2. prednáška

Perceptrón

1. Uvodné poznámky

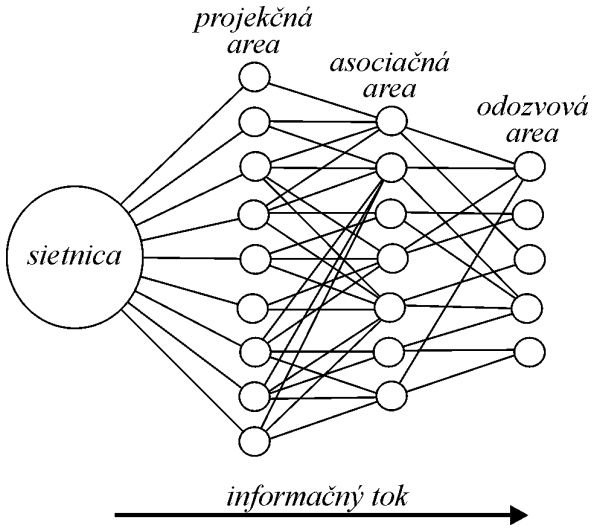
- ***Logický neurón McCullocha a Pittsa nie je schopný učenia***, jeho parametre (váhové a prahové koeficienty) sú pevne nastavené tak, aby neurón vykonával požadovanú Boolovu funkciu (logickú spojku alebo konjuktívnu klauzulu). Podobne, neurónové siete zostrojené z týchto neurónov sú navrhnuté tak, aby taktiež vykonávali Boolovu funkciu všeobecného tvaru. Konštrukcia neurónovej siete má „stavebnicový charakter“, používame vopred dané neuróny s požadovanými vlastnosťami, nešurónové siete sa neučia (neadaptujú) tak, aby vykonávali požadovanú Boolovu funkciu.

- Až zásluhou ***Franka Rosemblatta bolo zahrnuté učenie*** do konštrkcie neurónu typu McCullocha a Pittsa, váhové koeficienty a prahové koeficienty boli pokladané za premenné parametre „modelu“, ktoré sa nastavia procesom učenia.

Frank Rosenblatt (1928 - 1969)

