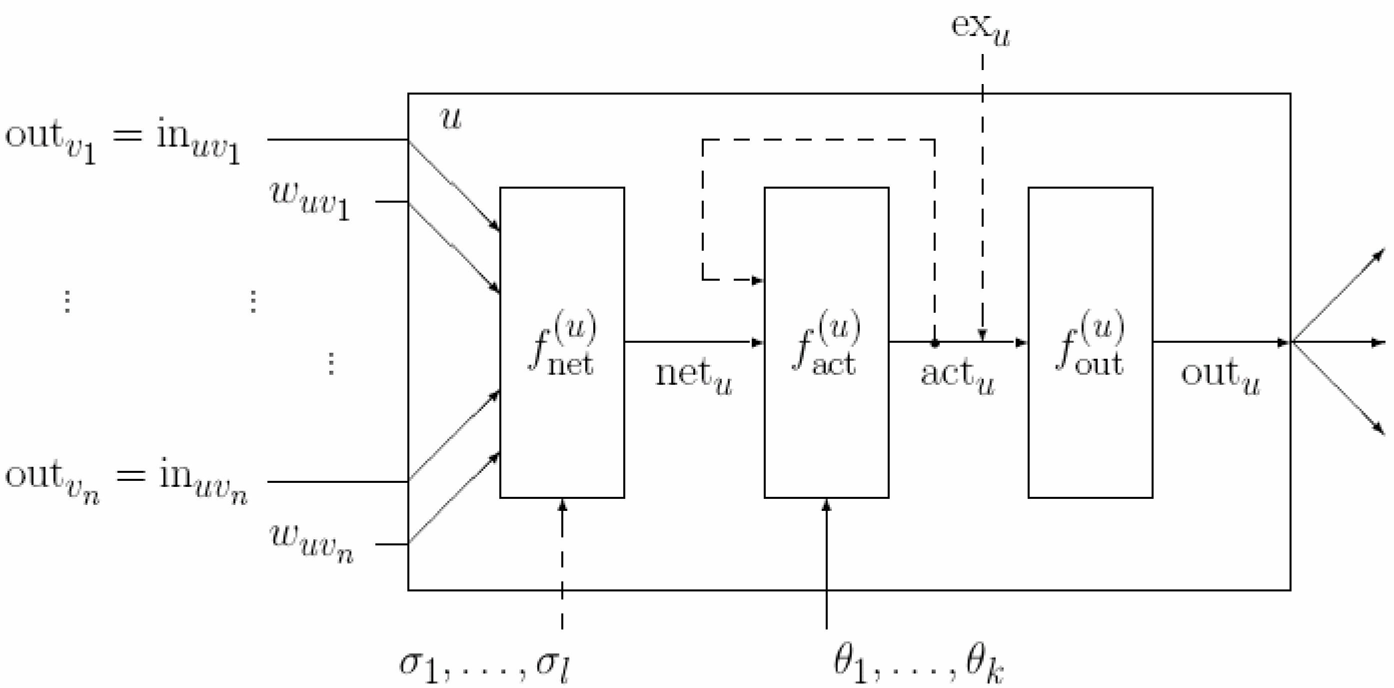
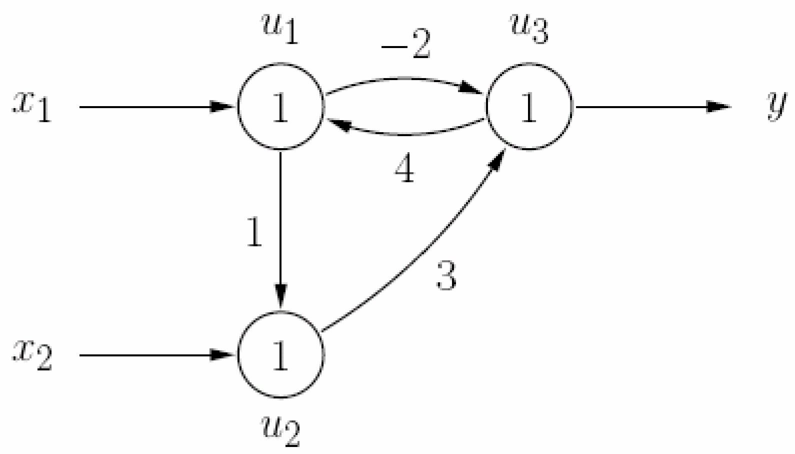
Structure of a Generalized Neuron

