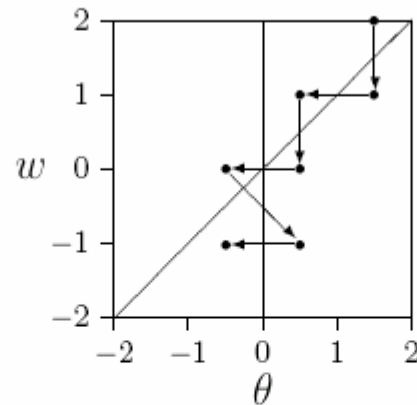
- Start at random point.
- Iteratively adapt parameters according to the direction corresponding to the current point.

Training Threshold Logic Units

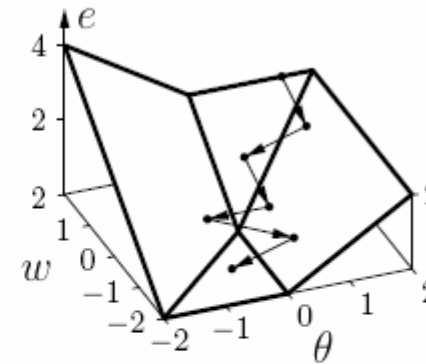
Example training procedure: Online and batch training.



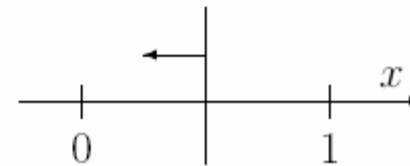
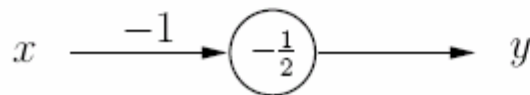
Online-Lernen



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:

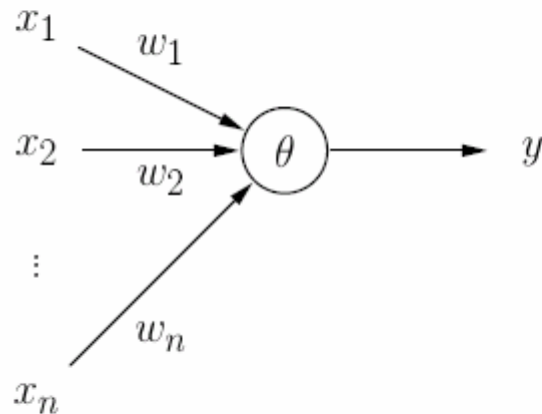
$$\begin{aligned} \theta^{(\text{new})} &= \theta^{(\text{old})} + \Delta\theta \quad \text{with} \quad \Delta\theta = -\eta(o - y), \\ \forall i \in \{1, \dots, n\} : w_i^{(\text{new})} &= w_i^{(\text{old})} + \Delta w_i \quad \text{with} \quad \Delta w_i = \eta(o - y)x_i, \end{aligned}$$

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].

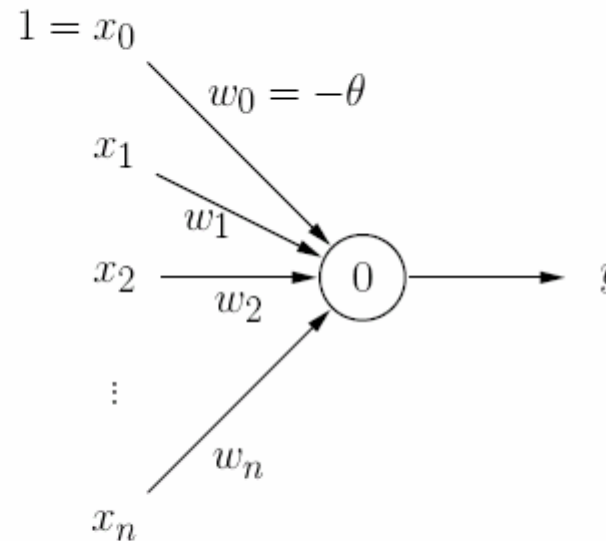
- **Online Training:** Adapt parameters after each training pattern.
- **Batch Training:** Adapt parameters only at the end of each **epoch**, i.e. after a traversal of all training patterns.

