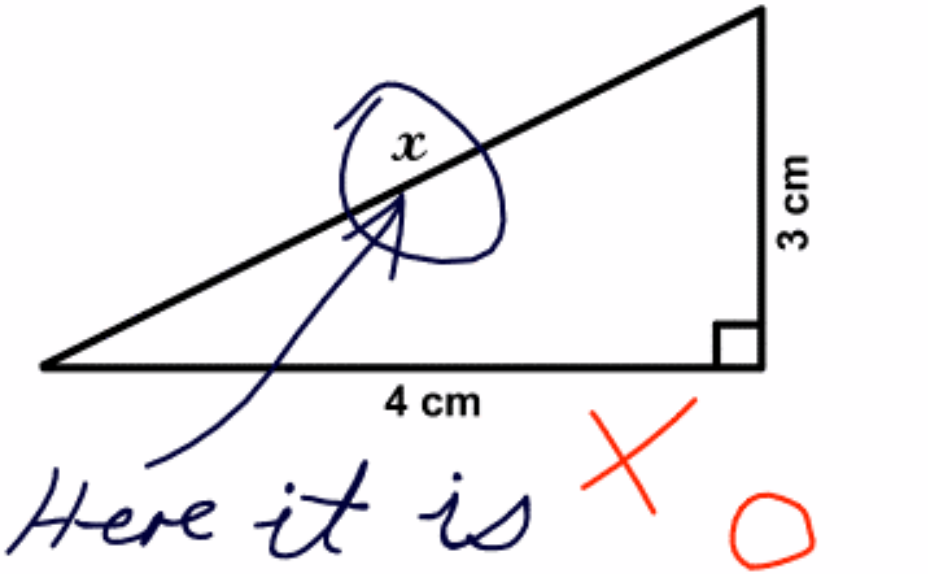
3. prednáška

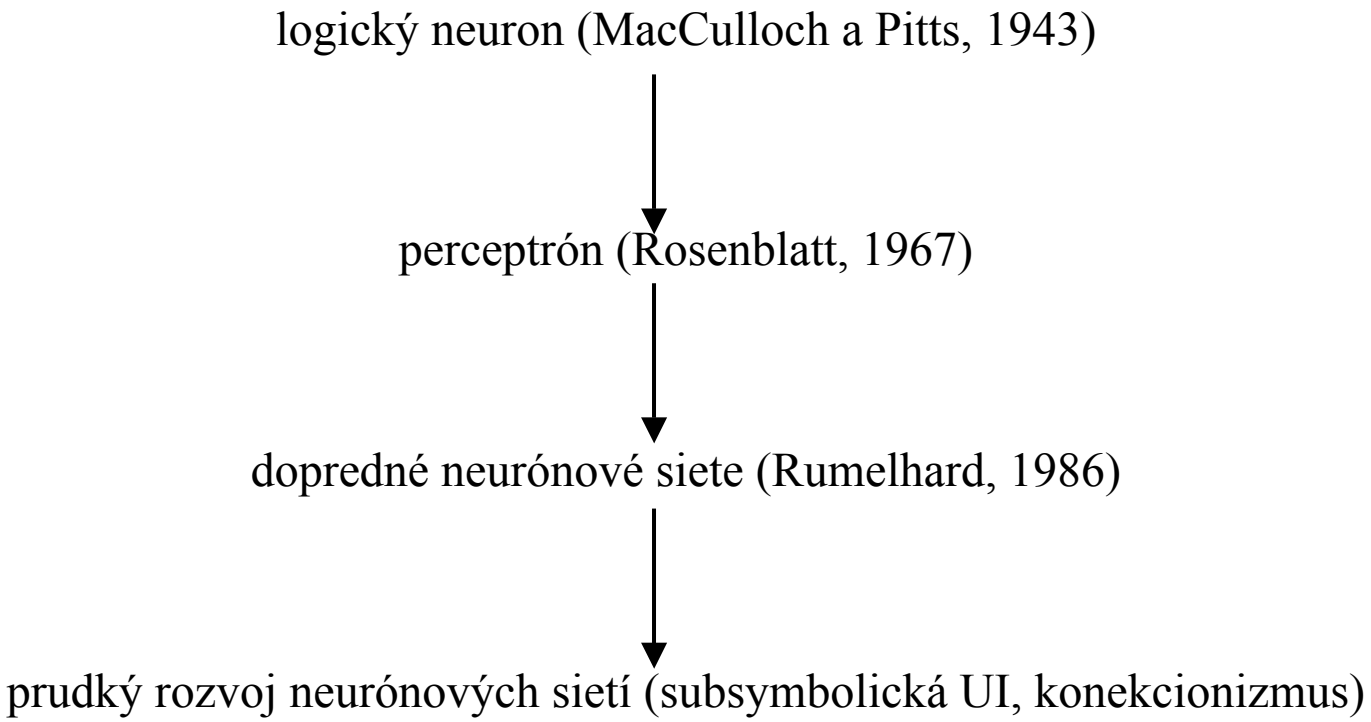
Dopredné neurónové siete

(multilayered perceptron)

3. Find x.



Dopredné neurónové siete (multilayer perceptron)

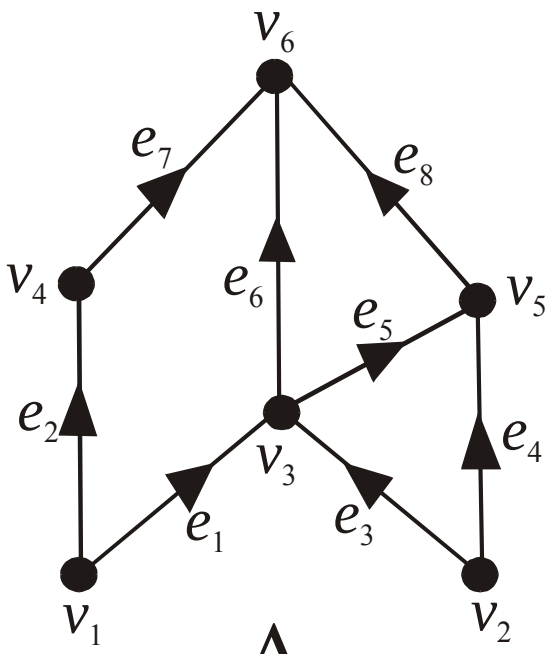


David Rumelhart

- D. E. Rumelhart, G. E. Hinton, and R. J. Williams: Learning representations by back-propagating errors. *Nature*, **323**(1986), 533-536.
- D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, editors: Parallel Distributed Processing, volumes I and II. MIT Press, Cambridge, MA, 1986.