priesvitka 3

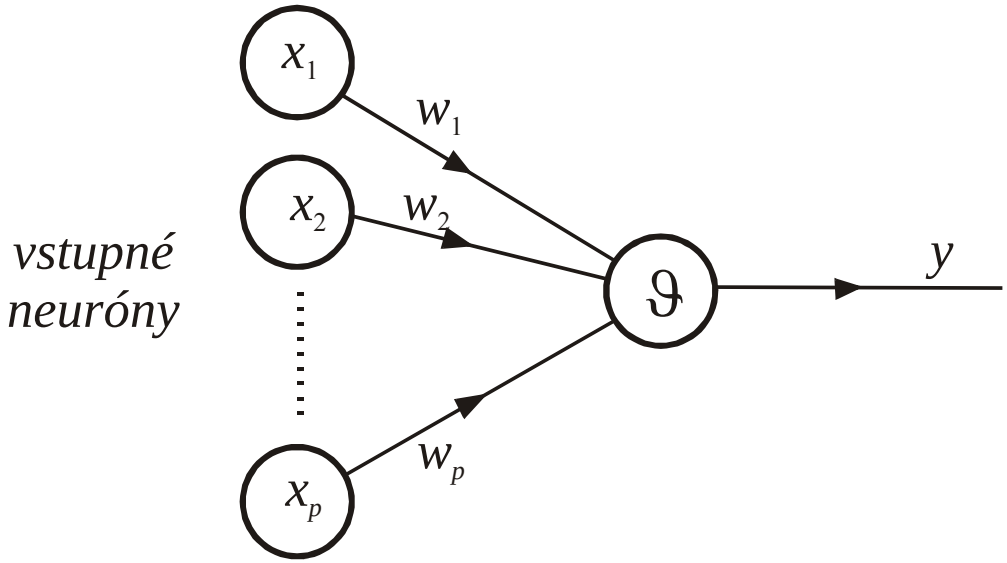
Rosenblattova predstava o perceptróne

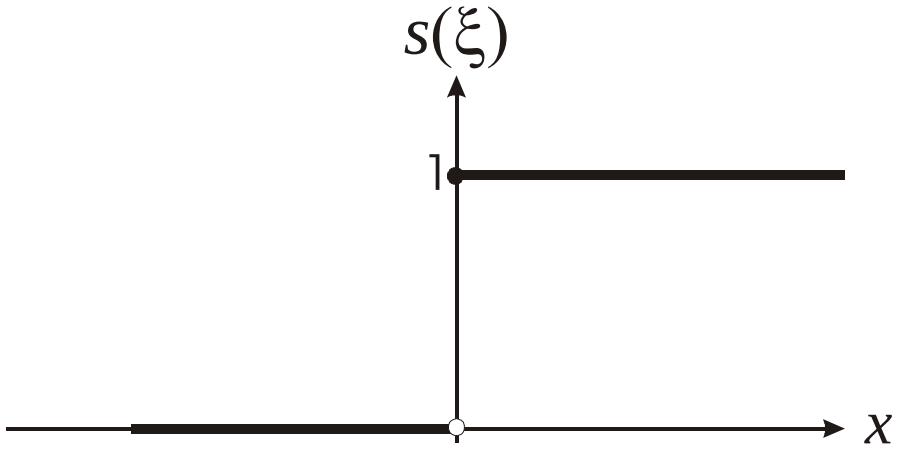


Oblasť, kde sa premieta optickým systémom oka pozorovaný objekt, sa nazýva sietnica. Tá prenáša binárne hodnoty do vrstvy nazývanej projekčná area, kde sa binárne kódovaný obraz numericky predspracováva. Spoje medzi sietnicou a projekčnou oblasťou sú pevné a neadaptabilné. Spoje do druhej vrstvy (asociačnej arey) a tiež aj do tretej vrstvy (odozvová area) sú stochasticky generované. Základným cieľom adaptačného procesu perceptrónu je nastaviť váhové koeficienty spojov tak, aby aktivity neurónov z tretej vrstvy (odozvová oblasť) správne klasifikovali obraz dopadajúci na sietnicu.

- F. Rosenblat, The perceptron: A probabilistic model for information storage and organization in the brain, *Psychological Review* **65** (1958), 386-408.
- F. Rosenblatt, *Principles of Neurodynamics.* Spartan Books, Washington D.C., 1962.

Rosenblattov perceptrón

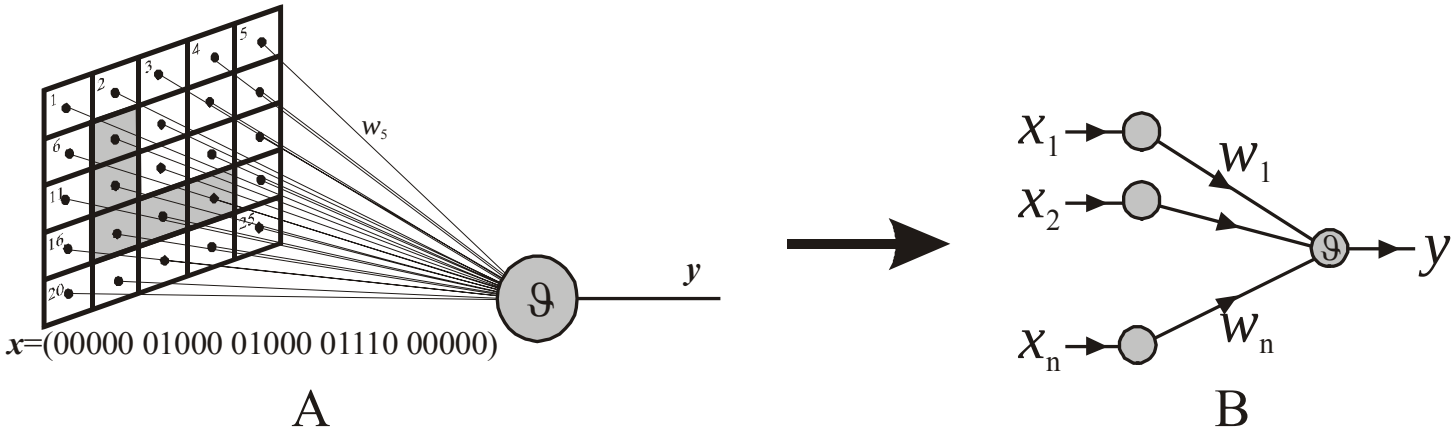




$$y = s(\xi) = s\left(\sum_{i=1}^{p} w_i x_i + \vartheta\right), \qquad s(\xi) = \begin{cases} 1 & (\xi \geq 0) \\ 0 & (\text{ináč}) \end{cases}$$

Poznámka: Váhové koeficienty w_i a prahový koeficient ϑ sú reálne premenné parametre.

