



# Analýza a klasifikace dat – přednáška 10



RNDr. Roman Vyškovský

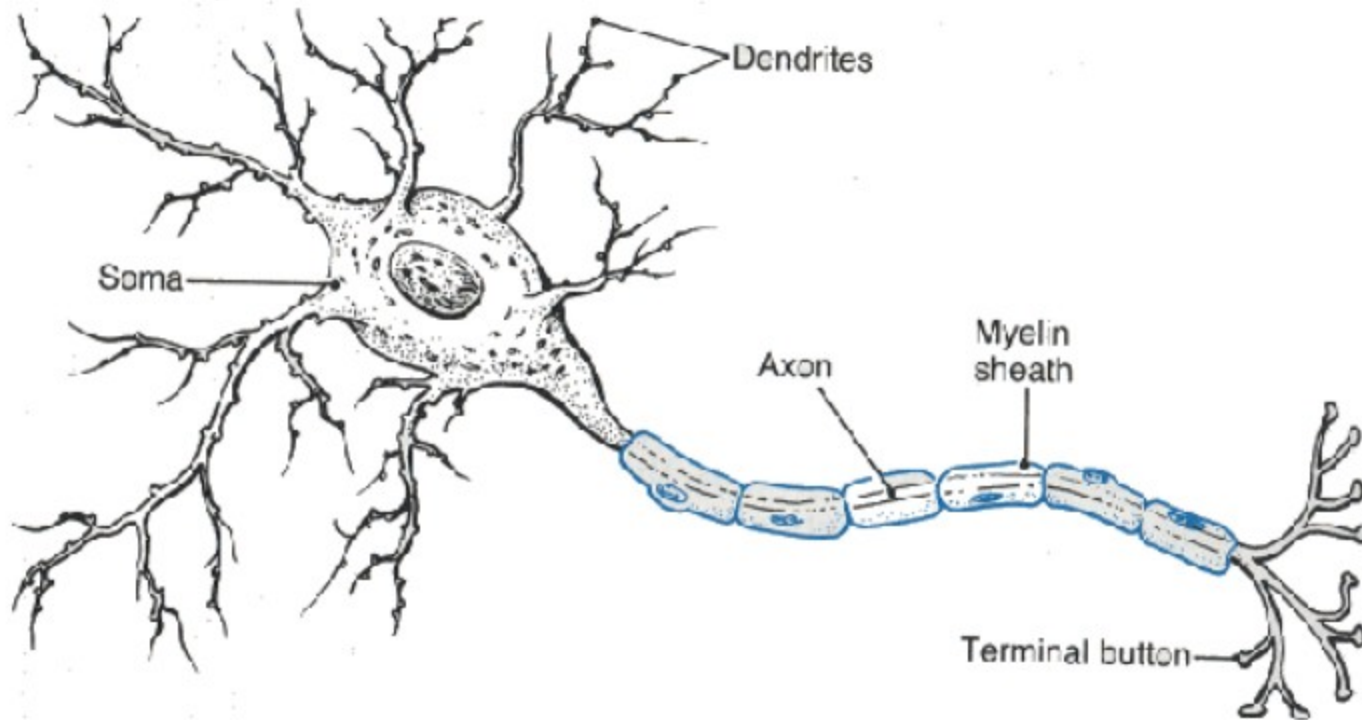
Podzim 2019

# Neuronové sítě

---

- Inspirace přírodou
- Jednotlivý neuron
- Dopředná neuronová síť
- RBF síť
- Soutěživé sítě

# Biologický neuron



Zdroj obrázku: AnimatLab. Neuron Morphology [online]. ©2011 [cit. 2013-4-9]. Dostupné z: <http://www.animatlab.com/NeuralNetworkEditor/FiringRateNeuralNet/NeuronBasics.htm>