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

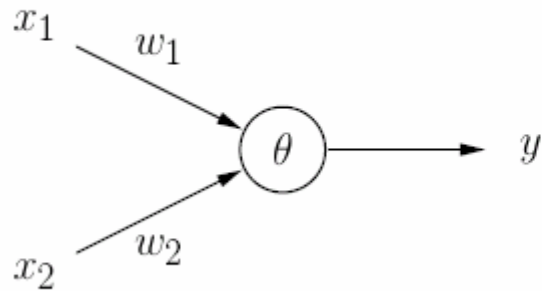
epoch	x	o	$\vec{x}\vec{w}$	y	e	$\Delta\theta$	Δw	θ	w
								1.5	2
1	0	1	-1.5	0	1	-1	0	0.5	2
	1	0	1.5	1	-1	1	-1	1.5	1
2	0	1	-1.5	0	1	-1	0	0.5	1
	1	0	0.5	1	-1	1	-1	1.5	0
3	0	1	-1.5	0	1	-1	0	0.5	0
	1	0	0.5	0	0	0	0	0.5	0
4	0	1	-0.5	0	1	-1	0	-0.5	0
	1	0	0.5	1	-1	1	-1	0.5	-1
5	0	1	-0.5	0	1	-1	0	-0.5	-1
	1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

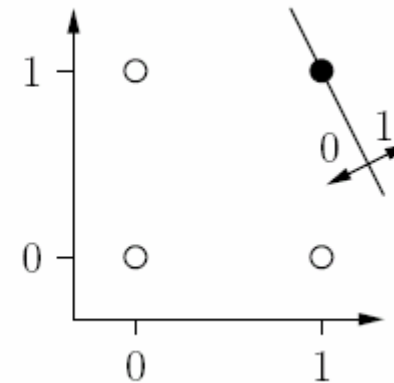
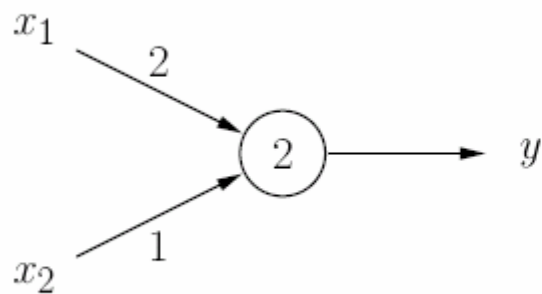
epoch	x	o	$\vec{x}\vec{w}$	y	e	$\Delta\theta$	Δw	θ	w
								1.5	2
1	0	1	-1.5	0	1	-1	0		
	1	0	0.5	1	-1	1	-1	1.5	1
2	0	1	-1.5	0	1	-1	0		
	1	0	-0.5	0	0	0	0	0.5	1
3	0	1	-0.5	0	1	-1	0		
	1	0	0.5	1	-1	1	-1	0.5	0
4	0	1	-0.5	0	1	-1	0		
	1	0	-0.5	0	0	0	0	-0.5	0
5	0	1	0.5	1	0	0	0		
	1	0	0.5	1	-1	1	-1	0.5	-1
6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

Chapter 3:

General Neural Networks

General Neural Networks

Basic graph theoretic notions

A (directed) **graph** is a pair $G = (V, E)$ consisting of a (finite) set V of **nodes** or **vertices** and a (finite) set $E \subseteq V \times V$ of **edges**.

We call an edge $e = (u, v) \in E$ **directed** from node u to node v .

Let $G = (V, E)$ be a (directed) graph and $u \in V$ a node. Then the nodes of the set

$$\text{pred}(u) = \{v \in V \mid (v, u) \in E\}$$

are called the **predecessors** of the node u
and the nodes of the set

$$\text{succ}(u) = \{v \in V \mid (u, v) \in E\}$$

are called the **successors** of the node u .

