Neural Networks

Prof. Dr. Rudolf Kruse

Computational Intelligence Group
Faculty for Computer Science
kruse@iws.cs.uni-magdeburg.de

Multilayer Perceptrons (MLPs)

Multilayer Perceptrons

An **r layer 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) $U_{\text{hidden}} = U_{\text{hidden}}^{(1)} \cup \cdots \cup U_{\text{hidden}}^{(r-2)}$,

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

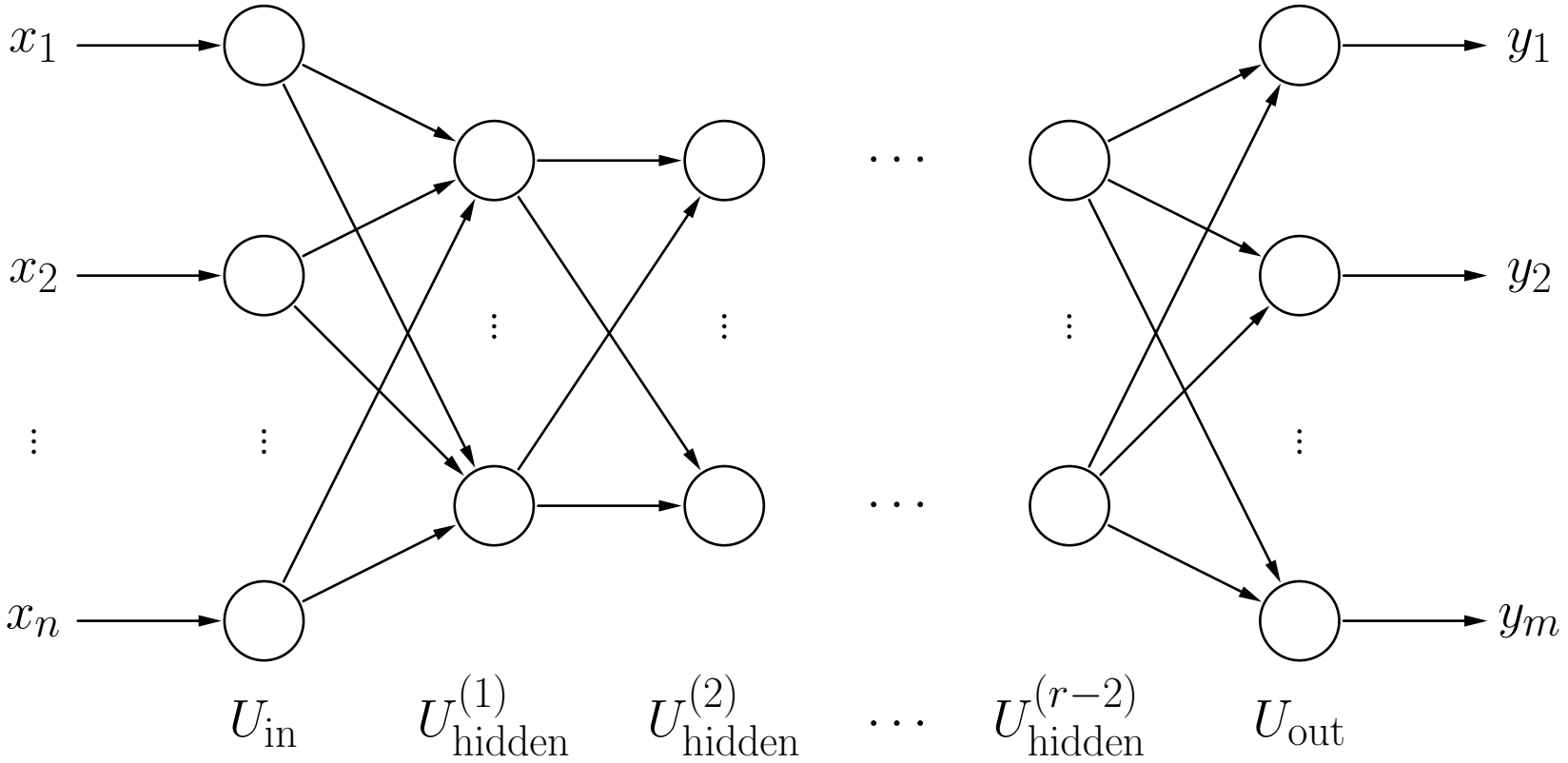
(iii) $C \subseteq \left(U_{\text{in}} \times U_{\text{hidden}}^{(1)}\right) \cup \left(\bigcup_{i=1}^{r-3} U_{\text{hidden}}^{(i)} \times U_{\text{hidden}}^{(i+1)}\right) \cup \left(U_{\text{hidden}}^{(r-2)} \times U_{\text{out}}\right)$

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.

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.

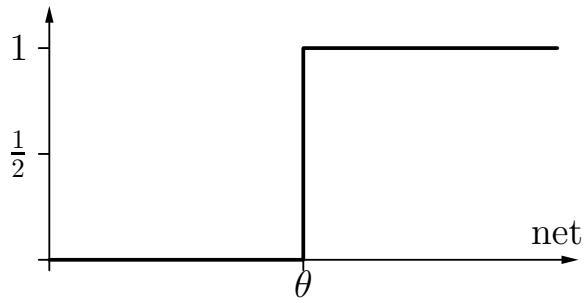
$$f_{\text{act}}(\text{net}, \theta) = \alpha\, \text{net} - \theta.$$

- Only the step function serves as a neurologically plausible activation function.

Sigmoid Activation Functions

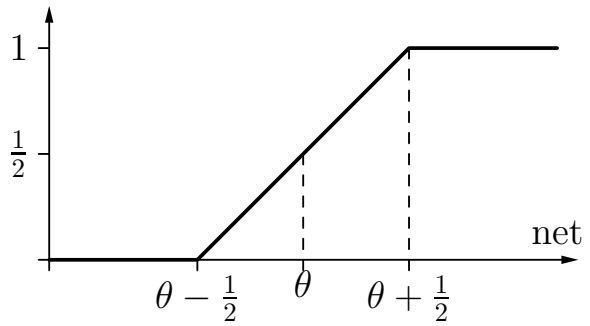
step function:

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



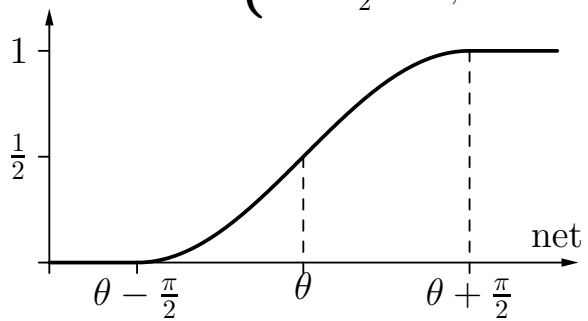
semi-linear function:

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



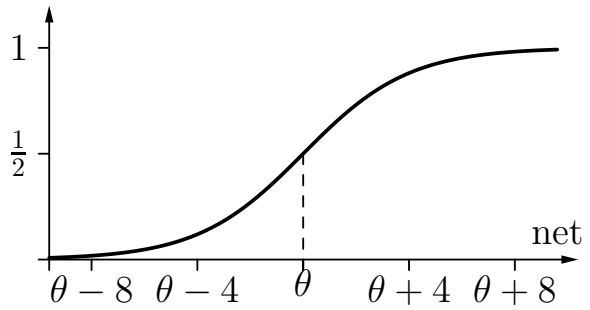
sine until saturation:

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



logistic function:

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

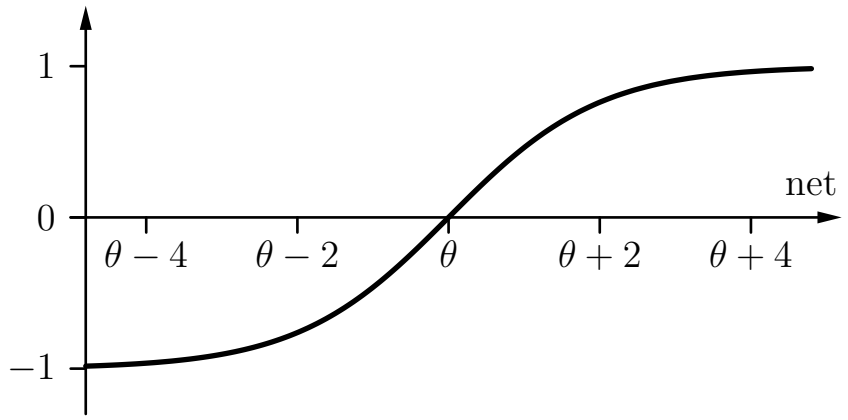


Sigmoid Activation Functions

- 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

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

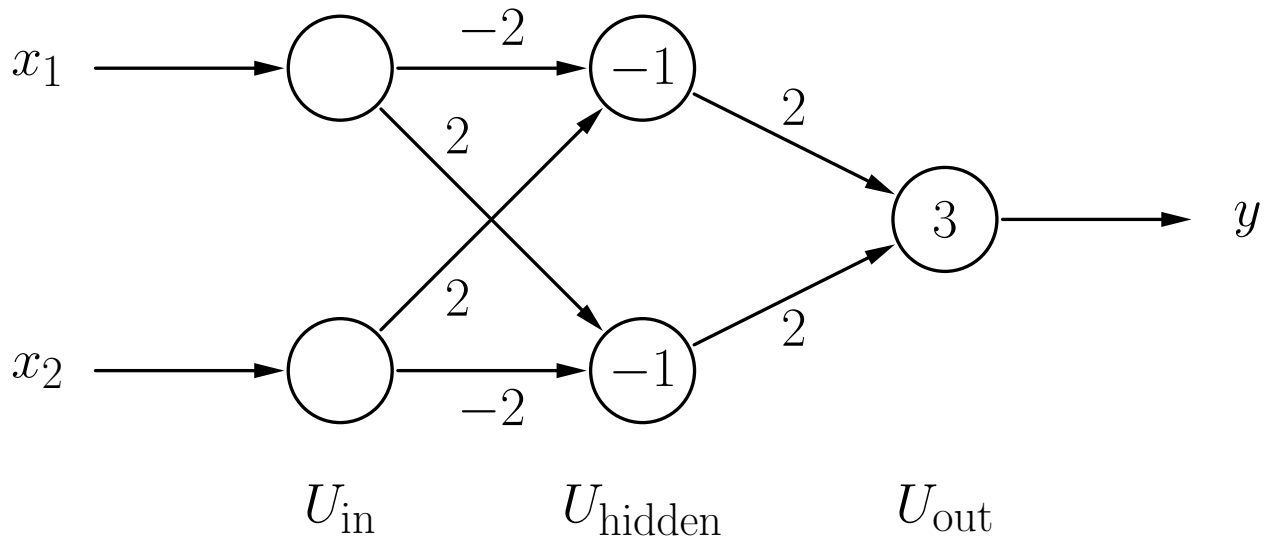
where $w_{u_i v_j} = 0$ if there is no connection from neuron v_j to neuron u_i.

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

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

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

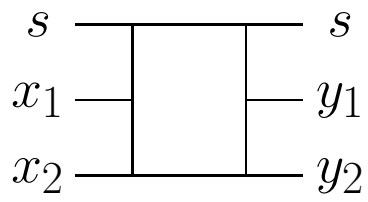
Solving the biimplication problem with a multilayer perceptron.

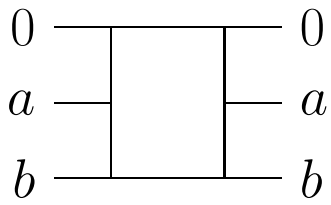


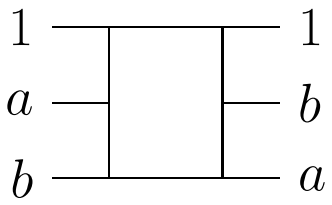
Note the additional input neurons compared to the TLU solution.