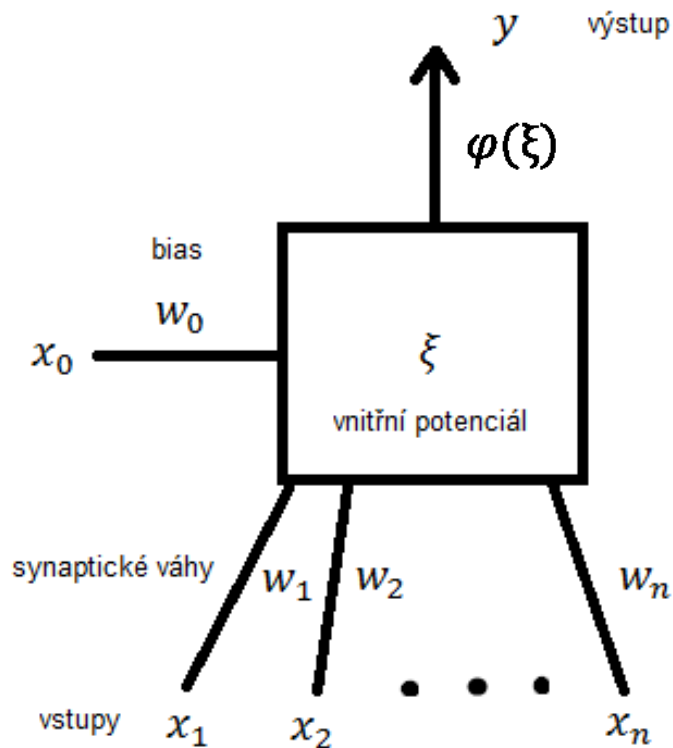
# 3 dynamiky

---

- V teorii neuronových sítí se zabýváme 3 dynamikami
  - Aktivní dynamika: výpočet odezvy neuronu resp. neuronové sítě (klasifikační třída, predikovaná hodnota) na předložený podnět (obrázek, vektor se záznamem o pacientovi)
  - Adaptační dynamika: zahrnuje proces učení neuronu (sítě)
  - Organizační dynamika: určuje topologii (počet neuronů, počet vrstev), nedává smysl u jednotlivého neuronu, pouze u neuronové sítě