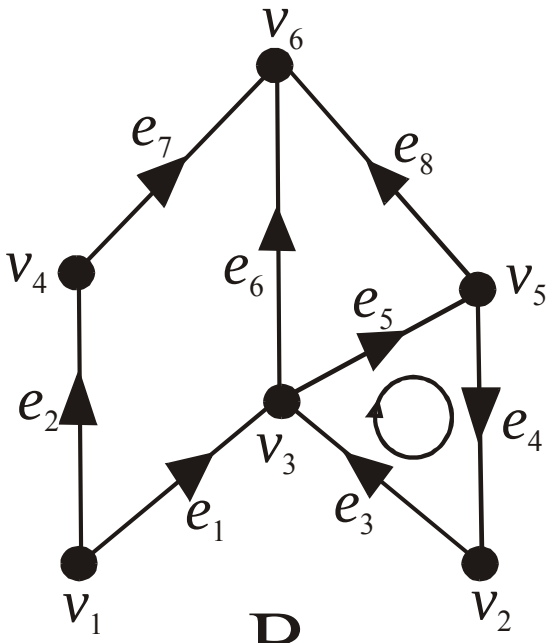
Formal definition of a multilayer perceptron

Let us consider an **oriented graph** $G=(V,E)$ determined as an ordered couple of a **vertex set** V and an **edge set** E.



A

acyclic oriented graph



B

cyclic oriented graph

Theorem. An oriented graph G is acyclic iff it may be indexed so that

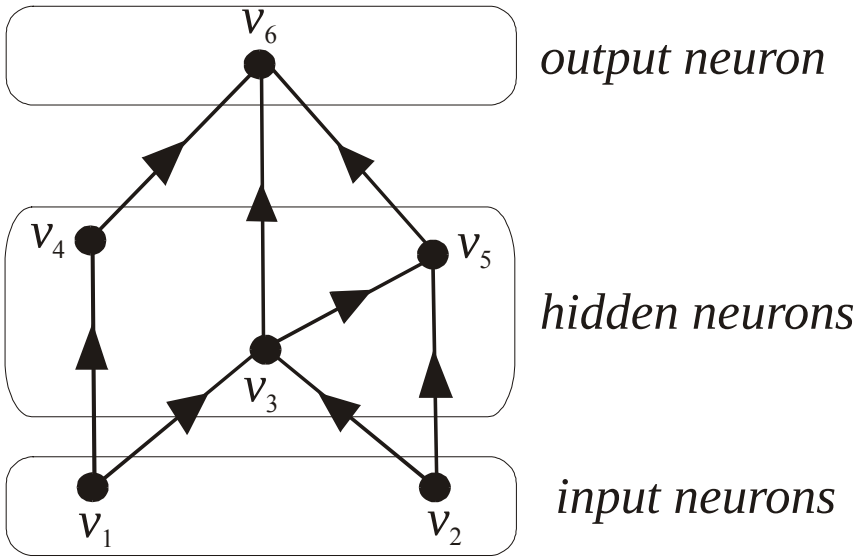
$$\forall e = (i, j) \in E : i < j$$

Vertices of an oriented graph can be classified as follows:

(1) **Input vertices** that are incident only with outgoing edges.

(2) **Hidden vertices** that are simultaneously incident at least with one incoming edge and at least with one outgoing edge.

(3) **Output vertices**, that are incident only with outgoing edges.



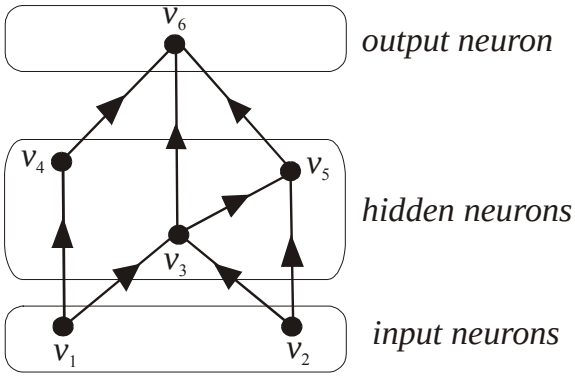
The vertex set V is unambiguously divided into three disjoint subsets

$$V = V_I \cup V_H \cup V_O$$

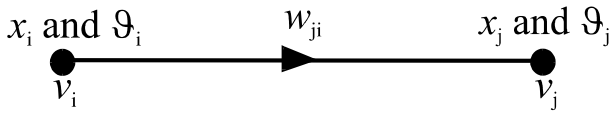
that are composed of input, hidden, and output vertices, respectively.

Assuming that the graph G is canonically indexed and that the graph is composed of n input vertices, h hidden vertices, and m output vertices, then the above three vertex subsets can be simply determined by

$$V_I = \{1,2,\ldots,n\}, \quad |V_I| = n$$
$$V_H = \{n+1, n+2,\ldots,n+h\}, \quad |V_H| = h$$
$$V_O = \{n+h+1, n+h+2,\ldots,n+h+m\}, \quad |V_O| = m$$



Vertices and edges of the oriented graph G are evaluated by real numbers. Each oriented edge $e=(v_i,v_j)\in E$ is evaluated by a real number w_{ji} called the **weight**. Each vertex $v_i\in V$ is evaluated by a real number x_i called the **activity** and each hidden or output vertex $v_i\in V$ is evaluated by a real number ϑ_i called the **threshold.**



The **activities** of hidden and output vertices are determined as follows

$$x_i = t(\xi_i)$$

$$\xi_i = \sum_{j\in\Gamma^{-1}(i)} w_{ij}x_j + \vartheta_i$$

where $t(\xi)$ is a **transfer function.** In our forthcoming considerations we will assume that its analytical form is specified by as the **sigmoid function**.

Definition. A **feed-forward neural network** is determined as an ordered triple

$$\mathcal{N}=(G,\boldsymbol{w},\vartheta)$$

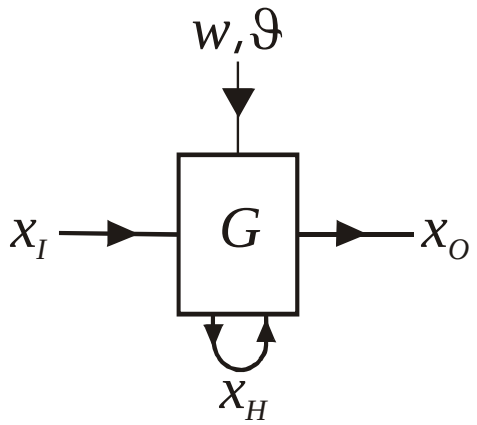
G is an acyclic, oriented, and connected graph and $\boldsymbol{w}$ and ϑ are weights and thresholds assigned to edges (connections) and vertices (neurons) of the graph. We say that the graph G specifies a **topology** (or **architecture**) of the neural network $\mathcal{N}$, and that weights $\boldsymbol{w}$ and thresholds ϑ specify **parameters** of the neural network.