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

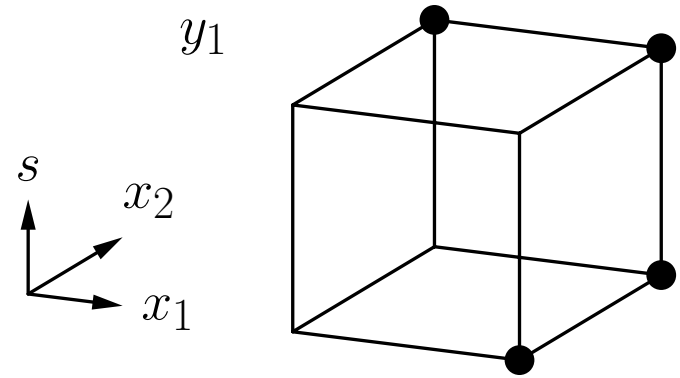
Multilayer Perceptrons: Fredkin Gate

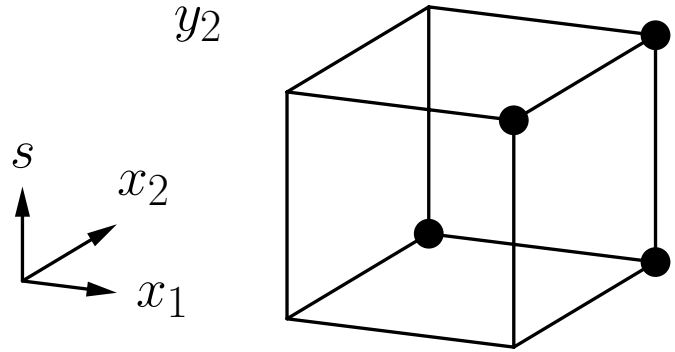




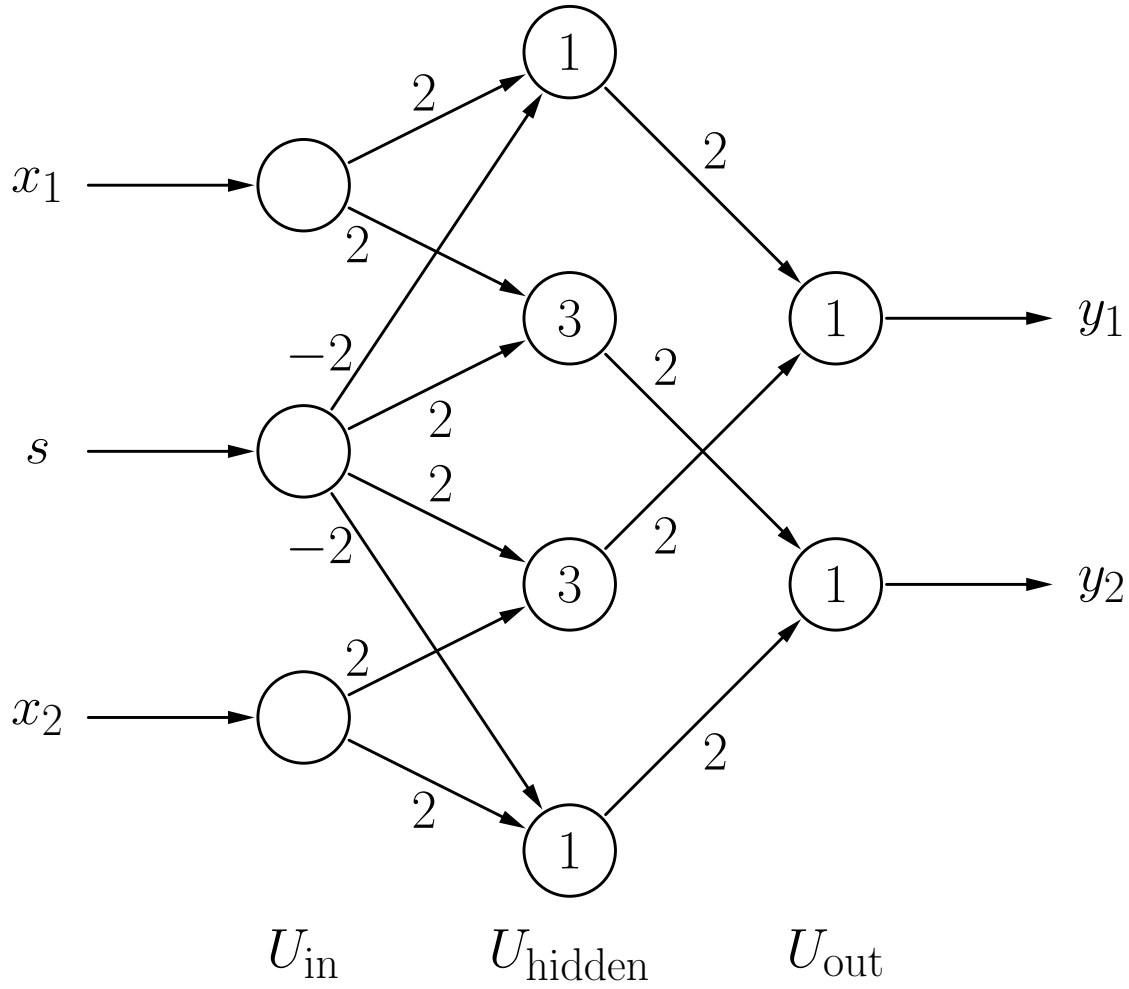


s	0	0	0	0	1	1	1	1
x_1	0	0	1	1	0	0	1	1
x_2	0	1	0	1	0	1	0	1
y_1	0	0	1	1	0	1	0	1
y_2	0	1	0	1	0	0	1	1





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

- $\vec{\text{act}}_{U_2} = (\text{act}_{u_1}, \ldots, \text{act}_{u_n})^\top$ 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
- $\vec{\theta} = (\theta_{u_1}, \ldots, \theta_{u_n})^\top$ is a bias vector.

Why Non-linear Activation Functions?

If the output function is also linear, it is analogously

$$\vec{\text{out}}_{U_2} = \mathbf{D}_{\text{out}} \cdot \vec{\text{act}}_{U_2} - \vec{\xi},$$

where

- $\vec{\text{out}}_{U_2} = (\text{out}_{u_1}, \ldots, \text{out}_{u_n})^\top$ is the output vector,
- $\mathbf{D}_{\text{out}}$ is again an $n \times n$ diagonal matrix of factors, and
- $\vec{\xi} = (\xi_{u_1}, \ldots, \xi_{u_n})^\top$ a bias vector.

Combining these computations we get

$$\vec{\text{out}}_{U_2} = \mathbf{D}_{\text{out}} \cdot \left(\mathbf{D}_{\text{act}} \cdot \left(\mathbf{W} \cdot \vec{\text{out}}_{U_1}\right) - \vec{\theta}\right) - \vec{\xi}$$

and thus

$$\vec{\text{out}}_{U_2} = \mathbf{A}_{12} \cdot \vec{\text{out}}_{U_1} + \vec{b}_{12}$$

with an $n \times m$ matrix $\mathbf{A}_{12}$ and an n-dimensional vector $\vec{b}_{12}$.

Why Non-linear Activation Functions?

Therefore we have

$$\vec{\text{out}}_{U_2} = \mathbf{A}_{12} \cdot \vec{\text{out}}_{U_1} + \vec{b}_{12}$$

and

$$\vec{\text{out}}_{U_3} = \mathbf{A}_{23} \cdot \vec{\text{out}}_{U_2} + \vec{b}_{23}$$

for the computations of two consecutive layers U_2 and U_3.

These two computations can be combined into

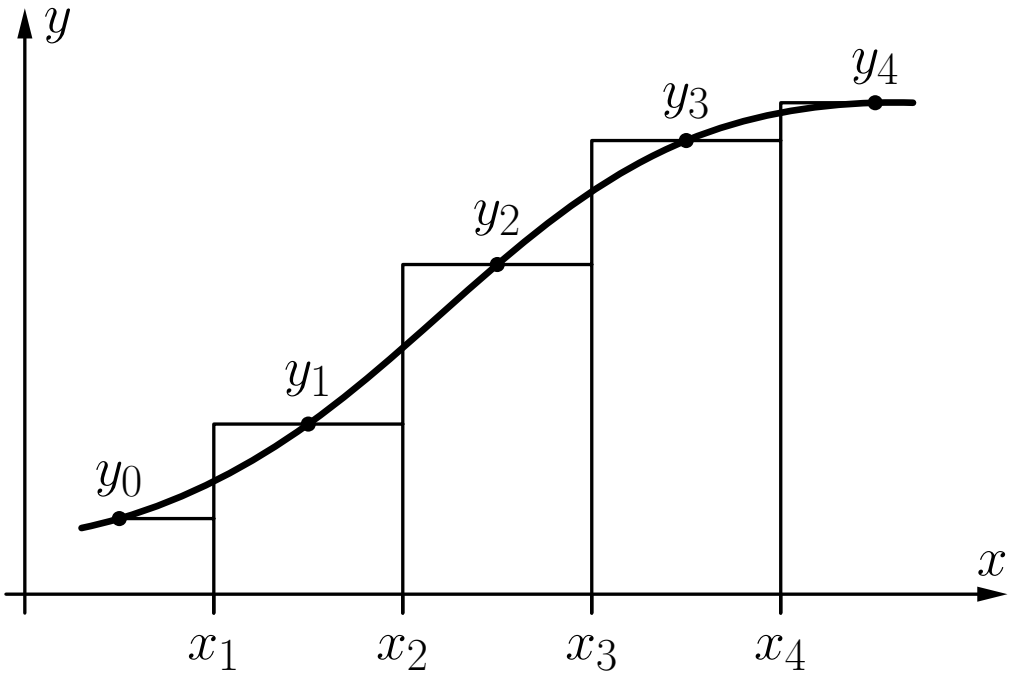
$$\vec{\text{out}}_{U_3} = \mathbf{A}_{13} \cdot \vec{\text{out}}_{U_1} + \vec{b}_{13},$$

where $\mathbf{A}_{13} = \mathbf{A}_{23} \cdot \mathbf{A}_{12}$ and $\vec{b}_{13} = \mathbf{A}_{23} \cdot \vec{b}_{12} + \vec{b}_{23}$.