# Umělý neuron

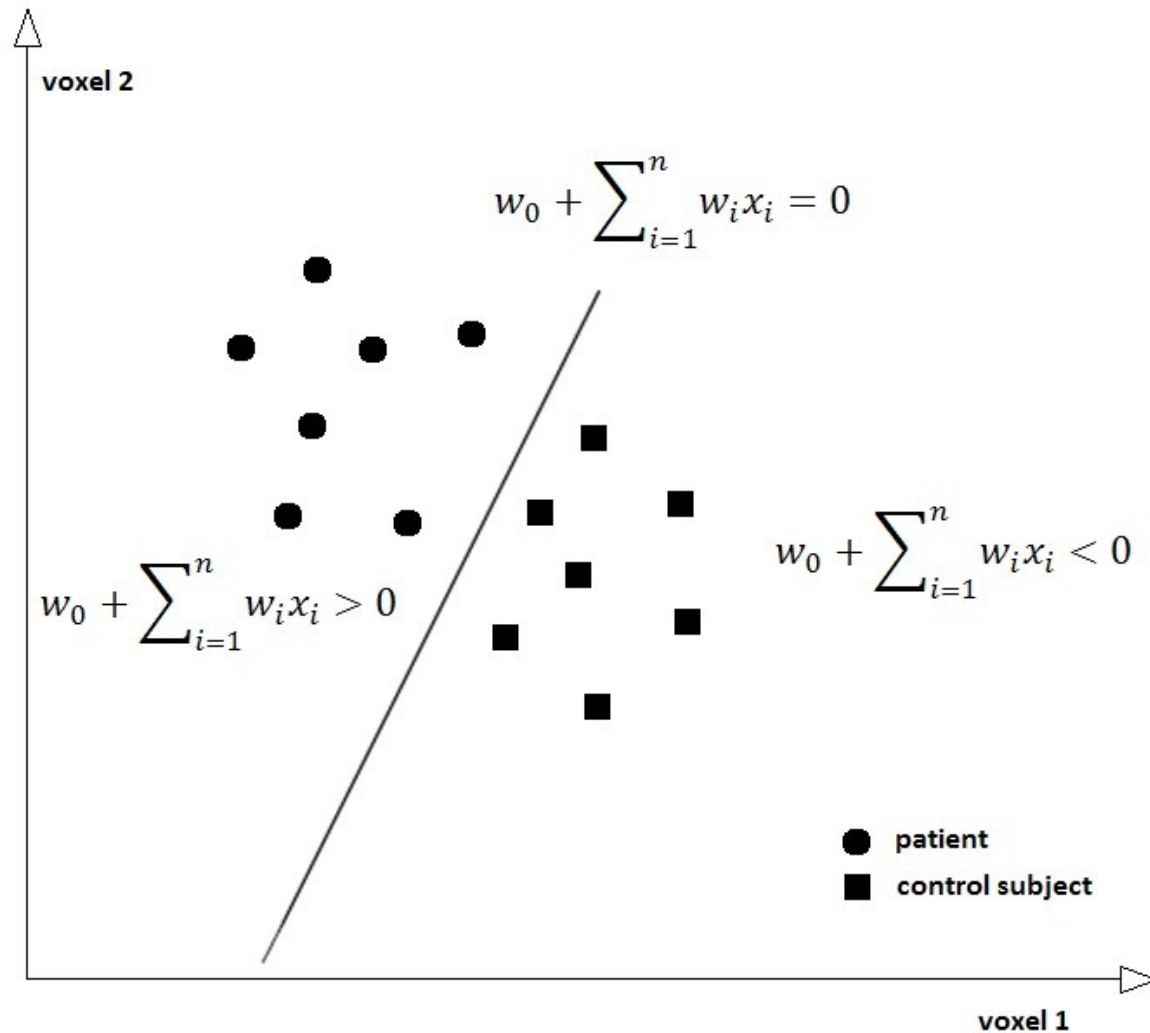
# Umělý neuron



$$\xi = w_0 + \sum_{i=1}^n w_i x_i = w_0 + \mathbf{w}^T \mathbf{x}$$

Vnitřní potenciál

# Geometrická interpretace neuronu



# Aktivační funkce

---

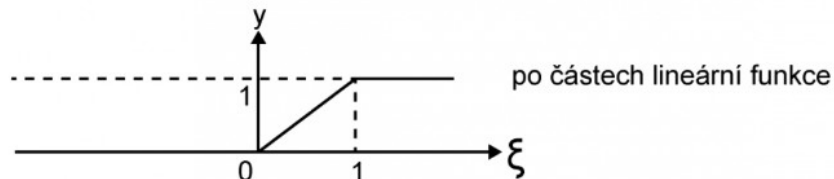
$$y = \varphi \left( w_0 + \sum_{i=1}^n w_i x_i \right)$$



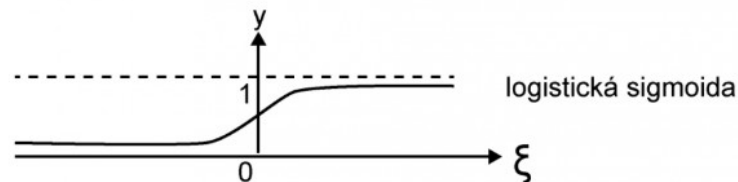
# Aktivační funkce



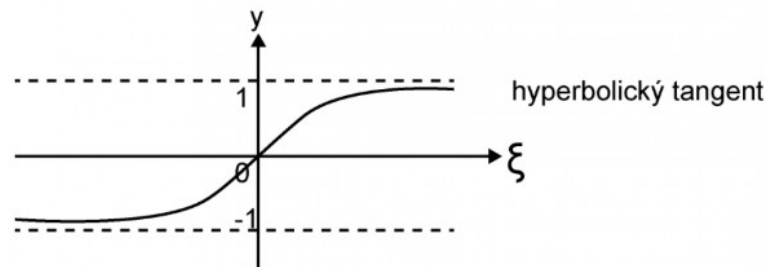
$$\sigma(\xi) = \begin{cases} 1 & \text{if } \xi \geq 0 \\ 0 & \text{if } \xi < 0 \end{cases}$$



$$\sigma(\xi) = \begin{cases} 1 & \xi > 1 \\ \xi & 0 \leq \xi \leq 1 \\ 0 & \xi < 0 \end{cases}$$



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

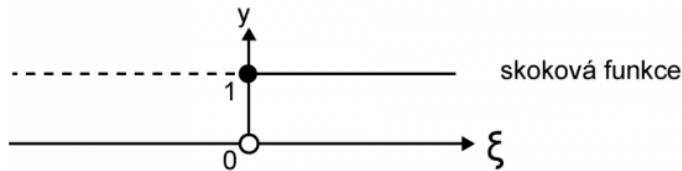


$$\sigma(\xi) = \tanh\left(\frac{1}{2}\xi\right) = \frac{1 - e^{-\xi}}{1 + e^{-\xi}}$$

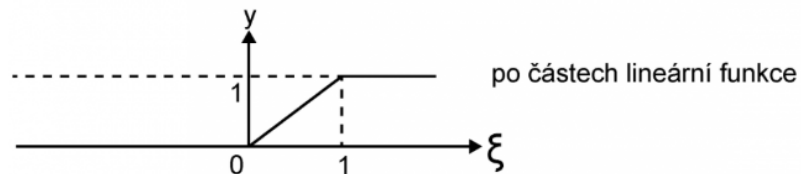
Obrázek převzat z el. učence Umělá inteligence (Autor: Ing. Milan Blaha, Ph.D.)

<https://portal.matematickabiologie.cz/index.php?pg=analiza-a-hodnoceni-biologicky-dat--umela-inteligence>

# Aktivační funkce

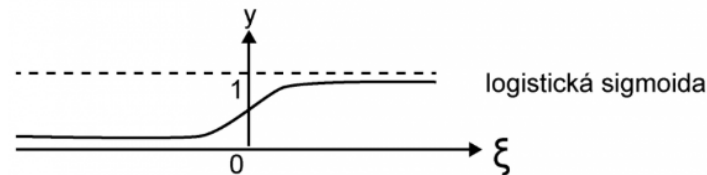


$$\sigma(\xi) = \begin{cases} 1 & \text{if } \xi \geq 0 \\ 0 & \text{if } \xi < 0 \end{cases}$$