Activities of neurons form a vector $\boldsymbol{x}=(x_1,x_2,\ldots,x_p)$. This vector can be divided formally onto three subvectors that are composed of input, hidden, and output activities

$$\boldsymbol{x} = \boldsymbol{x}_I \oplus \boldsymbol{x}_H \oplus \boldsymbol{x}_O$$

A neural network $\mathcal{N}=(G,\boldsymbol{w},\vartheta)$ with fixed weights and thresholds can be considered as a parametric function

$$G(\boldsymbol{w},\vartheta):R^n \to (0,1)^m$$



This function assigns to an input activity vector $\boldsymbol{x}_\mathrm{I}$ a vector of output activities $\boldsymbol{x}_\mathrm{O}$

$$\boldsymbol{x}_O = G(\boldsymbol{x}_I;\boldsymbol{w},\vartheta)$$

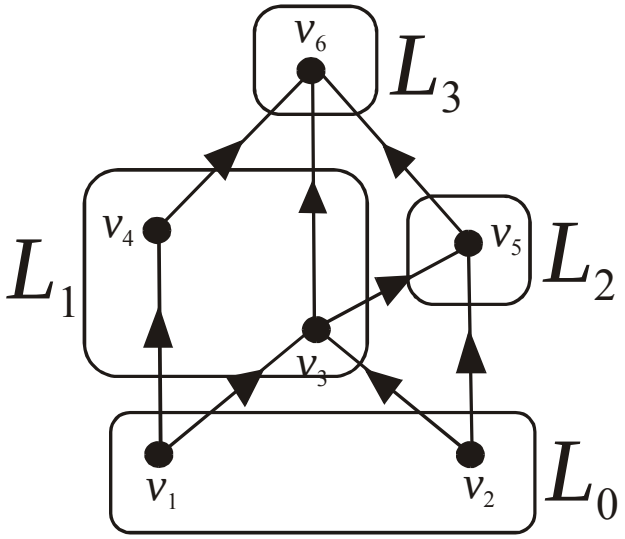
Hidden activities in this expression are not explicitly presented, they play only a role of byproducts.

How to calculate activities of a neural network $\mathcal{N}=(G,\boldsymbol{w},\vartheta)$?

(1) We shall postulate that activities of input neurons are kept fixed, in other words, we say that the vector of input activities $\boldsymbol{x}_{\mathrm{I}}$ is an inputs to the neural network. Usually its entries correspond to the so-called **descriptors** that specify a classified pattern.

(2) Activities of hidden and output neurons are calculated by simple recurrent procedure based on the fact that **the topology of neural network is determined by an acyclic oriented graph $\boldsymbol{G}$** .

$$x_i = f_i(x_1, x_2, \ldots, x_{i-1}) \quad (\text{for } i = n+1, n+2, \ldots, p)$$



Layer L_0:
x_1=*input constant*
x_2=*input constant*

Layer L_1:
$x_3=t(w_{31}x_1+w_{32}x_2+\vartheta_3)$
$x_4=t(w_{41}x_1+\vartheta_4)$

Layer L_2:
$x_5=t(w_{53}x_3+w_{52}x_2+\vartheta_5)$

Layer L_3:
$x_6=t(w_{63}x_3+w_{64}x_4+w_{65}x_5+\vartheta_5)$

Unfortunately, if the graph G **is not acyclic** (it contains oriented cycles), then the above simple recurrent calculation of activities is inapplicable, in some stage of algorithm we have to know an activity which was not yet calculated, activities are determined by

$$x_i = f_i\left(x_1, x_2, \ldots, x_n, x_{n+1}, \ldots, x_p\right) \quad (\text{for } i = n+1, n+2, \ldots, p)$$

These equations **are coupled**, their solution (if any) can be constructed by making use of an iterative procedure, which is started from initial activities, and calculated activities are used as an input for calculation of new activities, etc.

$$x_i^{(t+1)} = f_i\left(x_1, x_2, \ldots, x_n, x_{n+1}^{(t)}, \ldots, x_p^{(t)}\right) \quad (\text{for } i = n+1, n+2, \ldots, p)$$

This iterative procedure is repeated until a difference between new and old activities is smaller than a prescribed precision.

Learning (adaptation process) of feed-forward neural networks

A learning of a feed-forward neural network consists in a looking for such weights and thresholds that produced output activities (as a response on input activities) are closely related to the required ones.

First of all what we have to define is the training set composed of examples of input-activity vector and required output-activity vector

$$\mathcal{A}=\left\{\mathcal{A}_k=\left(\boldsymbol{x}_I^k,\hat{\boldsymbol{x}}_O^k\right);k=1,2,...,p\right\}$$

The error objective function is determined by

$$E_k(\boldsymbol{w},\vartheta)=\frac{1}{2}\left(\boldsymbol{x}_O^k-\hat{\boldsymbol{x}}_O^k\right)^2=\frac{1}{2}\left(G\left(\boldsymbol{x}_I^k;\boldsymbol{w},\vartheta\right)-\hat{\boldsymbol{x}}_O^k\right)^2$$

$$=\frac{1}{2}\sum_{i=1}^{p}g_i^2$$

$$g_i=\begin{cases}x_i-\hat{x}_i & \left(\text{for } i\in V_O\right)\\ 0 & (\text{otherwise})\end{cases}$$

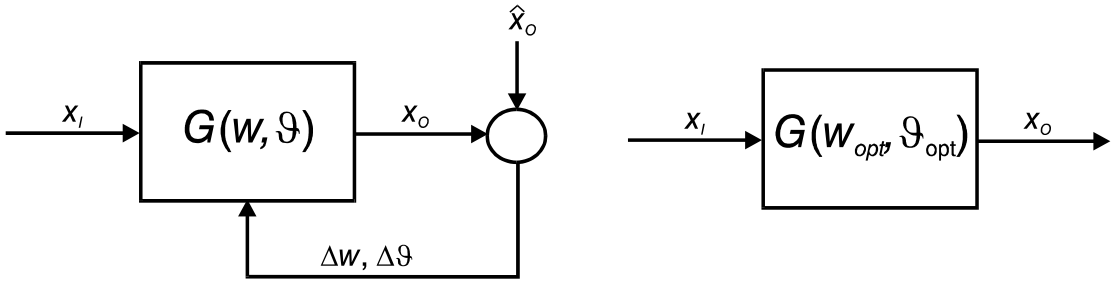
$$E(\boldsymbol{w},\vartheta)=\sum_{k=1}^{P}E_k(\boldsymbol{w},\vartheta)$$