Minského a Papertova predstava o perceptróne



Perceptrón obsahuje dve vrstvy. V prvej vrstve sú vstupné neuróny, pomocou ktorých sa binárne kóduje obrázok na "sietnici" perceptrónu. Druhá vrstva obsahuje len jeden (výstupný) neurón, ktorého binárna aktivita kóduje písmeno zo sietnice oka. Ak je toto písmeno "L", potom požadovaná výstupná aktivita je jednotková, v opačnom prípade, pre všetky ostatné písmená je výstupná aktivita nulová. Každý spoj z *i*-tého vstupného neurónu do výstupného neurónu je ohodnotený váhovým koeficientom w_i. Výstupný neurón je ohodnotený prahom ϑ. (B) Formálne vyjadrenie perceptrónu ako dvojvrstvovej neurónovej siete.

Rosenblatt's perceptron algorithm

A **training example** $\mathscr{A}_k$ is an ordered pair $(\boldsymbol{x}_k, \hat{y}_k)$, where $\boldsymbol{x}_k=(x_{1k},x_{2k},\ldots,x_{nk})$ is a pattern vector and $\hat{y}_k=0$ or 1 (depending on the required output of the neuron for input pattern $\boldsymbol{x}_k$).
A **training set** $\mathscr{A}$ is simply a set of training examples, $\mathscr{A}=\{\mathscr{A}_k\}$. Let

$$\mathscr{A} = \mathscr{A}^1 \cup \mathscr{A}^0$$
$$\mathscr{A}^1 = \{\boldsymbol{x}_k ;\ \mathscr{A}_k = (\boldsymbol{x}_k, \hat{y}_k) \in \mathscr{A} \text{ and } \hat{y}_k=1\}$$
$$\mathscr{A}^0 = \{\boldsymbol{x}_k ;\ \mathscr{A}_k = (\boldsymbol{x}_k, \hat{y}_k) \in \mathscr{A} \text{ and } \hat{y}_k=0\}$$

For a given training set $\mathscr{E}$ a **learning task** consists in finding a weight vector $\boldsymbol{w}^*$ and a threshold ϑ so that

$$\forall \boldsymbol{x}_k \in \mathscr{A}^1 : \boldsymbol{w}^* \cdot \boldsymbol{x}_k + \vartheta > 0$$
$$\forall \boldsymbol{x}_k \in \mathscr{A}^0 : \boldsymbol{w}^* \cdot \boldsymbol{x}_k + \vartheta < 0$$

Learning algorithm

Let $\mathcal{A}_k=(\boldsymbol{x}_k,\hat{y}_k)$ be an example from the training set. Let $\boldsymbol{w}=(w_1,w_2,...,w_n)$ and ϑ be the current values of weights and threshold of the neuron. For a pattern $\boldsymbol{x}_k$ we calculate a potential of neuron

$$\xi_k = \sum_{i=1}^{n} w_i x_{ik} + \vartheta$$

An output activity of the neuron is determined by

$$y_k = \begin{cases} 1 & (\text{if } \xi_k \geq 0) \\ 0 & (\text{otherwise}) \end{cases}$$

The weights and threshold are updated with respect to the example $\mathcal{A}_k=(\boldsymbol{x}_k, \hat{y}_k)$ as follows (Hebbian learning)

$$\boldsymbol{w}_{new} = \boldsymbol{w}_{old} + \lambda(\hat{y}_k - y_k)\boldsymbol{x}_k$$

$$\vartheta_{new} = \vartheta_{old} + \lambda(\hat{y}_k - y_k)$$

This process is repeated until all examples of the training set are correctly interpreted, i.e. the required output activity $\hat{y}_k$ is equal to the calculated output activity y. Parameter $\lambda>0$ is called the **learning rate**.