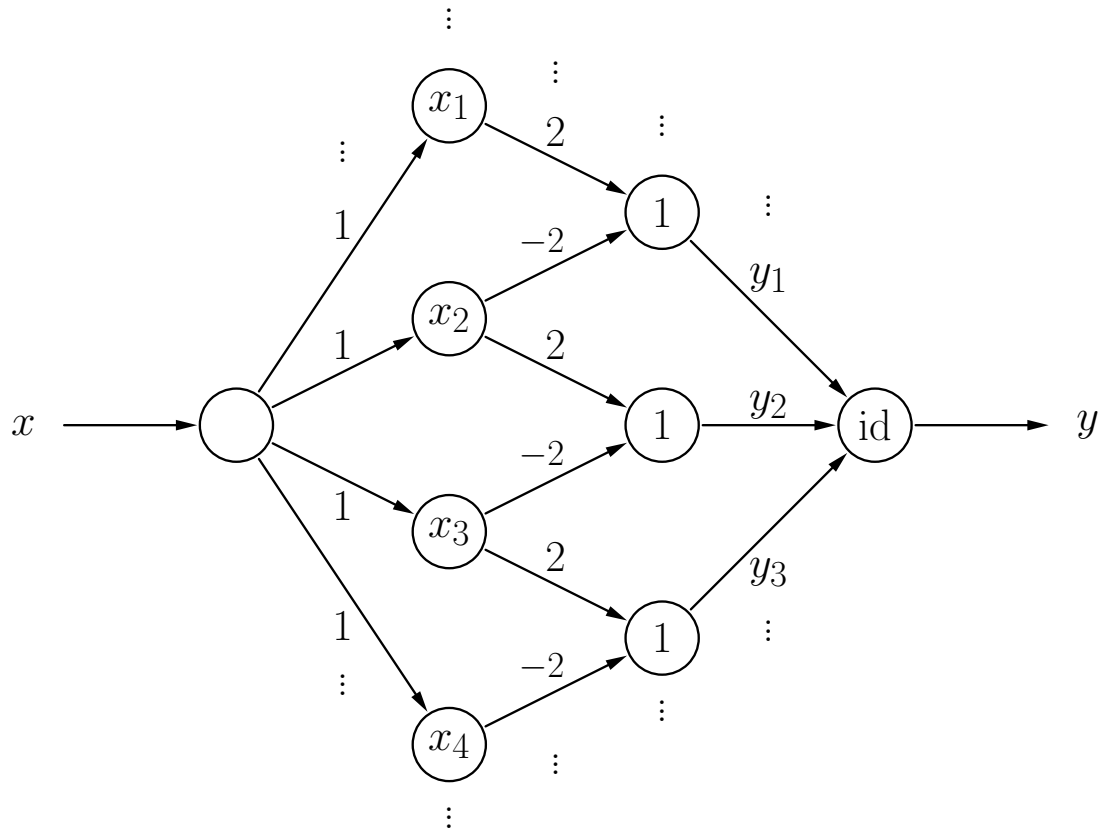
Result: With linear activation and output functions any multilayer perceptron can be reduced to a two-layer perceptron.

General idea of function approximation

- Approximate a given function by a step function.
- Construct a neural network that computes the step function.



Multilayer Perceptrons: Function Approximation

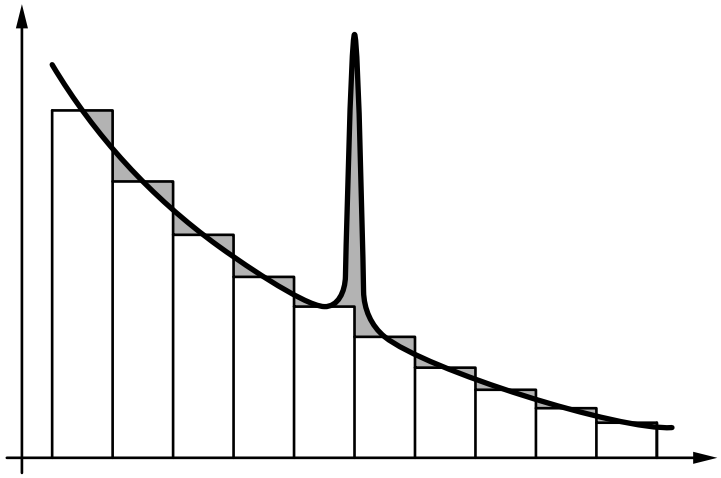


A neural network calculating the step function shown on the previous slide. Only one step is active at a time, and it's output is the step height.
Note: The output neuron is not a threshold logic unit. It's output is the identity of it's input.

Multilayer Perceptrons: Function Approximation

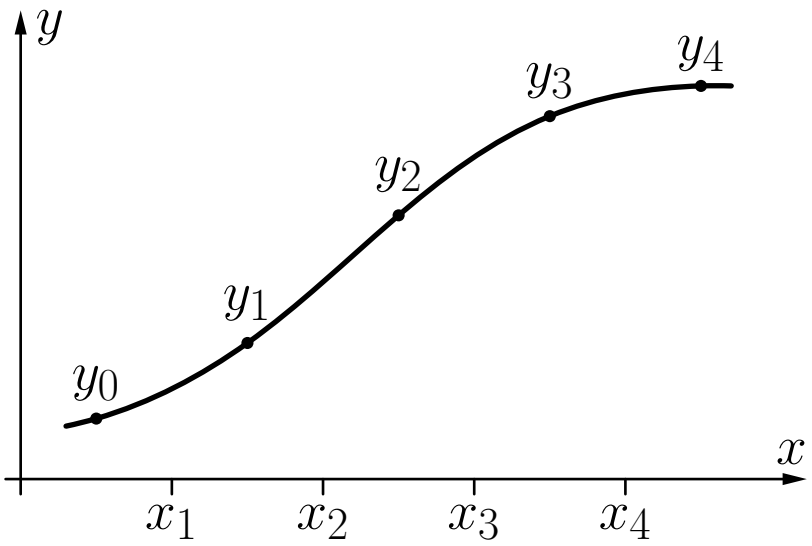
Theorem: Any Riemann-integrable function can be approximated with arbitrary accuracy by a four-layer perceptron.

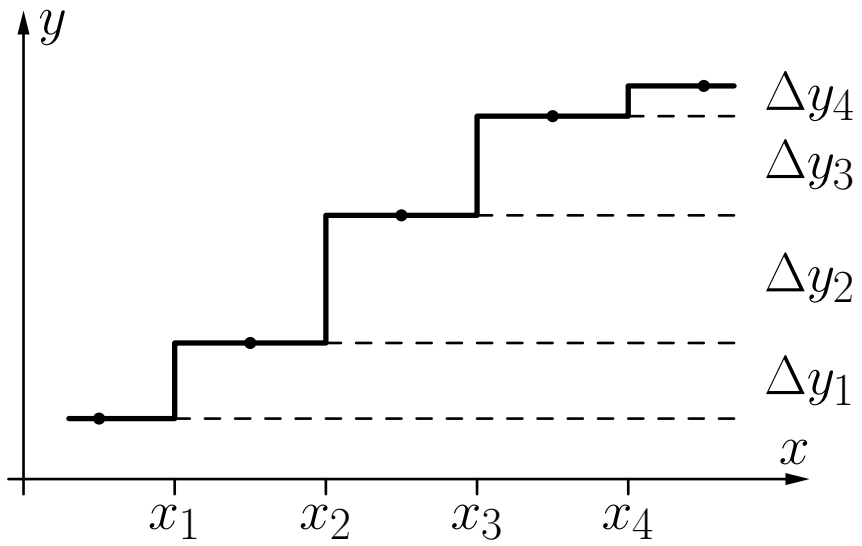
- But: Error is measured as the **area** between the functions.