The learning process of the given feed-forward neural network $\mathcal{NN}(G,\bullet,\bullet)$ specified by the graph G and with unknown weights and thresholds is realized by the following minimization process

$$\left(\boldsymbol{w}_{opt},\vartheta_{opt}\right)=\arg\min_{(\boldsymbol{w},\vartheta)} E(\boldsymbol{w},\vartheta)$$

If we know the optimal weights and thresholds (that minimize the error objective function), then an **active process** is a calculation of output activities as a response on input activities for parameters determined by the learning - adaptation process

$$\boldsymbol{x}_O = G\left(\boldsymbol{x}_I;\boldsymbol{w}_{opt},\vartheta_{opt}\right)$$



learning process *active process*

The active process of neural networks is usually used **for a classification or prediction** of unknown patterns that are described only by input activities (descriptors that specify a structure of patterns). In order to quantify this process we have to introduce the so-called **test set** composed of examples of patterns specified by an input-activity vector and a required output-activity vector

$$\mathcal{A}_{test} = \left\{ \mathcal{A}_k = \left(\boldsymbol{y}_I^k , \hat{\boldsymbol{y}}_O^k \right); k = 1,2,...,q \right\}$$

An analogue of the error objective function is

$$E_k^{(test)} \left(\boldsymbol{w}_{opt} , \vartheta_{opt} \right) = \frac{1}{2} \left(\boldsymbol{y}_O^k - \hat{\boldsymbol{y}}_O^k \right)^2 = \frac{1}{2} \left(G \left(\boldsymbol{y}_I^k ; \boldsymbol{w}_{opt} , \vartheta_{opt} \right) - \hat{\boldsymbol{y}}_O^k \right)^2$$

$$E^{(test)} \left(\boldsymbol{w}_{opt} , \vartheta_{opt} \right) = \sum_{k=1}^{q} E_k^{(test)} \left(\boldsymbol{w}_{opt} , \vartheta_{opt} \right)$$

We say that an adapted neural network **correctly interprets** patterns from the test set if **this objective function is sufficiently small**. If this requirement is not fulfilled, then there exists an example from the test set which incorrectly interpreted.

Gradient of error objective function

Partial derivatives of the error objective function (defined over a training set) with respect to weights and thresholds are determined by

$$\frac{\partial E_k}{\partial w_{ij}} = \frac{\partial E_k}{\partial x_i}\frac{\partial x_i}{\partial w_{ij}} = \frac{\partial E_k}{\partial x_i} t'\left(\xi_i\right) x_j$$

$$\frac{\partial E_k}{\partial \vartheta_i} = \frac{\partial E_k}{\partial x_i}\frac{\partial x_i}{\partial \vartheta_i} = \frac{\partial E_k}{\partial x_i} t'\left(\xi_i\right)$$

where the partial derivatives $\partial x_i / \partial w_{ij}$ is simply calculated by making use the relation $x_i = t(\xi_i)$, then $\partial x_i / \partial w_{ij} = t'(\xi_i) \cdot \partial \xi_i / \partial w_{ij} = t'(\xi_i) x_j$. Similar considerations are applicable also for the partial derivative $\partial x_i / \partial \vartheta_i$. If we compare both these equations we get simple relationship between partial derivatives $\partial E_k / \partial w_{ij}$ and $\partial E_k / \partial \vartheta_i$

$$\frac{\partial E_k}{\partial w_{ij}} = \frac{\partial E_k}{\partial \vartheta_i} x_j$$

Let us study the partial derivative $\partial E_k / \partial x_i$, its calculation depends on whether the index i corresponds to an output neuron or hidden neuron

$$\frac{\partial E_k}{\partial x_i} = g_i \qquad \left(\text{for } i \in V_O\right)$$

$$\frac{\partial E_k}{\partial x_i} = \sum_l \frac{\partial E_k}{\partial x_l} \frac{\partial x_l}{\partial x_i} \qquad \left(\text{for } i \in V_H\right)$$

where summation runs over all neurons that are successors of the i-the neuron. The second above formula results as an application of the well-known theorem called the **chain rule** for calculation of partial derivatives of the composite function. Last expressions can be unified at one expression

$$\frac{\partial E_k}{\partial x_i} = g_i + \sum_l \frac{\partial E_k}{\partial x_l} \frac{\partial x_l}{\partial x_i}$$

where it is necessary to remember that the summation on r.h.s. is vanishing is the i-th neuron has not successors, that is output neurons.

A final expression for the partial derivatives of error objective function with respect to thresholds

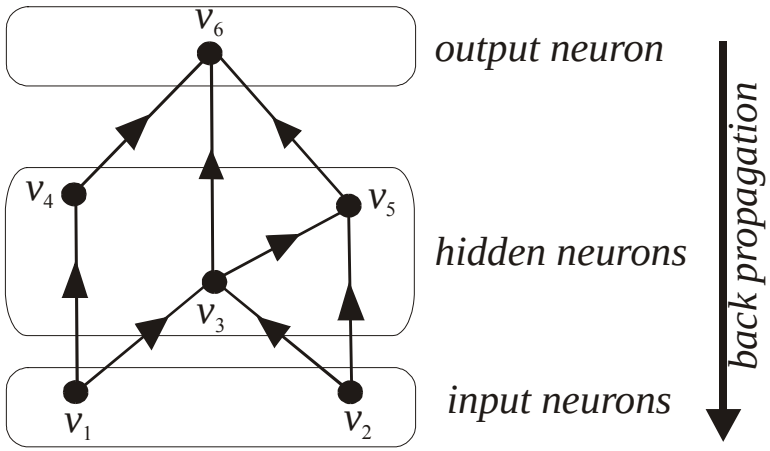
$$\frac{\partial E_k}{\partial \vartheta_i} = t'(\xi_i)\left(g_i + \sum_l \frac{\partial E_k}{\partial \vartheta_l} w_{li} \right) \qquad (\text{for } i \in V_H \cup V_O)$$

(1) In general, the above formula for calculation of partial derivative of the objective function can be characterized as a system of linear equations the solution of which determines partial derivatives $\partial E_k / \partial \vartheta_i$.