Illustrative example

For better understanding, the above percetron algorithm will be illustrated by simple example. Let us have a training set composed of four examples (Boolean OR function)

Example	x_0	x_1	x_2	$\hat{y}$
$\mathscr{A}_1$	1	0	0	0
$\mathscr{A}_2$	1	0	1	1
$\mathscr{A}_3$	1	1	0	1
$\mathscr{A}_4$	1	1	1	1

The weight vector is determined by $\boldsymbol{w}=(\vartheta,w_1,w_2)$ and the pattern vector should be as $\boldsymbol{x}=(1,x_1,x_2)$. Single iteration steps of the preceptor algorithm for $\lambda=1$ are

Step 0 (initialization). $\boldsymbol{w}=(0,0,0)$.
Step 1. example $\mathscr{E}_1$ is selected, $\xi_1=\boldsymbol{w}\cdot\boldsymbol{x}_1=(0,0,0)\cdot(1,0,0)=0$, $y_1=0=\hat{y}_1$, $\boldsymbol{w}$ is not updated.
Step 2. example $\mathscr{E}_2$ is selected, $\xi_2=\boldsymbol{w}\cdot\boldsymbol{x}_2=(0,0,0)\cdot(1,0,1)=0$, $y_2=0\neq\hat{y}_2$.
Step 2'. $\boldsymbol{w}=(0,0,0)+(1-0)(1,0,1)=(1,0,1)$.
Step 3. example $\mathscr{E}_3$ is selected, $\xi_3=\boldsymbol{w}\cdot\boldsymbol{x}_3=(1,0,1)\cdot(1,1,0)=1$, $y_3=1=\hat{y}_3$, $\boldsymbol{w}$ is not updated.
Step 4. example $\mathscr{E}_4$ is selected, $\xi_4=\boldsymbol{w}\cdot\boldsymbol{x}_4=(1,0,1)\cdot(1,1,1)=2$, $y_4=1=\hat{y}_4$, $\boldsymbol{w}$ is not updated.
Step 5. example $\mathscr{E}_1$ is selected, $\xi_1=\boldsymbol{w}\cdot\boldsymbol{x}_1=(1,0,1)\cdot(1,0,0)=1$, $y_1=1\neq\hat{y}_1$.
Step 5'. $\boldsymbol{w}=(1,0,1)+(0-1)(1,0,0)=(0,0,1)$.
Step 6. example $\mathscr{E}_2$ is selected, $\xi_2=\boldsymbol{w}\cdot\boldsymbol{x}_2=(0,0,1)\cdot(1,0,1)=1$, $y_2=1=\hat{y}_2$, $\boldsymbol{w}$ is not updated.
Step 7. example $\mathscr{E}_3$ is selected, $\xi_3=\boldsymbol{w}\cdot\boldsymbol{x}_3=(0,0,1)\cdot(1,1,0)=0$, $y_3=0\neq\hat{y}_3$.
Step 7'. $\boldsymbol{w}=(0,0,1)+(1-0)(1,1,0)=(1,1,1)$.
Step 8. example $\mathscr{E}_4$ is selected, $\xi_4=\boldsymbol{w}\cdot\boldsymbol{x}_4=(1,1,1)\cdot(1,1,1)=1$, $y_4=1=\hat{y}_4$, $\boldsymbol{w}$ is not updated.
Step 9. example $\mathscr{E}_1$ is selected, $\xi_1=\boldsymbol{w}\cdot\boldsymbol{x}_1=(1,1,1)\cdot(1,0,0)=1$, $y_1=1\neq\hat{y}_1$.
Step 9'. $\boldsymbol{w}=(1,1,1)+(0-1)(1,0,0)=(0,1,1)$.
Step 10. example $\mathscr{E}_2$ is selected, $\xi_2=\boldsymbol{w}\cdot\boldsymbol{x}_2=(0,1,1)\cdot(1,0,1)=1$, $y_2=1=\hat{y}_2$, $\boldsymbol{w}$ is not updated.
Step 11. example $\mathscr{E}_3$ is selected, $\xi_3=\boldsymbol{w}\cdot\boldsymbol{x}_3=(0,1,1)\cdot(1,1,0)=1$, $y_3=1=\hat{y}_3$, $\boldsymbol{w}$ is not updated.
Step 12. example $\mathscr{E}_4$ is selected, $\xi_4=\boldsymbol{w}\cdot\boldsymbol{x}_4=(0,1,1)\cdot(1,1,1)=2$, $y_4=1=\hat{y}_4$, $\boldsymbol{w}$ is not updated.
Step 13. example $\mathscr{E}_1$ is selected, $\xi_1=\boldsymbol{w}\cdot\boldsymbol{x}_1=(0,1,1)\cdot(1,0,0)=0$, $y_1=0=\hat{y}_1$, $\boldsymbol{w}$ is not updated.