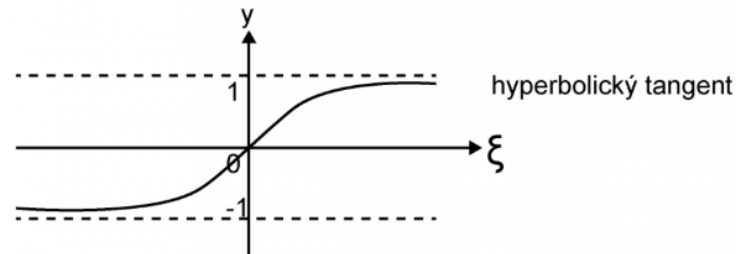


$$\sigma(\xi) = \begin{cases} 1 & \xi > 1 \\ \xi & 0 \leq \xi \leq 1 \\ 0 & \xi < 0 \end{cases}$$

Nediferencovatelné funkce  
- jen pro přímý výpočet vah



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



$$\sigma(\xi) = \tanh\left(\frac{1}{2}\xi\right) = \frac{1 - e^{-\xi}}{1 + e^{-\xi}}$$

Diferencovatelnost funkce  
- je potřeba pro učící algoritmy založené na zpětném šíření chyby

Obrázek převzat z el. učence Umělá inteligence (Autor: Ing. Milan Blaha, Ph.D.)

<https://portal.matematickabiologie.cz/index.php?pg=analiza-a-hodnoceni-biologickych-dat--umela-inteligence>

# Postup učení

---

Učení je iterativní, přičemž jedna iterace vypadá následovně:

0. Inicializace vah
1. Předložíme vstupní vektor
2. Vypočítáme výstup sítě
3. Výstup srovnáme s očekávaným výstupem (změříme chybu)
4. Provedeme korekci vah

Epocha:

- Epochou označujeme interval, kdy předložíme všechny vzory z trénovací množiny
- Po 1 epoše zkontrolujeme, jestli bylo splněno některé kritérium pro ukončení učení, pokud ne, proces pokračuje od bodu 1 další epochou
- Epoch může být stovky, tisíce i více

# Hebbovo učení

Pro neuron s binárními vstupy a binárními výstupy:

- 1) Neuron excitován korektně (výstup = 1, očekávaný výstup = 1)
  - Váhy vedoucí od vstupů  $x_i=1$  se posílí o delta
- 2) Neuron excitován nekorektně (výstup = 1, očekávaný výstup = 0)
  - Váhy vedoucí od vstupů  $x_i=1$  se zmenší o delta
- 3) Neuron neexcitován (výstup = 0)
  - Nic se neupravuje

# Delta pravidlo

Pro neuron s reálnými vstupy, reálným výstupem a lineární aktivační funkcí:

$$w(n + 1)_i = w(n)_i + \mu(y_d - y)$$

$\mu \in (0,1)$ ...rychlost učení

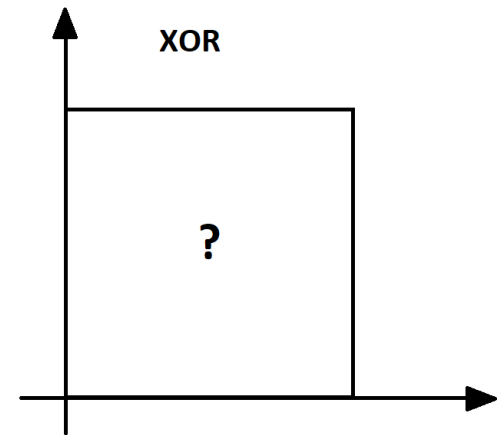
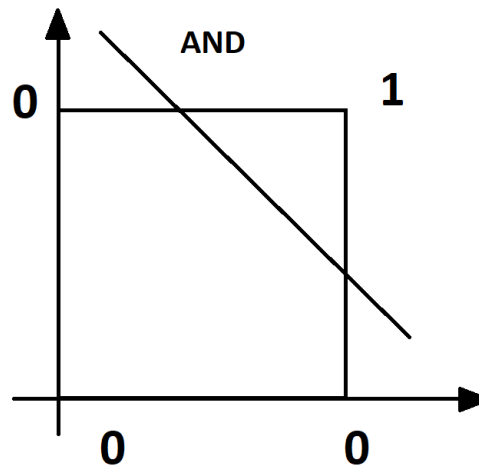
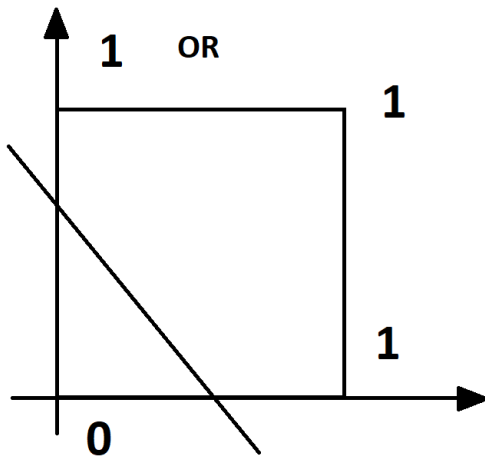
$y_d$ ...očekávaný výstup

Očekávaní	Výsledek	$\mu$ *?	Posun vah
0	0	0	-
1	0	$\mu$	Zvětší se*
0	1	$-\mu$	Zmenší se*
1	1	0	-