(2) For feed-forward neural networks the above formula can be solved recurrently. Starting from top layer L_t we calculate all partial derivatives assigned to output neurons, $\partial E_k / \partial \vartheta_i = t'(\xi_i) g_i$. In the next step we calculate partial derivatives from the layer L_{t-1}, for their calculation we need to know partial derivatives from the layer L_t , which were calculated in the previous step, etc. Therefore this recurrent approach of calculation of partial derivatives based is called the **back propagation.**

(3) Knowing all partial derivatives $\partial E_k / \partial \vartheta_i$, we may calculate simply partial derivatives $\partial E_k / \partial w_{ij}$.

(4) An analogue of the above formula was initially derived by Rumelhart et al. in 1986, this work is considered in literature as one **of milestones of the development of theory of neural network**. They demonstrated that multilayer perceptrons together with the back propagation method for calculation of gradients of error objective functions are able to overcome boundaries of simple perceptrons stated by Minsky and Papert, that is to classify correctly all patterns and not only those ones that are linearly separable.



Theorem. For feed-forward neural networks the first partial derivatives of the error objective function E_k are determined recurrently by

$$\frac{\partial E_k}{\partial \vartheta_i} = t'(\xi_i)\left(g_i + \sum_l \frac{\partial E_k}{\partial \vartheta_l} w_{li} \right)$$

$$\frac{\partial E_k}{\partial w_{ij}} = \frac{\partial E_k}{\partial \vartheta_i} x_j$$

where in order to calculate $\partial E_k / \partial \vartheta_i$ we have to know either g_i (for $i \in V_O$) or partial derivatives $\partial E_k / \partial \vartheta_l$ (for $l \in \Gamma_i$) .

We have to emphasize that in the course of derivation of the above formula **we have never used an assumption that the considered neural network corresponds to an acyclic graph**. This means that this formula is correct also for neural networks assigned to graphs that are either cyclic or acyclic.

For **neural networks assigned to cyclic graphs** the above discussed recurrent approach of calculation of partial derivatives is inapplicable. For this type of neural networks the partial derivatives are not determined recurrently, but only as a solution of linear coupled equations

$$\frac{\partial E_k}{\partial \vartheta_i} - \sum_{l \in \Gamma(i)} \frac{\partial E_k}{\partial \vartheta_l} w_{li} x_i' = x_i' g_i$$

Its matrix form is

$$\left(\boldsymbol{1} - \mathrm{diag}(\boldsymbol{x}')\boldsymbol{w}^T\right)\boldsymbol{e}_k' = \boldsymbol{a}$$

where $\boldsymbol{e}_k' = \left(\partial E_k / \partial \vartheta_1, \ldots, \partial E_k / \partial \vartheta_p\right)^T$ is a column vector composed of first derivatives of the error objective function E_k with respect to thresholds, $\mathrm{diag}(\boldsymbol{x}') = \left(\delta_{ij} x_i'\right)$ is a diagonal matrix composed of first derivatives of activities, and $\boldsymbol{a} = \left(g_1 x_1', \ldots, g_p x_p'\right)^T$ is a column vector composed of products $g_i x_i'$. Assuming that the matrix $\left(1 - \mathrm{diag}(\boldsymbol{x}')\boldsymbol{w}^T\right)$ is **nonsingular**, then the solution is determined by

$$\boldsymbol{e}' = \left(\boldsymbol{1} - \mathrm{diag}(\boldsymbol{x}')\boldsymbol{w}^T\right)^{-1} \boldsymbol{a}$$

This means, for neural networks represented either by cyclic or acyclic oriented graph, first partial derivatives of E_k are determined as a solution of the above matrix equation.

The present approach is simply generalized also for calculation of partial derivatives of the total error objective function determined over all training patterns

$$\frac{\partial E}{\partial w_{ij}} = \sum_{k=1}^{P} \frac{\partial E_k}{\partial w_{ij}} \quad \text{and} \quad \frac{\partial E}{\partial \vartheta_i} = \sum_{k=1}^{P} \frac{\partial E_k}{\partial \vartheta_i}$$

If we know the partial derivatives, then the **batch adaptation** process is simply realized by a **steepest-descent optimization accelerated by a momentum term**

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \lambda \frac{\partial E}{\partial w_{ij}} + \mu \Delta w_{ij}^{(t)}$$

$$\vartheta_i^{(t+1)} = \vartheta_i^{(t)} - \lambda \frac{\partial E}{\partial \vartheta_i} + \mu \Delta \vartheta_i^{(t)}$$

where the learning parameter $\lambda>0$ should be sufficiently small (usually $\lambda=0.01$-0.1) to ensure a monotonous convergence of the optimization method. Initial values of weights $w_{ij}^{(0)}$ and thresholds $\vartheta_i^{(0)}$ are randomly generated . The last terms in the above formulae correspond to **momentum terms** determined as a difference of terms from the last two iterations, $\Delta w_{ij}^{(t)} = w_{ij}^{(t)} - w_{ij}^{(t-1)}$ and $\Delta \vartheta_i^{(t)} = \vartheta_i^{(t)} - \vartheta_i^{(t-1)}$. The momentum terms may be important at the initial stage of optimization as a simple tool how to escape local minima, the value of the momentum parameter is usually $0.5 \le \mu \le 0.7$.

Hessian of error objective function

One of the most efficient optimization techniques is the Newton optimization method based on the following recurrent updating formula

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - H^{-1}(\boldsymbol{x}_k)\,\mathrm{grad}\ f(\boldsymbol{x}_k)$$

where $H(\boldsymbol{x}_k)$ is a Hessian matrix composed of partial derivatives of second order and $f(\boldsymbol{x})$ is an objective function to be minimized. This recurrent scheme is stopped if a norm of gradient is smaller than a prescribed precision, the obtained solution $\boldsymbol{x}^*$ is a minimum for a positive definite Hessian $H(\boldsymbol{x}^*)$.