A generalized neuron is a simple numeric processor



General Neural Networks: Example



$$f_{\text{net}}^{(u)}(\vec{w}_u, \vec{\text{in}}_u) = \textstyle\sum_{v \in \text{pred}(u)} w_{uv} \text{in}_{uv} = \sum_{v \in \text{pred}(u)} w_{uv} \, \text{out}_v$$

$$f_{\text{act}}^{(u)}(\text{net}_u, \theta) = \begin{cases} 1, & \text{if} \quad \text{net}_u \geq \theta, \\ 0, & \text{otherwise.} \end{cases}$$

$$f_{\text{out}}^{(u)}(\text{act}_u) = \text{act}_u$$

General Neural Networks: Example

Updating the activations of the neurons

	u_1	u_2	u_3	
input phase	**1**	**0**	**0**	
work phase	1	0	**0**	$\text{net}_{u_3} = -2$
	0	0	0	$\text{net}_{u_1} = 0$
	0	**0**	0	$\text{net}_{u_2} = 0$
	0	0	**0**	$\text{net}_{u_3} = 0$
	0	0	0	$\text{net}_{u_1} = 0$

- Order in which the neurons are updated:
  $u_3, u_1, u_2, u_3, u_1, u_2, u_3, \ldots$
- A stable state with a unique output is reached.

General Neural Networks: Example

Updating the activations of the neurons

	u_1	u_2	u_3	
input phase	**1**	**0**	**0**	
work phase	1	0	**0**	$\text{net}_{u_3} = -2$
	1	**1**	0	$\text{net}_{u_2} = 1$
	0	1	0	$\text{net}_{u_1} = 0$
	0	1	**1**	$\text{net}_{u_3} = 3$
	0	**0**	1	$\text{net}_{u_2} = 0$
	1	0	1	$\text{net}_{u_1} = 4$
	1	0	**0**	$\text{net}_{u_3} = -2$

- Order in which the neurons are updated:
  $u_3, u_2, u_1, u_3, u_2, u_1, u_3, \ldots$
- No stable state is reached (oscillation of output).

General Neural Networks: Training

Definition of learning tasks for a neural network

A **fixed learning task** L_{fixed} for a neural network with

- n input neurons, i.e. $U_{\text{in}} = \{u_1, \ldots, u_n\}$, and
- m output neurons, i.e. $U_{\text{out}} = \{v_1, \ldots, v_m\}$,

is a set of **training patterns** $l = (\vec{\imath}^{\,(l)}, \vec{o}^{\,(l)})$, each consisting of

- an **input vector** $\vec{\imath}^{\,(l)} = (\,\text{ex}_{u_1}^{(l)}, \ldots, \text{ex}_{u_n}^{(l)}\,)$ and
- an **output vector** $\vec{o}^{\,(l)} = (o_{v_1}^{(l)}, \ldots, o_{v_m}^{(l)})$.

A fixed learning task is solved, if for all training patterns $l \in L_{\text{fixed}}$ the neural network computes from the external inputs contained in the input vector $\vec{\imath}^{\,(l)}$ of a training pattern l the outputs contained in the corresponding output vector $\vec{o}^{\,(l)}$.

General Neural Networks: Training

Solving a fixed learning task: Error definition

- Measure how well a neural network solves a given fixed learning task.
- Compute differences between desired and actual outputs.
- Do not sum differences directly in order to avoid errors canceling each other.
- Square has favorable properties for deriving the adaptation rules.

$$e = \sum_{l \in L_{\text{fixed}}} e^{(l)} = \sum_{v \in U_{\text{out}}} e_v = \sum_{l \in L_{\text{fixed}}} \sum_{v \in U_{\text{out}}} e_v^{(l)},$$

$$\text{where} \qquad e_v^{(l)} = \left(o_v^{(l)} - \text{out}_v^{(l)} \right)^2$$

General Neural Networks: Training

Definition of learning tasks for a neural network

A **free learning task** L_{free} for a neural network with

- n input neurons, i.e. $U_{\text{in}} = \{u_1, \ldots, u_n\}$,

is a set of **training patterns** $l = (\vec{\imath}^{\,(l)})$, each consisting of

- an **input vector** $\vec{\imath}^{\,(l)} = (\,\text{ex}_{u_1}^{(l)}, \ldots, \text{ex}_{u_n}^{(l)}\,)$.