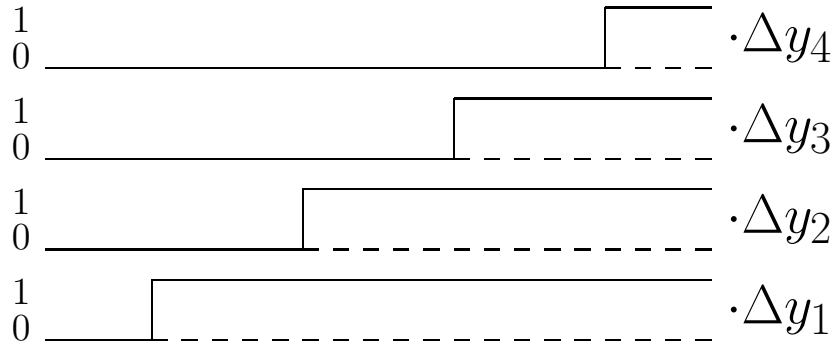


- More sophisticated mathematical examination allows a stronger assertion:
  With a three-layer perceptron any continuous function can be approximated with arbitrary accuracy (error: maximum function value difference).

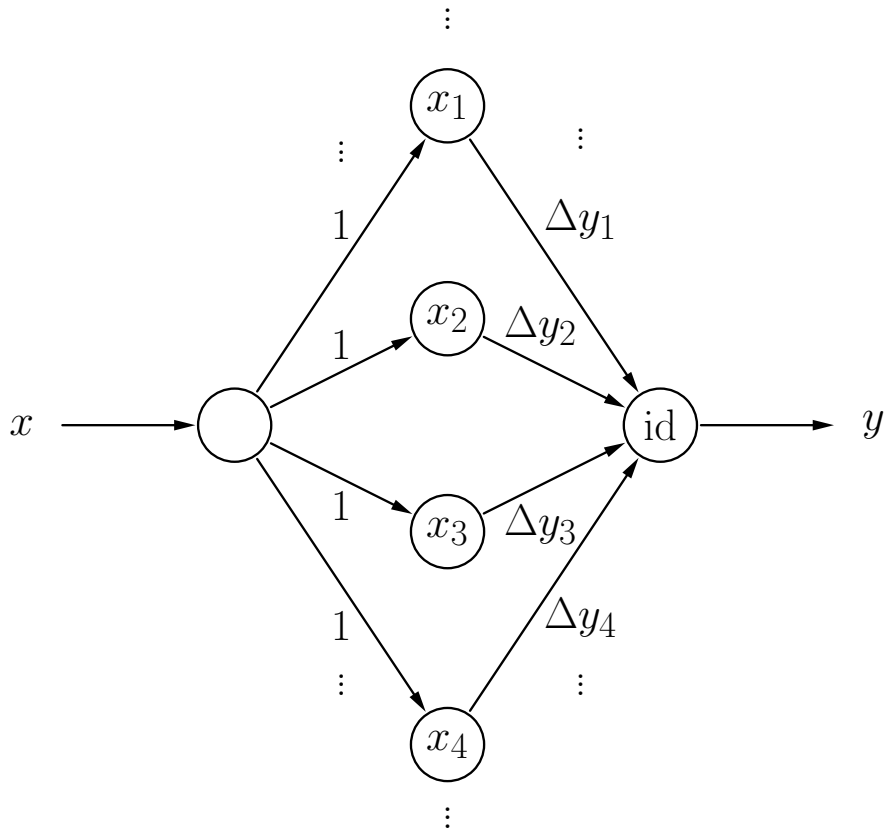
Multilayer Perceptrons: Function Approximation





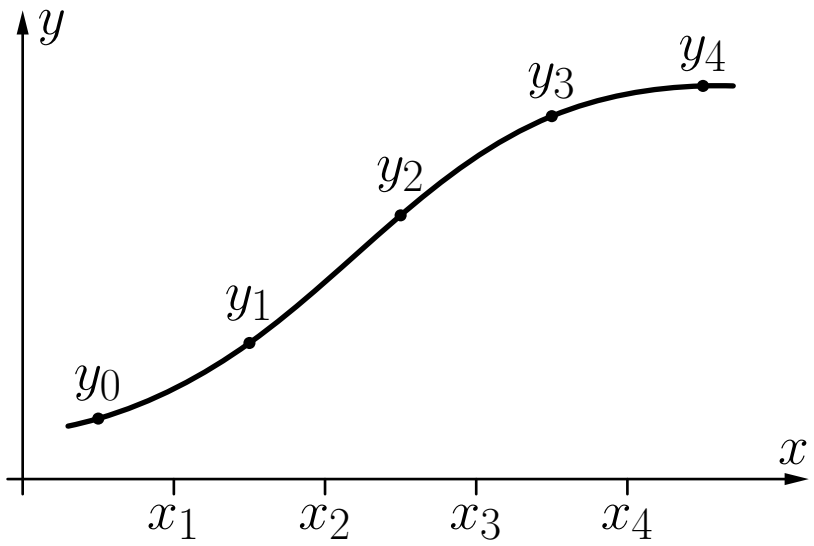


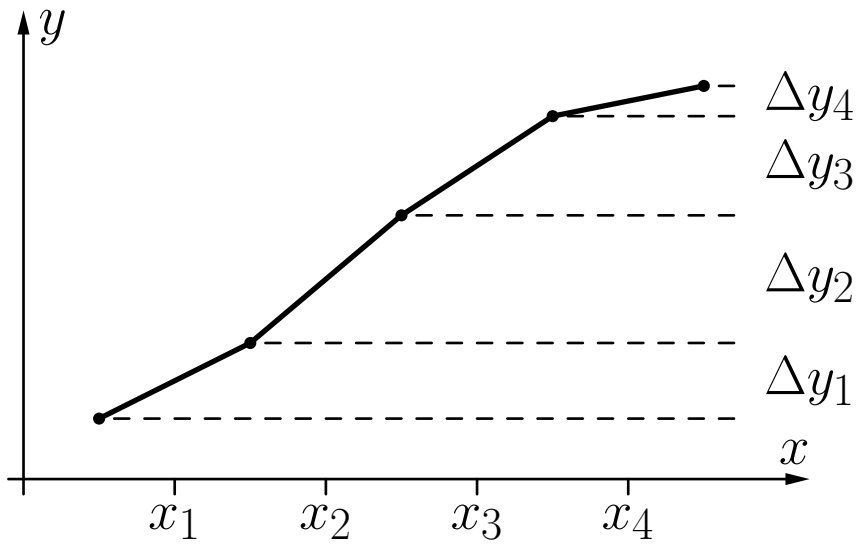
Multilayer Perceptrons: Function Approximation

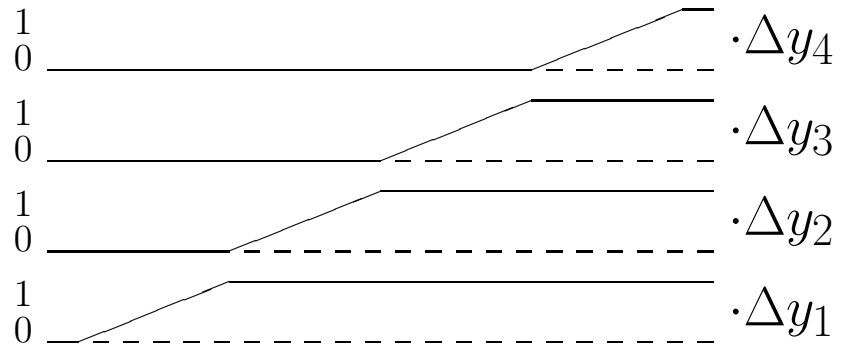


A neural network calculating the step function shown on the previous slide as the weighted sum of 1-step functions.