General Neural Networks

General definition of a neural network

An (artificial) **neural network** is a (directed) graph $G = (U, C)$, whose nodes $u \in U$ are called **neurons** or **units** and whose edges $c \in C$ are called **connections**.

The set U of nodes is partitioned into

- the set U_{in} of **input neurons**,
- the set U_{out} of **output neurons**, and
- the set U_{hidden} of **hidden neurons**.

It is

$$U = U_{\text{in}} \cup U_{\text{out}} \cup U_{\text{hidden}},$$

$$U_{\text{in}} \neq \emptyset, \quad U_{\text{out}} \neq \emptyset, \quad U_{\text{hidden}} \cap (U_{\text{in}} \cup U_{\text{out}}) = \emptyset.$$

General Neural Networks

Each connection $(v, u) \in C$ possesses a **weight** w_{uv} and each neuron $u \in U$ possesses three (real-valued) state variables:

- the **network input** net_u ,
- the **activation** act_u , and
- the **output** out_u .

Each input neuron $u \in U_{\text{in}}$ also possesses a fourth (real-valued) state variable,

- the **external input** ex_u .

Furthermore, each neuron $u \in U$ possesses three functions:

- the **network input function** $f_{\text{net}}^{(u)} : \mathbb{R}^{2|\text{pred}(u)|+\kappa_1(u)} \rightarrow \mathbb{R}$,
- the **activation function** $f_{\text{act}}^{(u)} : \mathbb{R}^{\kappa_2(u)} \rightarrow \mathbb{R}$, and
- the **output function** $f_{\text{out}}^{(u)} : \mathbb{R} \rightarrow \mathbb{R}$,

which are used to compute the values of the state variables.

General Neural Networks

Types of (artificial) neural networks

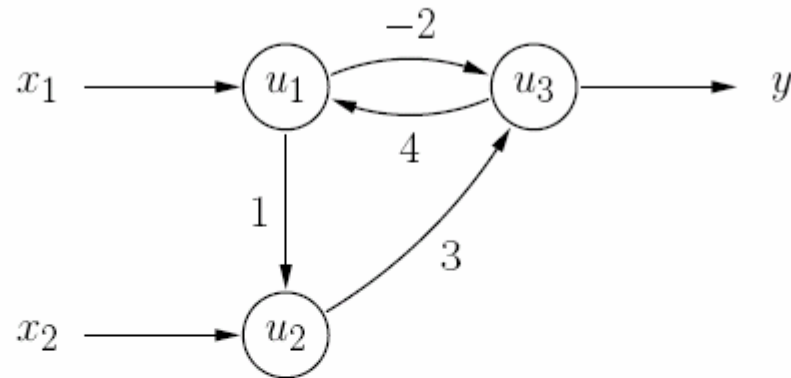
- If the graph of a neural network is **acyclic**, it is called a **feed-forward network**.
- If the graph of a neural network contains **cycles** (backward connections), it is called a **recurrent network**.

Representation of the connection weights by a matrix

$$\begin{array}{cccc} & u_1 & u_2 & \dots & u_r \\ \left(\begin{array}{cccc} w_{u_1 u_1} & w_{u_1 u_2} & \dots & w_{u_1 u_r} \\ w_{u_2 u_1} & w_{u_2 u_2} & & w_{u_2 u_r} \\ \vdots & & & \vdots \\ w_{u_r u_1} & w_{u_r u_2} & \dots & w_{u_r u_r} \end{array} \right) & \begin{array}{c} u_1 \\ u_2 \\ \vdots \\ u_r \end{array} \end{array}$$

General Neural Networks: Example

A simple recurrent neural network



Weight matrix of this network

$$\begin{matrix} & u_1 & u_2 & u_3 \\ \begin{pmatrix} 0 & 0 & 4 \\ 1 & 0 & 0 \\ -2 & 3 & 0 \end{pmatrix} & u_1 \\ & u_2 \\ & u_3 \end{matrix}$$