Properties:

- There is no desired output for the training patterns.
- Outputs can be chosen freely by the training method.
- Solution idea: **Similar inputs should lead to similar outputs.** (clustering of input vectors)

General Neural Networks: Preprocessing

Normalization of the input vectors

- Compute expected value and standard deviation for each input:

$$\mu_k = \frac{1}{|L|} \sum_{l \in L} \mathrm{ex}_{u_k}^{(l)} \qquad \text{and} \qquad \sigma_k = \sqrt{\frac{1}{|L|} \sum_{l \in L} \left(\mathrm{ex}_{u_k}^{(l)} - \mu_k \right)^2},$$

- Normalize the input vectors to expected value 0 and standard deviation 1:

$$\mathrm{ex}_{u_k}^{(l)(\mathrm{neu})} = \frac{\mathrm{ex}_{u_k}^{(l)(\mathrm{alt})} - \mu_k}{\sigma_k}$$

- Avoids unit and scaling problems.

Chapter 5:

Multilayer Perceptrons

Multilayer Perceptrons

An **r-layered perceptron** is a neural network with a graph $G = (U, C)$ that satisfies the following conditions:

(i) $U_{\text{in}} \cap U_{\text{out}} = \emptyset$,

(ii) $U_{\text{hidden}} = U_{\text{hidden}}^{(1)} \cup \dots \cup U_{\text{hidden}}^{(r-2)}$,

$\forall 1 \leq i < j \leq r-2: \quad U_{\text{hidden}}^{(i)} \cap U_{\text{hidden}}^{(j)} = \emptyset$,

(iii) $C \subseteq \left(U_{\text{in}} \times U_{\text{hidden}}^{(1)}\right) \cup \left(\bigcup_{i=1}^{r-3} U_{\text{hidden}}^{(i)} \times U_{\text{hidden}}^{(i+1)}\right) \cup \left(U_{\text{hidden}}^{(r-2)} \times U_{\text{out}}\right)$

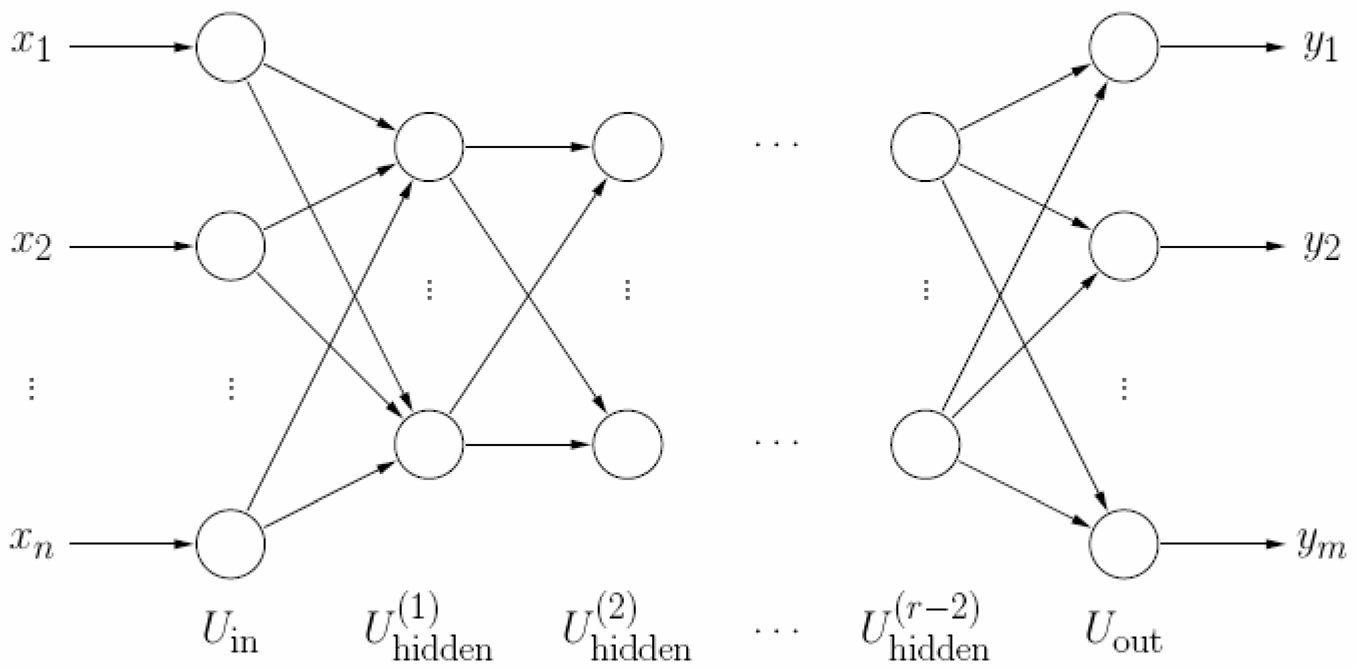
or, if there are no hidden neurons ($r = 2, U_{\text{hidden}} = \emptyset$),