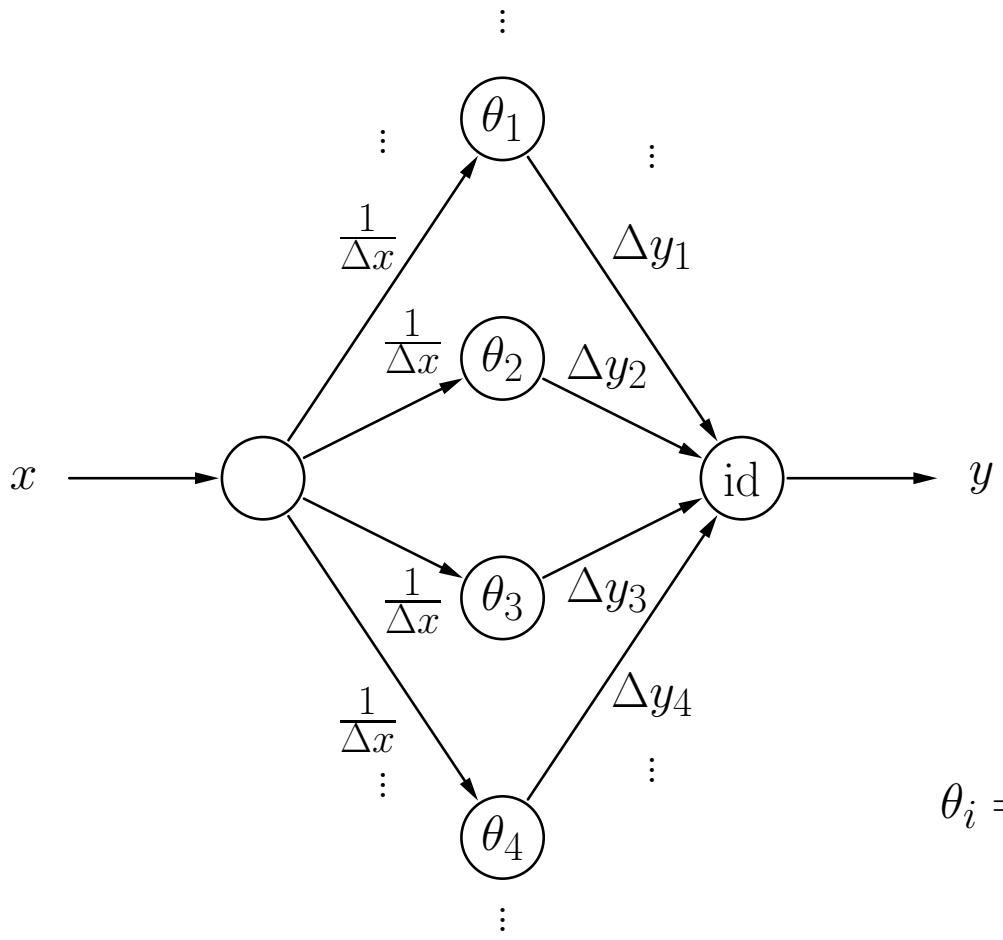
Multilayer Perceptrons: Function Approximation







Multilayer Perceptrons: Function Approximation



$$\theta_i = \frac{x_i}{\Delta x}$$

A neural network calculating the partially linear function shown on the previous slide using the weighted sum of semi-linear functions, with $\Delta x = x_{i+1} - x_i$.

Mathematical Background: Regression

Training neural networks is closely related to regression

Given:
- A dataset $((x_1, y_1), \ldots, (x_n, y_n))$ of n data tuples and
- a hypothesis about the functional relationship, e.g. $y = g(x) = a + bx$.

Approach: Minimize the sum of squared errors, i.e.

$$F(a, b) = \sum_{i=1}^{n} (g(x_i) - y_i)^2 = \sum_{i=1}^{n} (a + bx_i - y_i)^2.$$

Necessary conditions for a minimum:

$$\frac{\partial F}{\partial a} = \sum_{i=1}^{n} 2(a + bx_i - y_i) = 0 \quad \text{and}$$

$$\frac{\partial F}{\partial b} = \sum_{i=1}^{n} 2(a + bx_i - y_i)x_i = 0$$

Mathematical Background: Linear Regression

Result of necessary conditions: System of so-called **normal equations**, i.e.

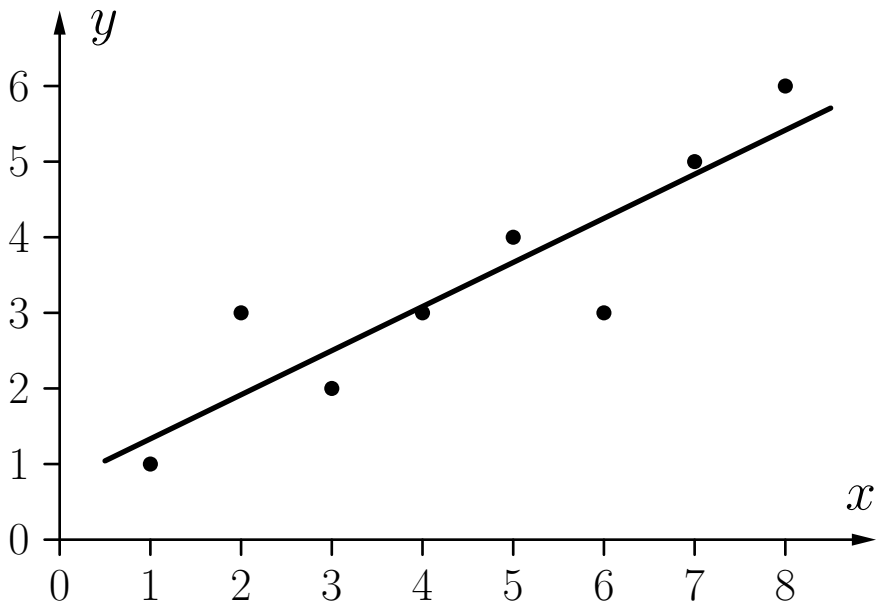
$$na + \left(\sum_{i=1}^{n} x_i\right) b = \sum_{i=1}^{n} y_i,$$
$$\left(\sum_{i=1}^{n} x_i\right) a + \left(\sum_{i=1}^{n} x_i^2\right) b = \sum_{i=1}^{n} x_i y_i.$$

- Two linear equations for two unknowns a and b.
- System can be solved with standard methods from linear algebra.
- Solution is unique unless all x-values are identical.
- The resulting line is called a **regression line**.