Three types of partial derivatives of second order will be calculated

$$\frac{\partial^2 E_k}{\partial \vartheta_a \partial \vartheta_i}, \quad \frac{\partial^2 E_k}{\partial \vartheta_i \partial w_{ab}}, \quad \frac{\partial^2 E_k}{\partial w_{ij} \partial w_{ab}}$$

The partial derivatives $\partial^2 E_k / \partial \vartheta_i \partial \vartheta_a$ (where $i,a \in V_H \cup V_O$) can be calculated as follows

$$\frac{\partial^2 E_k}{\partial \vartheta_a \partial \vartheta_i} = \frac{\partial}{\partial \vartheta_a}\left(\frac{\partial E_k}{\partial \vartheta_i}\right) = \frac{\partial}{\partial \vartheta_a}\left[t'(\xi_i)\left(g_i + \sum_l \frac{\partial E_k}{\partial \vartheta_l} w_{li} \right)\right]$$

$$= x_i'' \delta_{ia}\left(g_i + \sum_l \frac{\partial E_k}{\partial \vartheta_l} w_{li} \right) + x_i'\left(\delta_{ia}\delta_i(O) x_i' + \sum_l \frac{\partial^2 E_k}{\partial \vartheta_a \partial \vartheta_l} w_{li} \right)$$

where

$$\delta_{ia} = \begin{cases} 1 & (\text{if } i = a) \\ 0 & (\text{if } i \neq a) \end{cases} \qquad \text{and} \qquad \delta_i(O) = \begin{cases} 1 & (\text{if } i \in V_O) \\ 0 & (\text{if } i \notin V_O) \end{cases}$$

Symbols x_i' and x_i'' are first and second derivative of activities assigned to hidden or output neurons.

This formula allows **a recurrent "back propagation"** calculation of second partial derivatives $\partial^2 E_k / \partial \vartheta_a \partial \vartheta_i$.

The partial derivatives $\partial^2 E_k / \partial w_{ab} \partial \vartheta_i$ are calculated immediately from the above two formulae, we get

$$\frac{\partial^2 E_k}{\partial w_{ab} \partial \vartheta_i} = \frac{\partial}{\partial \vartheta_i}\left(\frac{\partial E_k}{\partial w_{ab}}\right) = \frac{\partial}{\partial \vartheta_i}\left(\frac{\partial E_k}{\partial \vartheta_a} x_b\right)$$

$$= \frac{\partial^2 E_k}{\partial \vartheta_i \partial \vartheta_a} x_b + \delta_{ib} x_b' \frac{\partial E_k}{\partial \vartheta_a}$$

In a similar way we may calculate partial derivatives $\partial^2 E_k / \partial w_{ab} \partial w_{ij}$, we get

$$\frac{\partial^2 E_k}{\partial w_{ab} \partial w_{ij}} = \frac{\partial}{\partial w_{ab}}\left(\frac{\partial E_k}{\partial w_{ij}}\right) = \frac{\partial}{\partial w_{ab}}\left(\frac{\partial E_k}{\partial \vartheta_i} x_j\right)$$

$$= \frac{\partial}{\partial w_{ab}}\left(\frac{\partial E_k}{\partial \vartheta_i}\right) x_j + \frac{\partial E_k}{\partial \vartheta_i}\frac{\partial x_j}{\partial w_{ab}} = \frac{\partial}{\partial \vartheta_i}\left(\frac{\partial E_k}{\partial w_{ab}}\right) x_j + \frac{\partial E_k}{\partial \vartheta_i} t'\left(\xi_j\right) \delta_{ja} x_b$$

$$= \frac{\partial}{\partial \vartheta_i}\left(\frac{\partial E_k}{\partial \vartheta_a} x_b\right) x_j + \frac{\partial E_k}{\partial \vartheta_i} t'\left(\xi_j\right) \delta_{ja} x_b = \frac{\partial^2 E_k}{\partial \vartheta_i \partial \vartheta_a} x_b x_j + \frac{\partial E_k}{\partial \vartheta_a}\frac{\partial x_b}{\partial \vartheta_i} x_j + \frac{\partial E_k}{\partial \vartheta_i} t'\left(\xi_j\right) \delta_{ja} x_b$$

$$= \frac{\partial^2 E_k}{\partial \vartheta_i \partial \vartheta_a} x_b x_j + \delta_{ib} x_b' \frac{\partial E_k}{\partial \vartheta_a} x_j + \delta_{ja} x_j' \frac{\partial E_k}{\partial \vartheta_i} x_b$$

A calculation of partial derivatives $\partial^2 E_k / \partial w_{ab} \partial \vartheta_i$ and $\partial^2 E_k / \partial w_{ab} \partial w_{ij}$ requires only first partial derivatives $\partial E_k / \partial \vartheta_i$ and second partial derivatives $\partial^2 E_k / \partial \vartheta_i \partial \vartheta_a$.

Theorem. For feed-forward neural networks the second partial derivatives of the error objective function E_k are determined by

$$\frac{\partial^2 E_k}{\partial \vartheta_a \partial \vartheta_i} = \delta_{ia}\left(g_i + \sum_l \frac{\partial E_k}{\partial \vartheta_l} w_{li} \right) x_i'' + \left(\delta_{ia}\delta_i(O) x_i' + \sum_l \frac{\partial^2 E_k}{\partial \vartheta_a \partial \vartheta_l} w_{li} \right) x_i'$$

$$\frac{\partial^2 E_k}{\partial w_{ab} \partial \vartheta_i} = \frac{\partial^2 E_k}{\partial \vartheta_i \partial \vartheta_a} x_b + \delta_{ib} \frac{\partial E_k}{\partial \vartheta_a} x_b'$$

$$\frac{\partial^2 E_k}{\partial w_{ab} \partial w_{ij}} = \frac{\partial^2 E_k}{\partial \vartheta_i \partial \vartheta_a} x_b x_j + \delta_{ib} \frac{\partial E_k}{\partial \vartheta_a} x_b' x_j + \delta_{ja} \frac{\partial E_k}{\partial \vartheta_i} x_b x_j'$$

where partial derivatives $\partial^2 E_k / \partial \vartheta_i \partial \vartheta_a$ may be calculated recurrently in a back propagation manner whereas other two partial derivatives $\partial^2 E_k / \partial w_{ab} \partial \vartheta_i$ and $\partial^2 E_k / \partial w_{ab} \partial w_{ij}$ are calculated directly.