The algorithm appears to successfully perform the learning task. But it is not obvious whether it would work in general.

Theorem (proved by Novikoff). For a training set $\mathcal{A} = \mathcal{A}^0 \cup \mathcal{A}^1$ the perceptron learning algorithm is guaranteed to find a weight vector $\boldsymbol{w}^*$ such that

$$\forall \mathcal{A}_k \in \mathcal{A}^1 : \boldsymbol{w}^* \cdot \boldsymbol{x}_k \geq \delta \text{ and } \forall \mathcal{A}_k \in \mathcal{A}^0 : \boldsymbol{w}^* \cdot \boldsymbol{x}_k \leq -\delta$$

for some $\delta > 0$, whenever such a solution $\boldsymbol{w}^*$ exists.

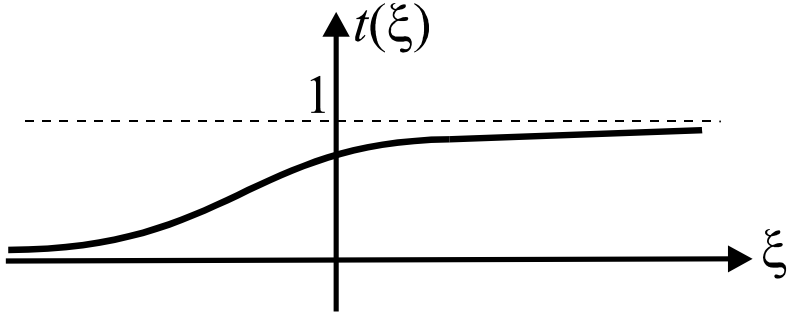
Continuous perceptron

Many practical problems call for continuous function approximation. This requires neurons that are able to compute a real valued function $f : R_n \rightarrow (0,1)$. This function is determined for purposes of our forthcoming considerations by a training set $\mathscr{A} = \{\mathscr{A}_k = (\boldsymbol{x}_k, \hat{y}_k = f(\boldsymbol{x}_k)); k=1,2,...,P\}$. A **continuous threshold neuron** is determined by

$$y = t\left(\sum_{i=1}^{n} w_i x_i + \vartheta\right) = \varphi(x_1, x_2, \ldots, x_n; w_1, w_2, \ldots, \vartheta)$$

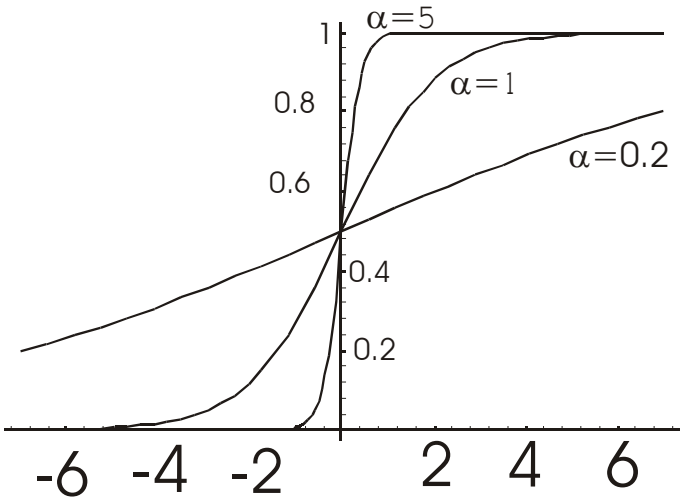
where $t : R \rightarrow (0,1)$ is a **transfer function** specified most generally as follows:
(1) It is differentiable and monotonously increasing,
(2) and satisfying two asymptotic conditions $t(-\infty)=0$ and $t(\infty)=1$.