\*o  $\mu$ -násobek rozdílu od očekávané hodnoty

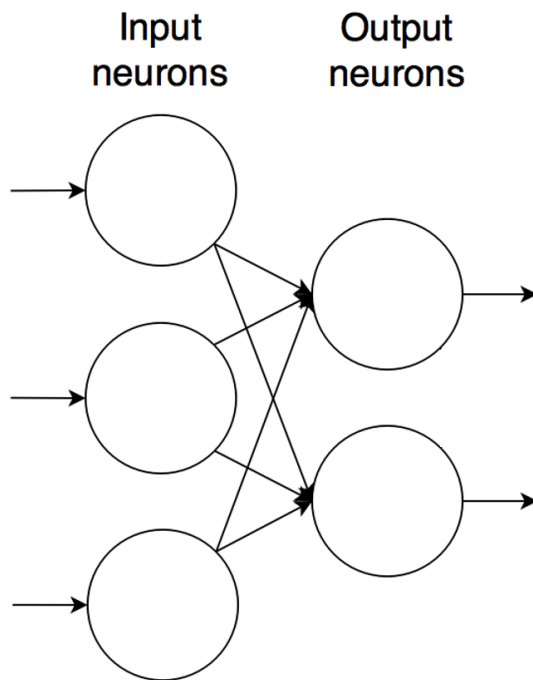
# Limitace klasifikace pomocí 1 neuronu

Jeden neuron rozdělí příznakový prostor nadrovinou na 2 poloroviny, tedy umí řešit jen lineárně separovatelné problémy, v případě booleovských funkcí umí řešit např. OR a AND, ale neumí řešit XOR



# Dopředná neuronová síť

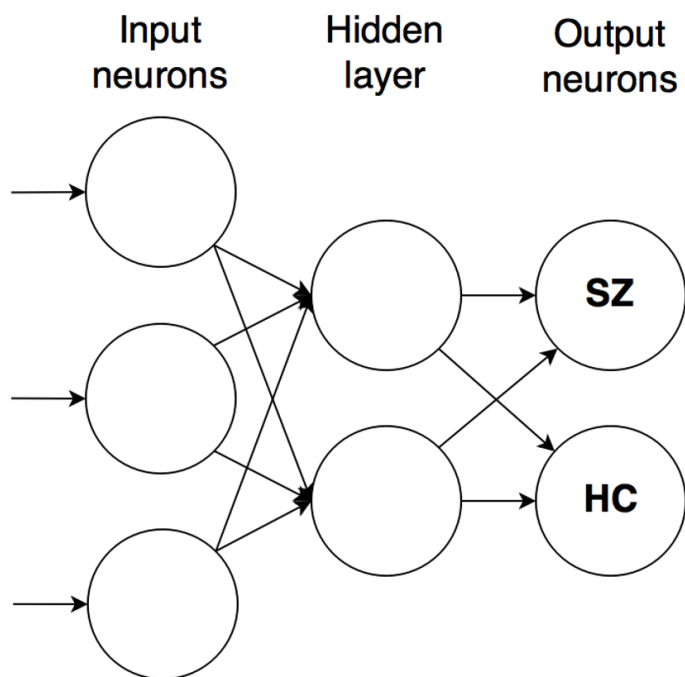
# Jednovrstvý perceptron



- Zobrazení z  $R^n$  do  $R^m$
- Úplně spojení vstupů se všemi neurony
- Neurony jsou na sobě nezávislé
- Pro klasifikaci stejně nepoužitelný jako jednovrstvý perceptron