$C \subseteq U_{\text{in}} \times U_{\text{out}}$.

- Feed-forward network with strictly layered structure.

Multilayer Perceptrons

General structure of a multilayer perceptron



Multilayer Perceptrons

- The network input function of each hidden neuron and of each output neuron is the **weighted sum** of its inputs, i.e.

$$\forall u \in U_{\text{hidden}} \cup U_{\text{out}} : \qquad f_{\text{net}}^{(u)}(\vec{w}_u, \vec{\text{in}}_u) = \vec{w}_u \vec{\text{in}}_u = \sum_{v \in \text{pred}\,(u)} w_{uv}\, \text{out}_v\,.$$

- The activation function of each hidden neuron is a so-called **sigmoid function**, i.e. a monotonously increasing function

$$f : \mathbb{R} \to [0, 1] \quad \text{with} \quad \lim_{x \to -\infty} f(x) = 0 \quad \text{and} \quad \lim_{x \to \infty} f(x) = 1.$$

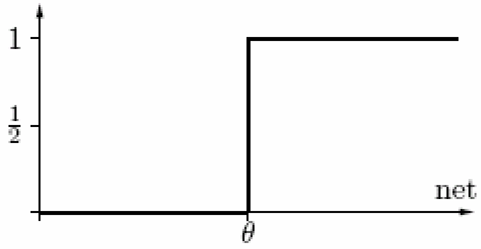
- The activation function of each output neuron is either also a sigmoid function or a **linear function**, i.e.

$$f_{\text{act}}(\text{net}, \theta) = \alpha\, \text{net} - \theta.$$

Sigmoid Activation Functions

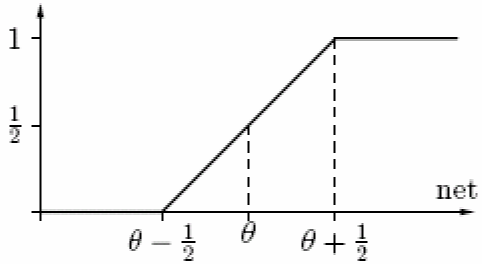
step function:

$$f_{\text{act}}(\text{net}, \theta) = \begin{cases} 1, & \text{if net} \geq \theta, \\ 0, & \text{otherwise.} \end{cases}$$



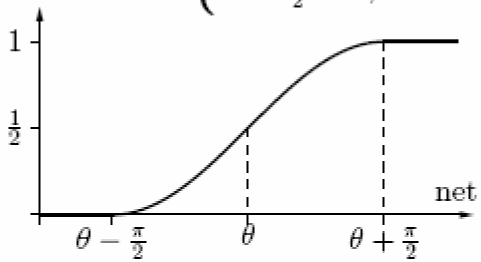
semi-linear function:

$$f_{\text{act}}(\text{net}, \theta) = \begin{cases} 1, & \text{if net} > \theta + \frac{1}{2}, \\ 0, & \text{if net} < \theta - \frac{1}{2}, \\ (\text{net} - \theta) + \frac{1}{2}, & \text{otherwise.} \end{cases}$$