Linear Regression: Example

x	1	2	3	4	5	6	7	8
y	1	3	2	3	4	3	5	6

$$y = \frac{3}{4} + \frac{7}{12}x.$$



Generalization to polynomials

$$y = p(x) = a_0 + a_1 x + \ldots + a_m x^m$$

Approach: Minimize the sum of squared errors, i.e.

$$F(a_0, a_1, \ldots, a_m) \;=\; \sum_{i=1}^{n} (p(x_i) - y_i)^2 \;=\; \sum_{i=1}^{n} (a_0 + a_1 x_i + \ldots + a_m x_i^m - y_i)^2$$

Necessary conditions for a minimum: All partial derivatives vanish, i.e.

$$\frac{\partial F}{\partial a_0} = 0, \quad \frac{\partial F}{\partial a_1} = 0, \quad \ldots \quad , \frac{\partial F}{\partial a_m} = 0.$$

System of normal equations for polynomials

$$
\begin{aligned}
n a_0 + \left(\sum_{i=1}^{n} x_i\right) a_1 + \ldots + \left(\sum_{i=1}^{n} x_i^m\right) a_m &= \sum_{i=1}^{n} y_i \\
\left(\sum_{i=1}^{n} x_i\right) a_0 + \left(\sum_{i=1}^{n} x_i^2\right) a_1 + \ldots + \left(\sum_{i=1}^{n} x_i^{m+1}\right) a_m &= \sum_{i=1}^{n} x_i y_i \\
\vdots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad &\vdots \\
\left(\sum_{i=1}^{n} x_i^m\right) a_0 + \left(\sum_{i=1}^{n} x_i^{m+1}\right) a_1 + \ldots + \left(\sum_{i=1}^{n} x_i^{2m}\right) a_m &= \sum_{i=1}^{n} x_i^m y_i,
\end{aligned}
$$

- $m+1$ linear equations for $m+1$ unknowns $a_0, \ldots, a_m$.
- System can be solved with standard methods from linear algebra.
- Solution is unique unless all x-values are identical.

Generalization to more than one argument

$$z = f(x, y) = a + bx + cy$$

Approach: Minimize the sum of squared errors, i.e.

$$F(a, b, c) \;=\; \sum_{i=1}^{n} (f(x_i, y_i) - z_i)^2 \;=\; \sum_{i=1}^{n} (a + bx_i + cy_i - z_i)^2$$

Necessary conditions for a minimum: All partial derivatives vanish, i.e.

$$\begin{aligned}
\frac{\partial F}{\partial a} &= \sum_{i=1}^{n} 2(a + bx_i + cy_i - z_i) &&= 0, \\
\frac{\partial F}{\partial b} &= \sum_{i=1}^{n} 2(a + bx_i + cy_i - z_i)x_i &&= 0, \\
\frac{\partial F}{\partial c} &= \sum_{i=1}^{n} 2(a + bx_i + cy_i - z_i)y_i &&= 0.
\end{aligned}$$

System of normal equations for several arguments

$$
\begin{aligned}
na + \left(\sum_{i=1}^{n} x_i\right) b + \left(\sum_{i=1}^{n} y_i\right) c &= \sum_{i=1}^{n} z_i \\
\left(\sum_{i=1}^{n} x_i\right) a + \left(\sum_{i=1}^{n} x_i^2\right) b + \left(\sum_{i=1}^{n} x_i y_i\right) c &= \sum_{i=1}^{n} z_i x_i \\
\left(\sum_{i=1}^{n} y_i\right) a + \left(\sum_{i=1}^{n} x_i y_i\right) b + \left(\sum_{i=1}^{n} y_i^2\right) c &= \sum_{i=1}^{n} z_i y_i
\end{aligned}
$$

- 3 linear equations for 3 unknowns a, b, and c.
- System can be solved with standard methods from linear algebra.
- Solution is unique unless all x- or all y-values are identical.

General multilinear case:

$$y = f(x_1, \ldots, x_m) = a_0 + \sum_{k=1}^{m} a_k x_k$$

Approach: Minimize the sum of squared errors, i.e.

$$F(\vec{a}) = (\mathbf{X}\vec{a} - \vec{y})^\top (\mathbf{X}\vec{a} - \vec{y}),$$

where

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & \ldots & x_{m1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{1n} & \ldots & x_{mn} \end{pmatrix}, \qquad \vec{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}, \qquad \text{and} \qquad \vec{a} = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{pmatrix}$$

Necessary conditions for a minimum:

$$\nabla_{\vec{a}} F(\vec{a}) = \nabla_{\vec{a}} (\mathbf{X}\vec{a} - \vec{y})^\top (\mathbf{X}\vec{a} - \vec{y}) = \vec{0}$$

Multilinear Regression

- $\nabla_{\vec{a}} F(\vec{a})$ may easily be computed by remembering that the differential operator

$$\nabla_{\vec{a}} = \left(\frac{\partial}{\partial a_0}, \ldots, \frac{\partial}{\partial a_m} \right)$$

  behaves formally like a vector that is "multiplied" to the sum of squared errors.

- Alternatively, one may write out the differentiation componentwise.

With the former method we obtain for the derivative:

$$\begin{aligned}
& \nabla_{\vec{a}} \left(\mathbf{X}\vec{a} - \vec{y} \right)^\top (\mathbf{X}\vec{a} - \vec{y}) \\
&= \left(\nabla_{\vec{a}} \left(\mathbf{X}\vec{a} - \vec{y} \right) \right)^\top (\mathbf{X}\vec{a} - \vec{y}) + \left((\mathbf{X}\vec{a} - \vec{y})^\top \left(\nabla_{\vec{a}} \left(\mathbf{X}\vec{a} - \vec{y} \right) \right) \right)^\top \\
&= \left(\nabla_{\vec{a}} \left(\mathbf{X}\vec{a} - \vec{y} \right) \right)^\top (\mathbf{X}\vec{a} - \vec{y}) + \left(\nabla_{\vec{a}} \left(\mathbf{X}\vec{a} - \vec{y} \right) \right)^\top (\mathbf{X}\vec{a} - \vec{y}) \\
&= 2\mathbf{X}^\top (\mathbf{X}\vec{a} - \vec{y}) \\
&= 2\mathbf{X}^\top \mathbf{X}\vec{a} - 2\mathbf{X}^\top \vec{y} \quad = \quad \vec{0}
\end{aligned}$$

Multilinear Regression

A few rules for vector / matrix calculations and derivations:

$$
\begin{aligned}
(\mathbf{A}+\mathbf{B})^\top &= \mathbf{A}^\top + \mathbf{B}^\top \\
(\mathbf{AB})^\top &= \mathbf{B}^\top \mathbf{A}^\top \\
\nabla_{\vec{z}} \mathbf{A}\vec{z} &= \mathbf{A}
\end{aligned}
\qquad\qquad
\begin{aligned}
\nabla_{\vec{z}} f(\vec{z})\mathbf{A} &= (\nabla_{\vec{z}} f(\vec{z}))\mathbf{A} \\
\nabla_{\vec{z}} (f(\vec{z}))^\top &= (\nabla_{\vec{z}} f(\vec{z}))^\top \\
\nabla_{\vec{z}} f(\vec{z})g(\vec{z}) &= (\nabla_{\vec{z}} f(\vec{z}))g(\vec{z}) + f(\vec{z})(\nabla_{\vec{z}} g(\vec{z}))^\top
\end{aligned}
$$