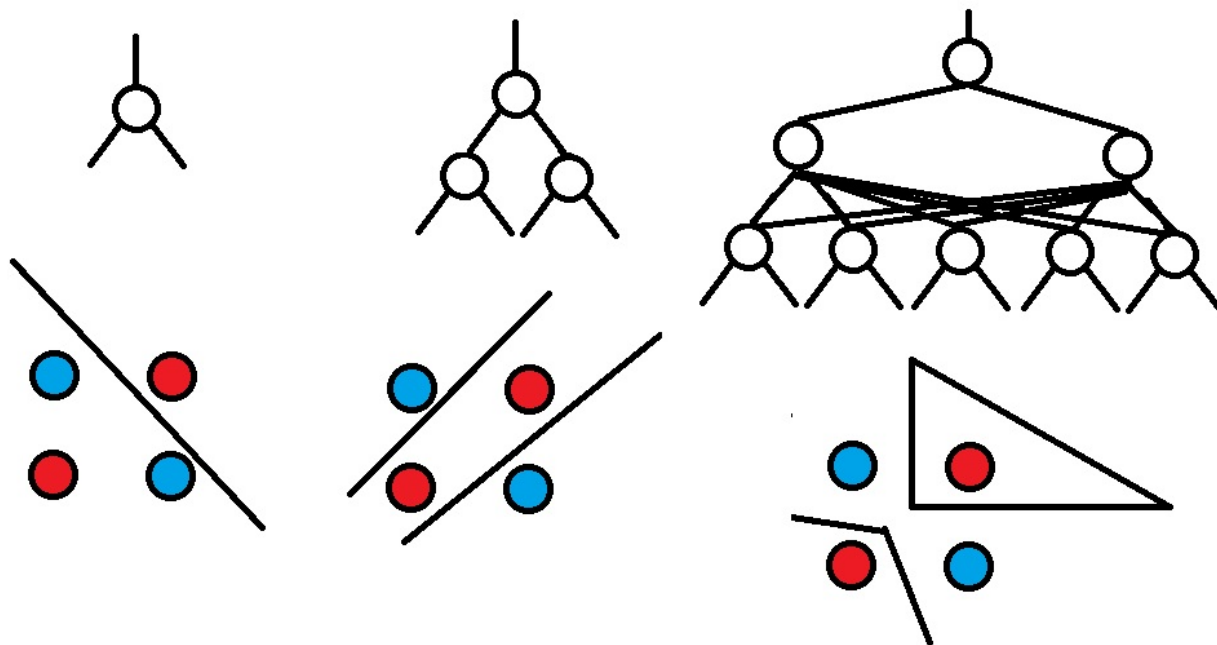
# Vícevrstvý perceptron



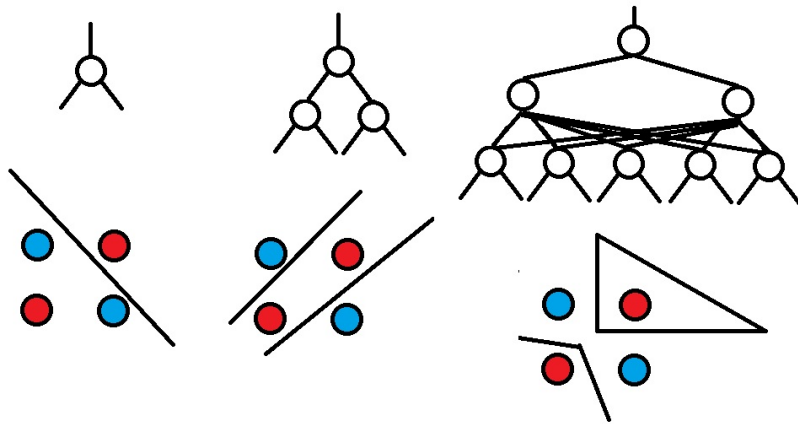
## Organizační dynamika:

- NS obsahuje alespoň jednu skrytou vrstvu neuronů
- Počet neuronů skryté vrstvy může být libovolný, obvykle se počet neuronů snižuje – dochází tak k redukci informace
- Počet neuronů výstupní vrstvy zpravidla odpovídá počtu klasifikačních tříd

# Klasifikační schopnost neuronové sítě



# Klasifikační schopnost neuronové sítě



- Příklad!!
  - První vrstva rozdělí příznakový prostor na poloroviny
  - Druhá vrstva rozdělí prostor na konvexní podoblasti
  - Třetí vrstva provede sjednocení těchto oblastí
- NS je black box, nevíme, jak se naučí, ale tento příklad lze pomocí NS realizovat, stejně jako jakoukoliv boolovskou funkci  $N$  proměnných

# Učení neuronové sítě

---

- Předkládáme příklady vstup – výstup
- Proces učení je iterativní a pro jeho ukončení lze využít validační množinu dat, což zaručuje nepřeučení sítě a nalezení dostatečně obecné transformace dat. Učení lze ukončit:
  - Dosažením maximálního počet epoch
  - Několik epoch po sobě nedochází ke snížení chyby
  - Dosažení požadované chyby
- NS se může naučit obtížně řešitelné úlohy
- Neuronová síť je black box, nelze nahlédnout do vnitřní struktury

# Vícevrstvý perceptron

**Tréninková množina  $T$**   
obsahuje dvojice  $(x_p, d_p)$ ,  
kde  $x \in R_n$  je vstupní vektor  
a  $d_p \in R$  očekávaná  
výstupní hodnota, v případě  
vícedimenzionálního  
výstupu výstupní vektor  
 $d_p \in R_m$

## Postup učení:

1. Ukážeme síti vstup  $x$
2. Vypočítáme výstup sítě  $y$
3. Změříme odchylku  
(chybu) na výstupu sítě:  
$$e_p = d_p - y_p$$
4. **Zpětným šířením chyby**  
přepočítáme tuto chybu  
na vnitřní vrstvy sítě
5. Pro každý neuron  
upravíme váhy podle  
naměřené chyby