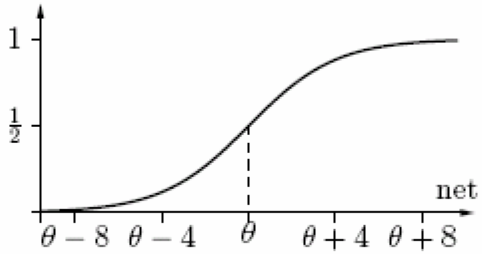
sine until saturation:

$$f_{\text{act}}(\text{net}, \theta) = \begin{cases} 1, & \text{if net} > \theta + \frac{\pi}{2}, \\ 0, & \text{if net} < \theta - \frac{\pi}{2}, \\ \frac{\sin(\text{net} - \theta) + 1}{2}, & \text{otherwise.} \end{cases}$$



logistic function:

$$f_{\text{act}}(\text{net}, \theta) = \frac{1}{1 + e^{-(\text{net} - \theta)}}$$

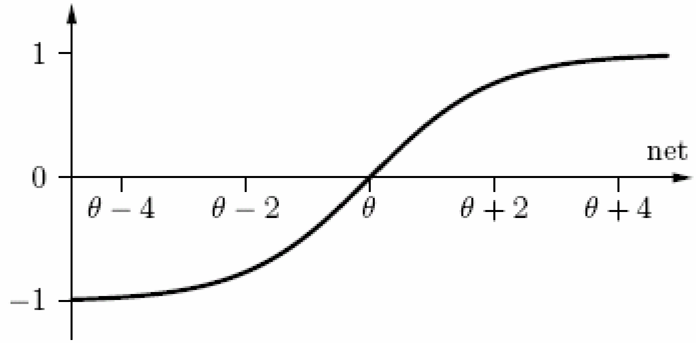


Sigmoid Activation Functions

- All sigmoid functions on the previous slide are **unipolar**, i.e., they range from 0 to 1.
- Sometimes **bipolar** sigmoid functions are used, like the *tangens hyperbolicus*.

tangens hyperbolicus:

$$f_{\text{act}}(\text{net}, \theta) = \tanh(\text{net} - \theta) = \frac{2}{1 + e^{-2(\text{net} - \theta)}} - 1$$



Multilayer Perceptrons: Weight Matrices

Let $U_1 = \{v_1, \ldots, v_m\}$ and $U_2 = \{u_1, \ldots, u_n\}$ be the neurons of two consecutive layers of a multilayer perceptron.

Their connection weights are represented by an $n \times m$ matrix

$$\mathbf{W} = \begin{pmatrix} w_{u_1 v_1} & w_{u_1 v_2} & \ldots & w_{u_1 v_m} \\ w_{u_2 v_1} & w_{u_2 v_2} & \ldots & w_{u_2 v_m} \\ \vdots & \vdots & & \vdots \\ w_{u_n v_1} & w_{u_n v_2} & \ldots & w_{u_n v_m} \end{pmatrix},$$

where $w_{u_i v_j} = 0$ if there is no connection from neuron v_j to neuron u_i.

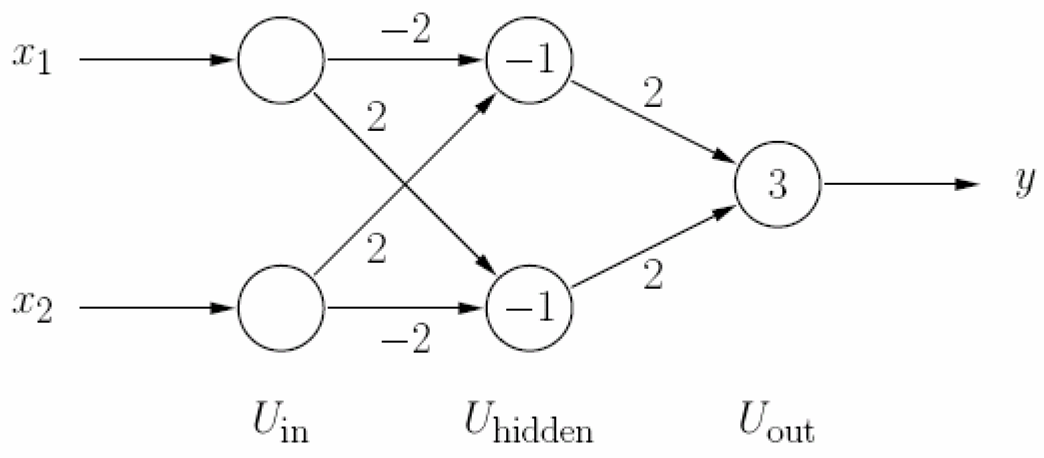
Advantage: The computation of the network input can be written as