Partial derivatives of second order of the total objective function determined over all patterns from the training set are determined as summations of partial derivatives of objective functions E_k

$$\frac{\partial^2 E}{\partial \vartheta_a \partial \vartheta_i} = \sum_{k=1}^{p} \frac{\partial^2 E_k}{\partial \vartheta_a \partial \vartheta_i}$$

$$\frac{\partial^2 E}{\partial \vartheta_i \partial w_{ab}} = \sum_{k=1}^{p} \frac{\partial^2 E_k}{\partial \vartheta_i \partial w_{ab}}$$

$$\frac{\partial^2 E}{\partial w_{ij} \partial w_{ab}} = \sum_{k=1}^{p} \frac{\partial^2 E_k}{\partial w_{ij} \partial w_{ab}}$$

An adaptation process of neural networks realized in the framework of Newton optimization method is based on the following recurrent updating formulae

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \sum_{(ab)} {}^{(t)}H^{-1}_{(ij)(ab)} \frac{\partial E^{(t)}}{\partial w_{ab}} - \sum_{(a)} {}^{(t)}H^{-1}_{(ij)(a)} \frac{\partial E^{(t)}}{\partial \vartheta_a}$$

$$\vartheta_i^{(t+1)} = \vartheta_i^{(t)} - \sum_{(ab)} {}^{(t)}H^{-1}_{(i)(ab)} \frac{\partial E^{(t)}}{\partial w_{ab}} - \sum_{(a)} {}^{(t)}H^{-1}_{(i)(a)} \frac{\partial E^{(t)}}{\partial \vartheta_a}$$

In a similar way as in the previous part of this lecture, where we have studied first partial derivatives of the error objective function E_k, the second partial derivatives were derived so that **we did not use any assumption that neural networks correspond to acyclic graphs**. Of course, an assumption that neural networks are acyclic considerably simplifies the calculation of second partial derivatives $\partial^2 E_k / \partial\vartheta_i \partial\vartheta_a$ by a recurrent method. For cyclic neural networks this recurrent approach is inapplicable, the partial derivatives are determined by a system of linear equations

$$\frac{\partial^2 E_k}{\partial\vartheta_a \partial\vartheta_i} - \sum_l \frac{\partial^2 E_k}{\partial\vartheta_a \partial\vartheta_l} w_{li} x_i' = \delta_{ia} x_i'' \left(g_i + \sum_l \frac{\partial E_k}{\partial\vartheta_l} w_{li} \right) + \delta_{ia} \delta_i \left(O \right) x_i'^2$$

This relation may be rewritten in a matrix form as follows

$$\boldsymbol{E''}\left(\boldsymbol{1} - \boldsymbol{w}\,\mathrm{diag}(\boldsymbol{x'})\right) = \boldsymbol{A}$$

where $\boldsymbol{E}''$ is a symmetric matrix composed of partial derivatives $\partial^2 E_k / \partial\vartheta_i \partial\vartheta_a$ and $\boldsymbol{A}=(A_{ai})$ is a diagonal matrix. Assuming that the matrix $(\boldsymbol{1}\text{-}\boldsymbol{w}\mathrm{diag}(\boldsymbol{x}'))$ is a nonsingular, then the matrix $\boldsymbol{E''}$ composed of second partial derivatives $\partial^2 E_k / \partial\vartheta_i \partial\vartheta_a$ is determined explicitly by

$$\boldsymbol{E''} = \boldsymbol{A}\left(\boldsymbol{1} - \boldsymbol{w}\,\mathrm{diag}(\boldsymbol{x'})\right)^{-1}$$

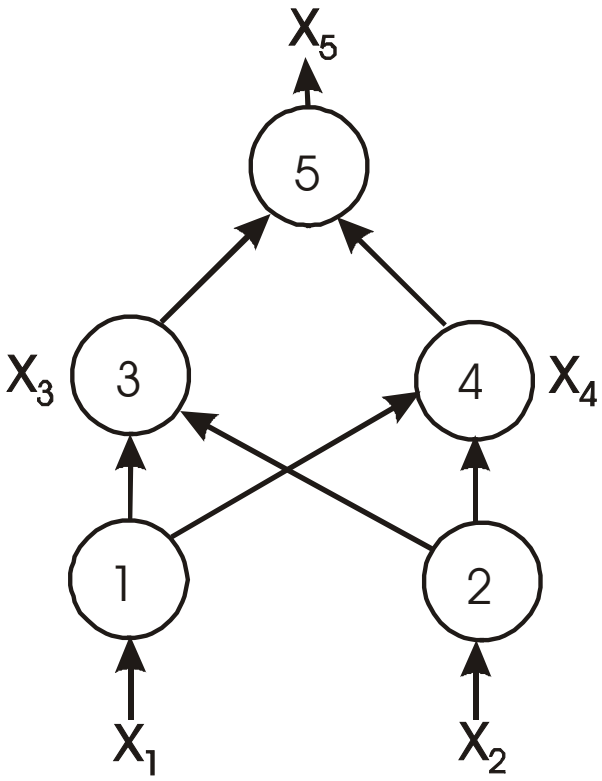
Illustrative example
Boolean function XOR

An effectiveness of neural networks with hidden neurons is illustrated by Boolean function XOR, which is not correctly classified by simple perceptron without hidden neurons.

The used neural networks will contain three layers, the first layer is composed of input neuron, the second one of hidden neurons, and the third (last) one of output neurons.

Hidden neurons create the so-called **inner representation** which is already linearly separable, this is the main reason why neural networks with hidden neurons are most frequently used in the whole theory of neural networks.

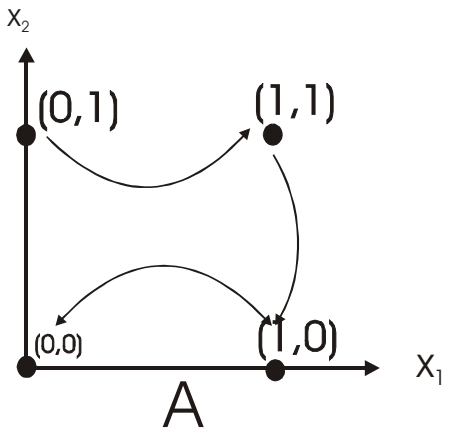
Parameters of adaptation process are: λ=0,1, μ=0,5; after 400 iterations the objective function achieved value E=0.031.

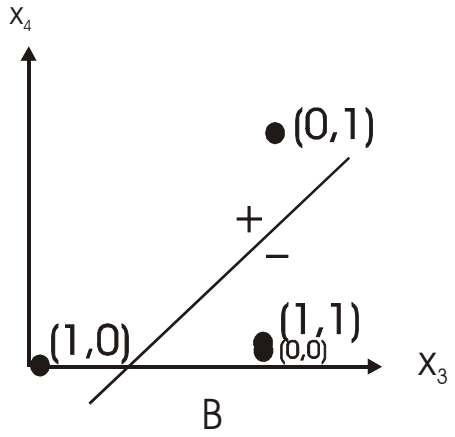


Activities of neurons from the feed-forward network composed of five neurons and two hidden neurons.