# Chybová funkce

$$E(\mathbf{w}) = \sum_{k=1}^p E_k(\mathbf{w})$$

- $\mathbf{w}$  – vektor vah
- $k$  –  $k$ -tý vzor
- $E$  – suma parciálních chyb pro jednotlivé vzory, přičemž platí:

$$E_k(\mathbf{w}) = \frac{1}{2} \sum_{j \in Y} (y_j(\mathbf{w}, \mathbf{x}_k) - d_{kj})^2$$

vypočítaný výstup – očekávaný výstup

- Suma chyb přes všechny výstupní neurony
- $\frac{1}{2}$  - kvůli zjednodušení derivování

# Úprava vah

- Váhy jsou inicializovány náhodně kolem 0
- V  $t$ -tém kroku je váha  $w^{(t)}$  vypočítána pomocí vztahu:

$$w_{ji}^{(t)} = w_{ji}^{(t-1)} + \Delta w_{ji}^{(t)},$$

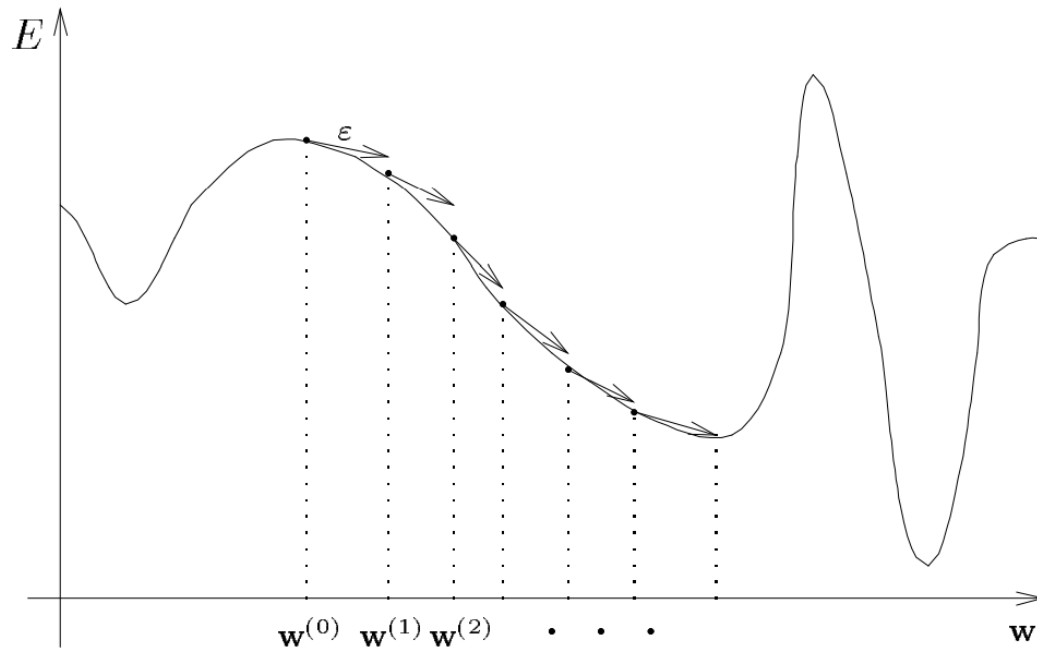
kde  $w_{ji}^{(t)}$  je nový stav váhy z neuronu  $i$  do  $j$  v čase  $t$ ,  $w_{ji}^{(t-1)}$  je stav váhy z neuronu  $i$  do  $j$  v čase  $t - 1$  a  $\Delta w_{ji}^{(t)}$  je změna váhy v kroku  $t$ , přičemž  $t > 0$

- Změna  $\Delta w_{ji}^{(t)}$  je úměrná zápornému gradientu funkce  $E(w)$  v bodě  $w^{(t-1)}$ :

$$w_{ji}^{(t)} = w_{ji}^{(t-1)} - \varepsilon(t) \frac{\partial E}{\partial w_{ji}}(w^{(t-1)}),$$

kde  $0 < \varepsilon(t) \leq 1$  rychlost učení a  $\frac{\partial E}{\partial w_{ji}}(w^{(t-1)})$  parciální derivace chybové funkce podle  $w_{ji}$