Since the transfer function is monotonously increasing there exists also an inverse function t^{-1} : $(0,1) \rightarrow R$ with respect to the original transfer function.



Usually, the transfer function is analytically realized by the so-called **sigmoid function**

$$t_\alpha(\xi) = \frac{1}{1+e^{-\alpha\xi}}$$



where α is a real positive parameter called the **slope**. It is easy to see that for $\alpha \to \infty$ the transfer function turns to a simple step function

$$\lim_{\alpha\to\infty} t_\alpha(\xi) = \begin{cases} 1 & (\xi > 0) \\ 1/2 & (\xi = 0) \\ 0 & (\xi < 0) \end{cases}$$

An activity of continuous neurons for an example $\mathscr{A}_k=(\boldsymbol{x}_k,\hat{y}_k)\in\mathscr{A}$ is determined

$$y_k = t\left(\sum_{i=1}^{n} w_i x_{ik} + \vartheta\right)$$

where the k-th pattern is $\boldsymbol{x}_k=(x_{1k}, x_{2k},..., x_{nk})$. A continuous neuron may be formally considered as a parametric function $y = \varphi(x_1, x_2,..., x_n; w_1, w_2,..., w_n, \vartheta)$ specified by parameters - weights and threshold.

Goal. To learn the neuron function so that its values will approximate the required values from the training set as closest as possible. Let us define an objective function for the k-th training example

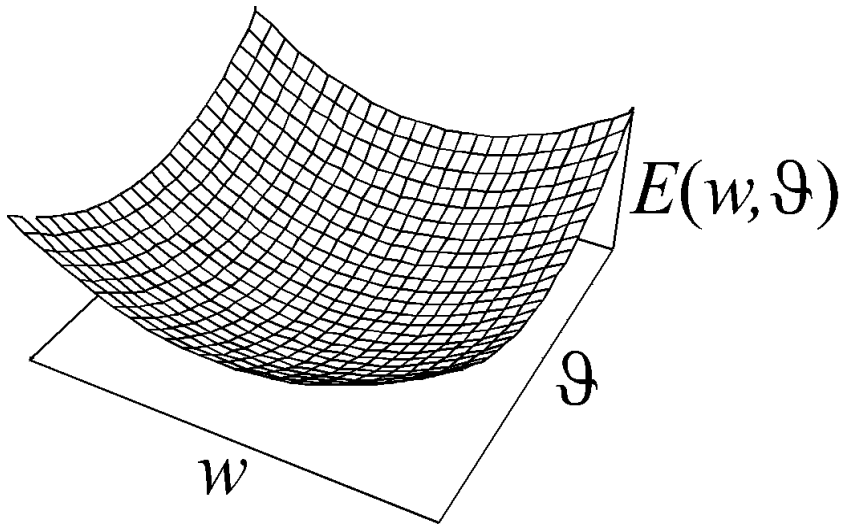
$$E_k(\boldsymbol{w}, \vartheta) = \frac{1}{2}(y_k - \hat{y}_k)^2$$

where y_k is a response of the neuron on the training set pattern $\boldsymbol{x}_k$. The objective function can be simply generalized for the whole training set

$$E(\boldsymbol{w}, \vartheta) = \sum_{k=1}^{P} E_k(\boldsymbol{w}, \vartheta) = \frac{1}{2}\sum_{k=1}^{P}(y_k - \hat{y}_k)^2$$

Alternative goal. To find such optimal weights and threshold that minimize the objective function, i.e. the continuous neuron with these optimal weights and threshold models closely the function $f(\boldsymbol{x})$ determined by the training set examples

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



Schematic plot of the objective function as a function of weights and threshold. Minimum of this function corresponds to optimal weights and threshold that best model function $f(\boldsymbol{x})$ through the training set of examples.

The optimization process of nonlinear objective function is realized by the so-called **steepest descent method**, i.e. by the simplest gradient method based on the following updating of weights and threshold

$$w_i^{(t+1)} = w_i^{(t)} - \lambda \frac{\partial E\left(\boldsymbol{w}^{(t)}, \vartheta^{(t)}\right)}{\partial w_i}$$

$$\vartheta^{(t+1)} = \vartheta^{(t)} - \lambda \frac{\partial E\left(\boldsymbol{w}^{(t)}, \vartheta^{(t)}\right)}{\partial \vartheta}$$

where $\lambda>0$ is a **learning step**.