No.	x_1	x_2	x_3	x_4	x_5	$\hat{x}_5$
1	0,00	0,00	0,96	0,08	0,06	0,00
2	0,00	1,00	1,00	0,89	0,95	1,00
3	1,00	0,00	0,06	0,00	0,94	1,00
4	1,00	1,00	0,96	0,07	0,05	0,00

Hidden activities of XOR problem are **linearly separable.**





Newtonova metóda

Budeme približne riešiť rovnicu

$$\text{grad } f(\boldsymbol{x}) = \boldsymbol{0}$$

Nech $\boldsymbol{x}=\boldsymbol{x}_0+\boldsymbol{\delta}$, kde $\boldsymbol{\delta}=(\delta_1,\delta_2,...,\delta_n)$, potom

$$\text{grad } f(\boldsymbol{x}_o + \mathrm{d}) = \boldsymbol{0}$$

ľavá strana bude upravená pomocou Taylorovho rozvoja

$$\text{grad } f(\boldsymbol{x}_o) + H(\boldsymbol{x}_o)\mathrm{d} + ... = \boldsymbol{0}$$

Ak budeme uvažovať len prvé dva členy rozvoja a budeme predpokladať, že Hessián $H(\boldsymbol{x}_o)$ je nesingulárna matica, potom

$$\delta \approx -H^{-1}(\boldsymbol{x}_o)\,\text{grad } f(\boldsymbol{x}_o)$$

Poznámka: Výchylka $\boldsymbol{\delta}$ je formálne určená dvoma ekvivalentnými spôsobmi: buď ako riešenie systému lineárnych rovníc

$$H(\boldsymbol{x}_o)\delta \approx -\text{grad } f(\boldsymbol{x}_o)$$

alebo pomocou inverznej matice

$$\delta \approx -H^{-1}(\boldsymbol{x}_o)\,\text{grad } f(\boldsymbol{x}_o)$$

Riešenie rovnice grad $f(\boldsymbol{x}) = \boldsymbol{0}$ má potom tento približný tvar

$$\boldsymbol{x} \approx \boldsymbol{x}_o - H^{-1}(\boldsymbol{x}_o) \text{ grad } f(\boldsymbol{x}_o)$$

Toto riešenie slúži ako podklad pre konštrukciu rekurentnej formule, ktorá je základom Newtonovej optimalizačnej metódy

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - H^{-1}(\boldsymbol{x}_k) \text{ grad } f(\boldsymbol{x}_k)$$

pre k=0,1,2,... a $\boldsymbol{x}_o$ je zadaná počiatočné riešenie. Rekurentná aplikácia tejto formule je ukončená keď začne platiť $|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k| < \varepsilon$, kde $\varepsilon>0$ je zadaná presnosť požadovaného riešenia.

Algoritmus Newtonovej metódy

```
read(x_o, k_max, ε);
k:=0; norm:=∞; x:=x_o;
while (k<kmax) and (norm>ε) do
begin k:=k+1;
      solve SLE H(x)δ=-grad f(x);
      x':=x+δ;
      norm:=|x-x'|;
      x:=x';
end;
write(x,f(x));
```

Poznámky

(1) Newtonova metóda nájde riešenie rovnice grad $f(\boldsymbol{x}) = \boldsymbol{0}$, t.j. nájde stacionárne stavy funkcie $f(\boldsymbol{x})$. Tieto stacionárne stavy sú klasifikované pomocou Hessiánu $H(\boldsymbol{x})$:

> Ak $\boldsymbol{x}$ je stacionárny bod a Hessián $H(\boldsymbol{x})$ je pozitívne negatívny, potom v bode $\boldsymbol{x}$ má funkcia $f(\boldsymbol{x})$ minimum.

(2) Funkcia $f(\boldsymbol{x})$ musí byť dvakrát diferencovateľná na celom R^n, počítame gradient a Hessián funkcie.

(3) Výpočet Hessiánu môže byť časovo a pamäťovo veľmi náročný, menovite pre funkcie mnohých (niekoľko sto) premenných.

Linearizovaná Newtonova metóda

Pre určitý typ minimalizovanej funkcie výpočet Hessiánu môže byť podstatne zjednodušený.

$$f(\boldsymbol{x}) = \sum_{k=1}^{p} g_k^2(\boldsymbol{x})$$

Pre túto funkciu platí vlastnosť

$$f(\boldsymbol{x}^*) = 0 \quad \Leftrightarrow \quad \forall k \in [1,p]: g_k(x^*) = 0$$

Nutná a postačujúca podmienka pre existenciu takého $\boldsymbol{x}^*$, $f(\boldsymbol{x}^*)=0$, aby pre každé $k \in [1,p]$ platilo $g_k(\boldsymbol{x}^*)=0$.

Výpočet Hessiánu pre funkciu $f(\boldsymbol{x}) = \sum_{k=1}^{p} g_k^2(\boldsymbol{x})$

$$\frac{\partial f(\boldsymbol{x})}{\partial x_i} = 2\sum_{k=1}^{p} g_k(\boldsymbol{x}) \frac{\partial g_k(\boldsymbol{x})}{\partial x_i}$$

$$\frac{\partial^2 f(\boldsymbol{x})}{\partial x_i \partial x_j} = 2\sum_{k=1}^{p} \frac{\partial g_k(\boldsymbol{x})}{\partial x_i} \frac{\partial g_k(\boldsymbol{x})}{\partial x_j} + \cancel{2\sum_{k=1}^{p} g_k(\boldsymbol{x}) \frac{\partial^2 g_k(\boldsymbol{x})}{\partial x_i \partial x_j}}$$

Budeme predpokladať, že vektor $\boldsymbol{x}$ je blízko $\boldsymbol{x}^*$, potom $g_k(\boldsymbol{x}) \approx 0$, pre k=1,2,..., potom približný výraz pre elementy Hessiánu má tento tvar

$$H_{ij}(\boldsymbol{x}) \approx \frac{\partial^2 f(\boldsymbol{x})}{\partial x_i \partial x_j} \approx 2\sum_{k=1}^{p} \frac{\partial g_k(\boldsymbol{x})}{\partial x_i} \frac{\partial g_k(\boldsymbol{x})}{\partial x_j}$$

V maticovom formalizme tento výraz má tvar

$$H(\boldsymbol{x}) \approx 2\sum_{k=1}^{p} \left(grad\ g_k(\boldsymbol{x})\right)^T \left(grad\ g_k(\boldsymbol{x})\right)$$