Derivative of the function to be minimized:

$$
\begin{aligned}
\nabla_{\vec{a}} F(\vec{a}) &= \nabla_{\vec{a}} (\mathbf{X}\vec{a} - \vec{y})^\top (\mathbf{X}\vec{a} - \vec{y}) \\
&= \nabla_{\vec{a}} ((\mathbf{X}\vec{a})^\top - \vec{y}^\top)(\mathbf{X}\vec{a} - \vec{y}) \\
&= \nabla_{\vec{a}} ((\mathbf{X}\vec{a})^\top \mathbf{X}\vec{a} - (\mathbf{X}\vec{a})^\top \vec{y} - \vec{y}^\top \mathbf{X}\vec{a} + \vec{y}^\top \vec{y}) \\
&= \nabla_{\vec{a}} (\mathbf{X}\vec{a})^\top \mathbf{X}\vec{a} - \nabla_{\vec{a}} (\mathbf{X}\vec{a})^\top \vec{y} - \nabla_{\vec{a}} \vec{y}^\top \mathbf{X}\vec{a} + \nabla_{\vec{a}} \vec{y}^\top \vec{y} \\
&= (\nabla_{\vec{a}} (\mathbf{X}\vec{a})^\top)\mathbf{X}\vec{a} + ((\mathbf{X}\vec{a})^\top (\nabla_{\vec{a}} \mathbf{X}\vec{a}))^\top - 2\nabla_{\vec{a}} (\mathbf{X}\vec{a})^\top \vec{y} \\
&= ((\nabla_{\vec{a}} \mathbf{X}\vec{a})^\top)\mathbf{X}\vec{a} + (\nabla_{\vec{a}} \mathbf{X}\vec{a})^\top \mathbf{X}\vec{a} - 2(\nabla_{\vec{a}} (\mathbf{X}\vec{a})^\top)\vec{y} \\
&= 2(\nabla_{\vec{a}} \mathbf{X}\vec{a})^\top \mathbf{X}\vec{a} - 2(\nabla_{\vec{a}} \mathbf{X}\vec{a})^\top \vec{y} \\
&= 2\mathbf{X}^\top \mathbf{X}\vec{a} - 2\mathbf{X}^\top \vec{y}
\end{aligned}
$$

Multilinear Regression

Necessary condition for a minimum therefore:

$$\begin{aligned} \nabla_{\vec{a}} F(\vec{a}) &= \nabla_{\vec{a}} (\mathbf{X}\vec{a} - \vec{y})^\top (\mathbf{X}\vec{a} - \vec{y}) \\ &= 2\mathbf{X}^\top \mathbf{X}\vec{a} - 2\mathbf{X}^\top \vec{y} \stackrel{!}{=} \vec{0} \end{aligned}$$

As a consequence we get the system of **normal equations**:

$$\mathbf{X}^\top \mathbf{X}\vec{a} = \mathbf{X}^\top \vec{y}$$

This system has a solution if $\mathbf{X}^\top \mathbf{X}$ is not singular. Then we have

$$\vec{a} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \vec{y}.$$

$(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top$ is called the (Moore-Penrose-)**Pseudoinverse** of the matrix $\mathbf{X}$.

With the matrix-vector representation of the regression problem an extension to **multipolynomial regression** is straighforward:
Simply add the desired products of powers to the matrix $\mathbf{X}$.

Generalization to non-polynomial functions

$$\text{Simple example:} \qquad y = ax^b$$

Idea: Find transformation to linear/polynomial case.

$$\text{Transformation for example:} \qquad \ln y = \ln a + b \cdot \ln x.$$

Special case: **logistic function**

$$y = \frac{Y}{1 + e^{a+bx}} \qquad \Leftrightarrow \qquad \frac{1}{y} = \frac{1 + e^{a+bx}}{Y} \qquad \Leftrightarrow \qquad \frac{Y - y}{y} = e^{a+bx}.$$

Result: Apply so-called **Logit-Transformation**

$$\ln\left(\frac{Y - y}{y}\right) = a + bx.$$

Logistic Regression: Example

x	1	2	3	4	5
y	0.4	1.0	3.0	5.0	5.6

Transform the data with

$$z = \ln\left(\frac{Y - y}{y}\right), \qquad Y = 6.$$

The transformed data points are

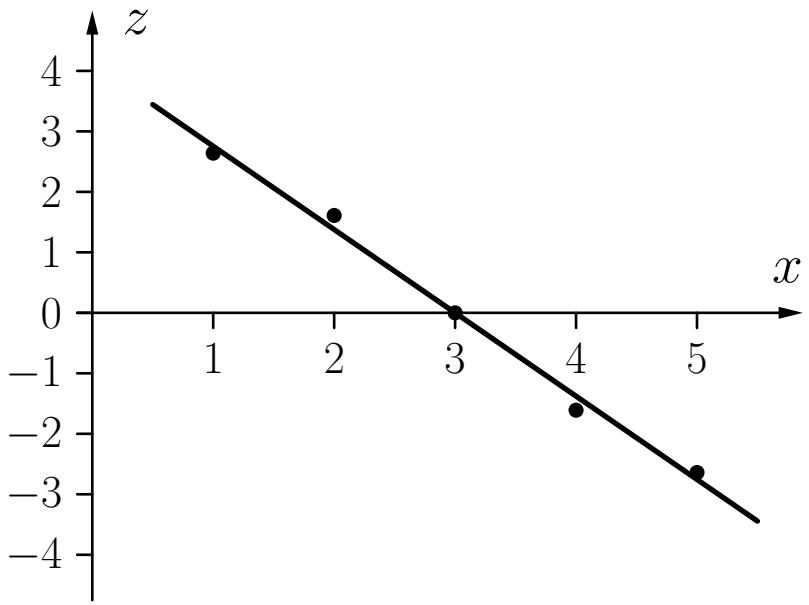
x	1	2	3	4	5
z	2.64	1.61	0.00	-1.61	-2.64

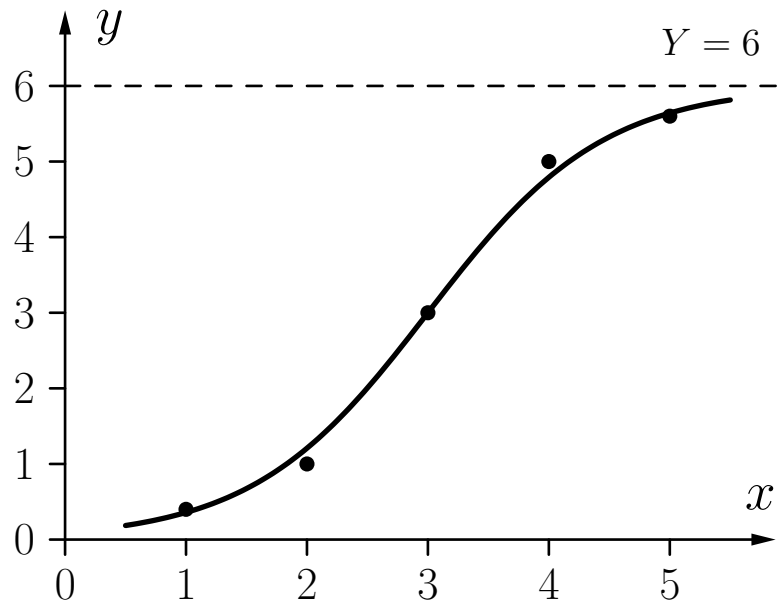
The resulting regression line is

$$z \approx -1.3775x + 4.133.$$

Thus the retransformation is $y = \frac{6}{1+e^{-1.3775x+4.133}}$.

Logistic Regression: Example





The logistic regression function can be computed by a single neuron with

- network input function $f_{\text{net}}(x) \equiv wx$ with $w \approx -1.3775$,
- activation function $f_{\text{act}}(\text{net}, \theta) \equiv (1 + e^{-(\text{net} - \theta)})^{-1}$ with $\theta \approx 4.133$ and
- output function $f_{\text{out}}(\text{act}) \equiv 6\,\text{act}$.