$$\vec{\mathrm{net}}_{U_2} = \mathbf{W} \cdot \vec{\mathrm{in}}_{U_2} = \mathbf{W} \cdot \vec{\mathrm{out}}_{U_1}$$

where $\vec{\mathrm{net}}_{U_2} = (\mathrm{net}_{u_1}, \ldots, \mathrm{net}_{u_n})^T$ and $\vec{\mathrm{in}}_{U_2} = \vec{\mathrm{out}}_{U_1} = (\mathrm{out}_{v_1}, \ldots, \mathrm{out}_{v_m})^T$.

Multilayer Perceptrons: Biimplication

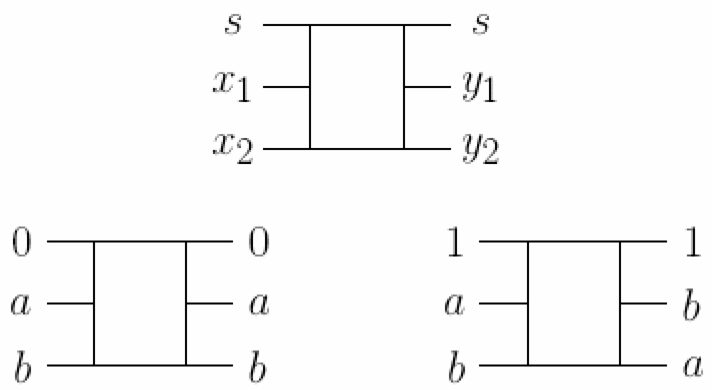
Solving the biimplication problem with a multilayer perceptron.



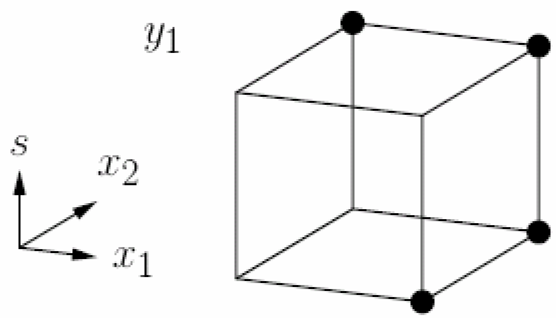
Note the additional input neurons compared to the TLU solution.

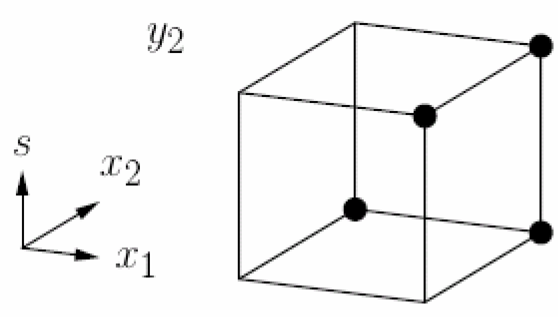
$$\mathbf{W}_1 = \begin{pmatrix} -2 & 2 \\ 2 & -2 \end{pmatrix} \qquad \text{and} \qquad \mathbf{W}_2 = \begin{pmatrix} 2 & 2 \end{pmatrix}$$