The recurrent updating of weights and threshold is initiated by a random generation of them, the above formulae are repeatedly applied until the gradient of the objective function is smaller than a prescribed norm (precision)

$$|\text{grad } E(\boldsymbol{w}, \vartheta)| = \left(\sum_{i=1}^{n} \left(\frac{\partial E}{\partial w_i} \right)^2 + \left(\frac{\partial E}{\partial \vartheta} \right)^2 \right)^{1/2} < \varepsilon$$

The gradient of the objective function E is determined by

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

where

$$\frac{\partial E_k}{\partial w_i} = (y_k - \hat{y}_k) t'(\xi_k) x_{ik} = (y_k - \hat{y}_k) y_k (1 - y_k) x_{ik}$$

$$\frac{\partial E_k}{\partial \vartheta} = (y_k - \hat{y}_k) t'(\xi_k) = (y_k - \hat{y}_k) y_k (1 - y_k)$$

Updating formulae are

$$w_i^{(t+1)} = w_i^{(t)} - \lambda \sum_{k=1}^{P} \frac{\partial E_k\left(\boldsymbol{w}^{(t)}, \vartheta^{(t)}\right)}{\partial w_i}$$

$$= w_i^{(t)} - \lambda \sum_{k=1}^{P} (y_k - \hat{y}_k) t'(\xi_k) x_{ik}$$

$$= w_i^{(t)} - \lambda \sum_{k=1}^{P} (y_k - \hat{y}_k) y_k (1 - y_k) x_{ik}$$

$$\vartheta^{(t+1)} = \vartheta^{(t)} - \lambda \sum_{k=1}^{P} \frac{\partial E_k\left(\boldsymbol{w}^{(t)}, \vartheta^{(t)}\right)}{\partial \vartheta}$$

$$= \vartheta^{(t)} - \lambda \sum_{k=1}^{P} (y_k - \hat{y}_k) t'(\xi_k)$$

$$= \vartheta^{(t)} - \lambda \sum_{k=1}^{P} (y_k - \hat{y}_k) y_k (1 - y_k)$$

These formulae correspond to the so-called **batch learning** when gradient is calculated through all examples of the training set.

There exist also the so-called **on-line learning** when updatings are performed separately for each training example, then learning formulae are

$$w_i^{(t+1)} = w_i^{(t)} - \lambda \frac{\partial E_k\left(\boldsymbol{w}^{(t)}, \vartheta^{(t)}\right)}{\partial w_i} = w_i^{(t)} + \lambda(\hat{y}_k - y_k)t'(\xi_k)x_{ik}$$

,

$$\vartheta^{(t+1)} = \vartheta^{(t)} - \lambda \frac{\partial E_k\left(\boldsymbol{w}^{(t)}, \vartheta^{(t)}\right)}{\partial \vartheta} = \vartheta^{(t)} + \lambda(\hat{y}_k - y_k)t'(\xi_k)$$

Since the transfer function is monotonously increasing, its first derivative should be positive, $\forall \xi \in R$: $t'(\xi)>0$, the above on-line updating formulae can be simplified so that the first derivative of transfer function is omitted

$$w_i^{(t+1)} = w_i^{(t)} + \lambda(\hat{y}_k - y_k)x_{ik}$$

$$\vartheta^{(t+1)} = \vartheta^{(t)} + \lambda(\hat{y}_k - y_k)$$

These two updating formulae are fully identical to original updating formulae ad-hoc introduced by Rosenblatt for the learning of threshold neurons.

High-order perceptron

Minsky and Papert in their seminal book ***Percetron*** very carefully studied the general properties of threshold neurons (perceptrons) from the point of view of their ability to model an arbitrary function. They demonstrated that **the perceptron is able to model such functions that are determined by linearly separable training sets of examples**. This fact was considered by Minsky and Papert as the very serious drawback of perceptrons, they **concluded that the perceptrons could not be classified as a universal computational device**.