Multilayer Perceptrons: Fredkin Gate

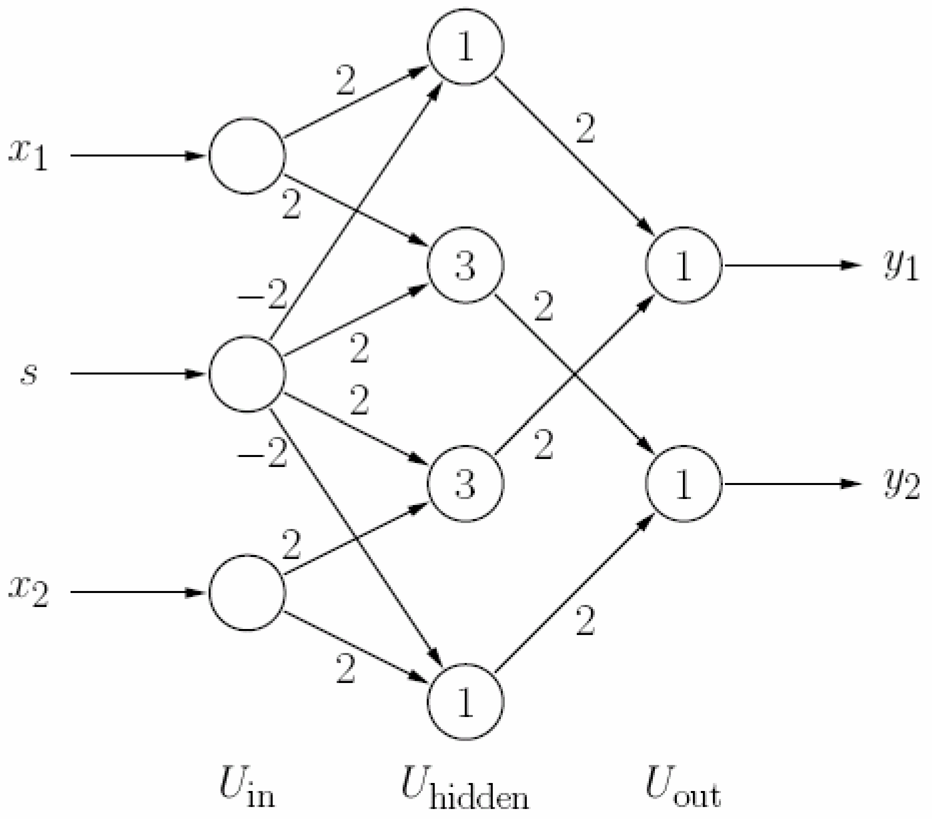


s	0	0	0	0	1	1	1	1
x_1	0	0	1	1	0	0	1	1
x_2	0	1	0	1	0	1	0	1
y_1	0	0	1	1	0	1	0	1
y_2	0	1	0	1	0	0	1	1





Multilayer Perceptrons: Fredkin Gate



$$\mathbf{W}_1 = \begin{pmatrix} 2 & -2 & 0 \\ 2 & 2 & 0 \\ 0 & 2 & 2 \\ 0 & -2 & 2 \end{pmatrix}$$

$$\mathbf{W}_2 = \begin{pmatrix} 2 & 0 & 2 & 0 \\ 0 & 2 & 0 & 2 \end{pmatrix}$$

Why Non-linear Activation Functions?

With weight matrices we have for two consecutive layers U_1 and U_2

$$\vec{\text{net}}_{U_2} = \mathbf{W} \cdot \vec{\text{in}}_{U_2} = \mathbf{W} \cdot \vec{\text{out}}_{U_1}.$$

If the activation functions are linear, i.e.,

$$f_{\text{act}}(\text{net}, \theta) = \alpha \,\text{net} - \theta.$$

the activations of the neurons in the layer U_2 can be computed as

$$\vec{\text{act}}_{U_2} = \mathbf{D}_{\text{act}} \cdot \vec{\text{net}}_{U_2} - \vec{\theta},$$

where

- $\vec{\text{act}}_{U_2} = (\text{act}_{u_1}, \ldots, \text{act}_{u_n})^T$ is the activation vector,
- $\mathbf{D}_{\text{act}}$ is an $n \times n$ diagonal matrix of the factors α_{u_i}, $i = 1, \ldots, n$, and
- $\vec{\theta} = (\theta_{u_1}, \ldots, \theta_{u_n})^T$ is a bias vector.