In order to overcome this shortcoming of perceptrons they introduced two new concepts, in particular the **high-order perceptron** and the **hidden neuron**. Both these concepts substantially increase a computational ability of perceptrons to model arbitrary functions, but Minsky and Papert have **precipitately concluded that these concepts, due to their numerical complexity, are not proper tools for generalizations of perceptrons and that the theory of perceptrons is definitely plagued by serious restrictions and therefore the perceptrons do not represent a perspective field in looking for an alternative to already known universal computational devices.**

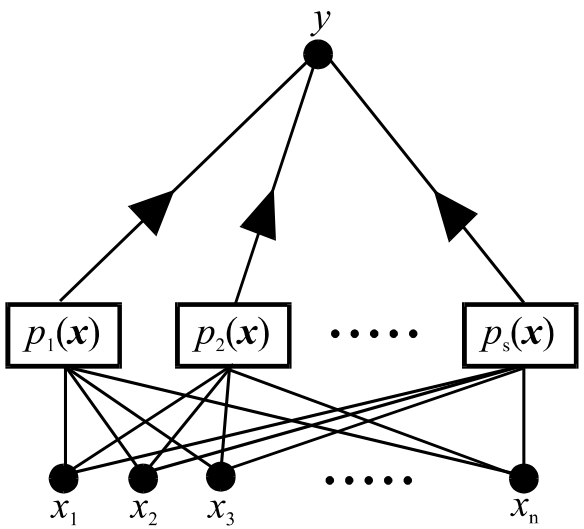
A **high-order threshold neuron** (high-order perceptron) is a generalization of simple threshold neuron (perceptron) so that its potential now contains not only constant and linear terms but also quadratic, cubic,... terms

$$y = t\left(\sum_{i=1}^{n} w_i x_i + \sum_{\substack{i,j=1 \\ (i \le j)}}^{n} w_{ij} x_i x_j + \sum_{\substack{i,j,k=1 \\ (i \le j \le k)}}^{n} w_{ijk} x_i x_j x_k + ... + \vartheta \right)$$

This general formula can be rewritten in another alternative form

$$y = t(w_1 p_1(\boldsymbol{x}) + w_2 p_2(\boldsymbol{x}) + ... + w_s p_s(\boldsymbol{x}) + \vartheta)$$

where $p_i(\boldsymbol{x}) = x_1^{\alpha_1} x_2^{\alpha_2} ... x_n^{\alpha_n}$ are different multilinear terms that are determined by exponents α_1, α_2, ..., α_n .



The high-order perceptron can be learned by the gradient method. In similar way as for simple perceptrons the goal of learning process is to find such optimal weights w_i and threshold ϑ that minimize the objective function.

$$w_i^{(t+1)} = w_i^{(t)} - \lambda \frac{\partial E\left(\boldsymbol{w}^{(t)}, \vartheta^{(t)}\right)}{\partial w_i}$$

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} - \lambda \frac{\partial E\left(\boldsymbol{w}^{(t)}, \vartheta^{(t)}\right)}{\partial w_{ij}}$$

.....

$$\vartheta^{(t+1)} = \vartheta^{(t)} - \lambda \frac{\partial E\left(\boldsymbol{w}^{(t)}, \vartheta^{(t)}\right)}{\partial \vartheta}$$

The gradient of the total objective function E over all training set examples, is calculated by a similar manner as for the "linear" perceptron.

Theorem. The **high-order pwerceptron** realized by a parametric function $\varphi(\boldsymbol{x};\boldsymbol{w},\vartheta) = t\left(\sum w_i p_i(\boldsymbol{x}) + \vartheta\right)$ is able to model with the prescribed precision ε an arbitrary function $f{:}R_n \rightarrow (0,1)$ determined by a training set $\mathscr{E}=\{\mathscr{E}_k=(\boldsymbol{x}_k,f(\boldsymbol{x}_k)); k=1,2,...,P\}$

$$\sum_{k=1}^{P} \left|\varphi(\boldsymbol{x}_k;\boldsymbol{w},\vartheta) - f(\boldsymbol{x}_k)\right| < \varepsilon$$

In other words, the high-order threshold neuron is a **universal approximator** of functions $f : R_n \rightarrow (0,1)$ that are determined by training-set examples $\mathscr{A}= \{\mathscr{A}_k = (\boldsymbol{x}_k,f(\boldsymbol{x}_k)); k=1,2,...,P\}$.