# Úprava vah





# Nakonec zderivujeme chybovou funkci podle vah

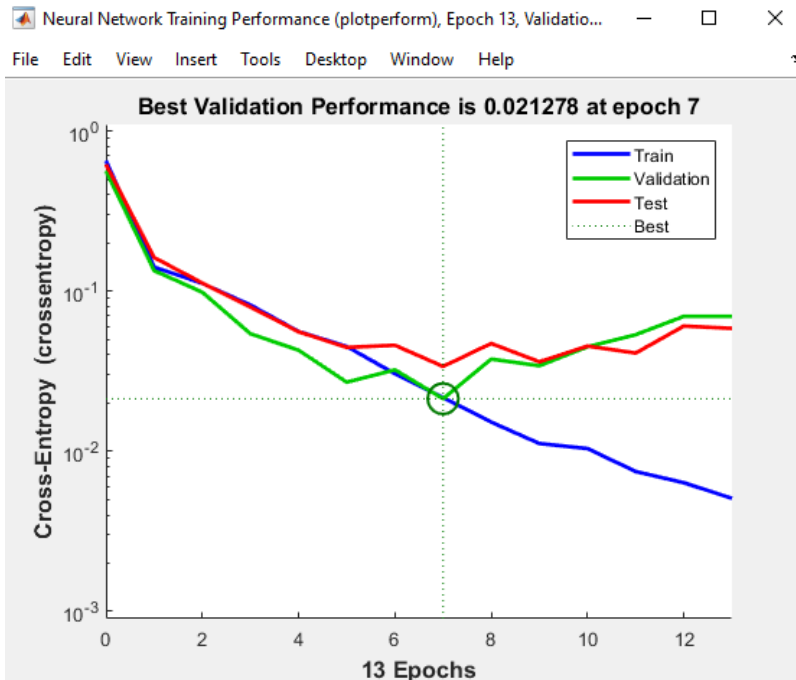
---

$$\frac{\partial E}{\partial w_{ji}} = \sum_{k=1}^p \frac{\partial E_k}{\partial w_{ji}}$$

...

...

# Kontrola procesu učení proti přeučení



- Využijeme validační množinu pro zastavení učení
- Na validační množině kontrolujeme bod (epochu), od kterého se klasifikace zhoršuje
- Další metoda:
  - Ve skrytých vrstvách definujeme menší počty neuron, ...

# Výukové materiály

---

**Například...**

**Neuron:**

<https://portal.matematickabiologie.cz/index.php?pg=analyza-a-hodnoceni-biologickych-dat--umela-intelligence--neuronove-site-jednotlivy-neuron>

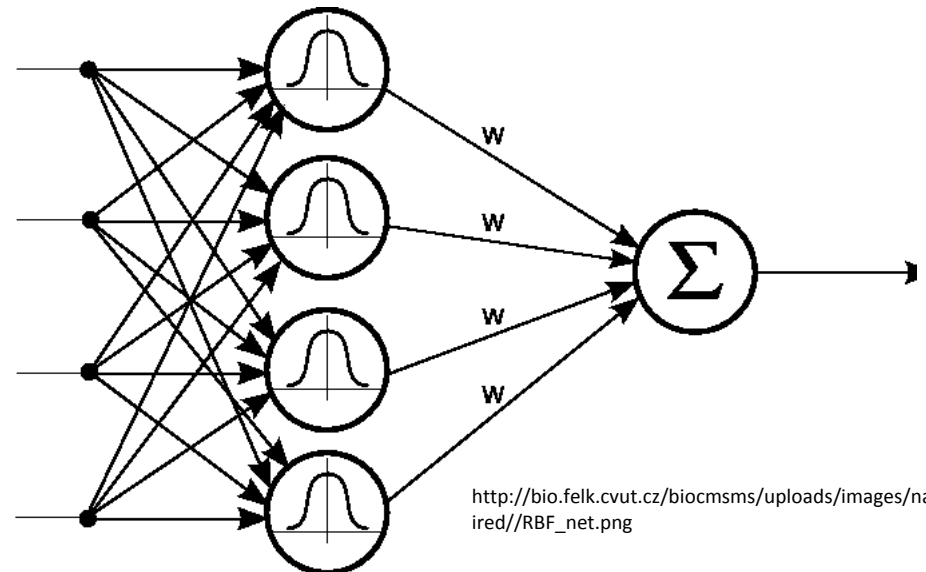
**Neuronová síť:**

<https://portal.matematickabiologie.cz/index.php?pg=analyza-a-hodnoceni-biologickych-dat--umela-intelligence--neuronove-site-perceptrony>

# RBF síť

# RBF síť

- Lokální výpočetní jednotky
- Každý neuron má definované okolí v euklidovském prostoru, kde je citlivý na vstupní vektor
- Citlivost je závislá na vzdálenosti od vektoru
- 2-vrstvé NS s úplným spojením neuronů:
  - 1. vrstva je tvořena RBF neurony
  - 2. vrstva je lineární
- Kombinace učení bez učitele a učení s učitelem



[http://bio.felk.cvut.cz/biocmsms/uploads/images/nature\\_inspired/RBF\\_net.png](http://bio.felk.cvut.cz/biocmsms/uploads/images/nature_inspired/RBF_net.png)

# Výpočet RBF neuronu

- Vnitřní potenciál:

$$\xi = \sqrt{\sum_{i=1}^n (x_i - c_i)^2}$$

kde  $\mathbf{x} = [x_1, \dots, x_n]^T$  je vektor vstupů a  $\mathbf{c} = [c_1, \dots, c_n]^T$  je tzv. střed

- Aktivační funkce:

$$y = \varphi(\xi) = e^{(-\frac{\xi}{\sigma})^2}$$

$\varphi$  – Gaussova funkce ( $\sigma \geq 0$ ) nebo diskretní případ vrací 0 nebo 1 podle prahu apod.

# Soutěživé sítě

Kohonenova mapa

Vektorová kvantizace učení

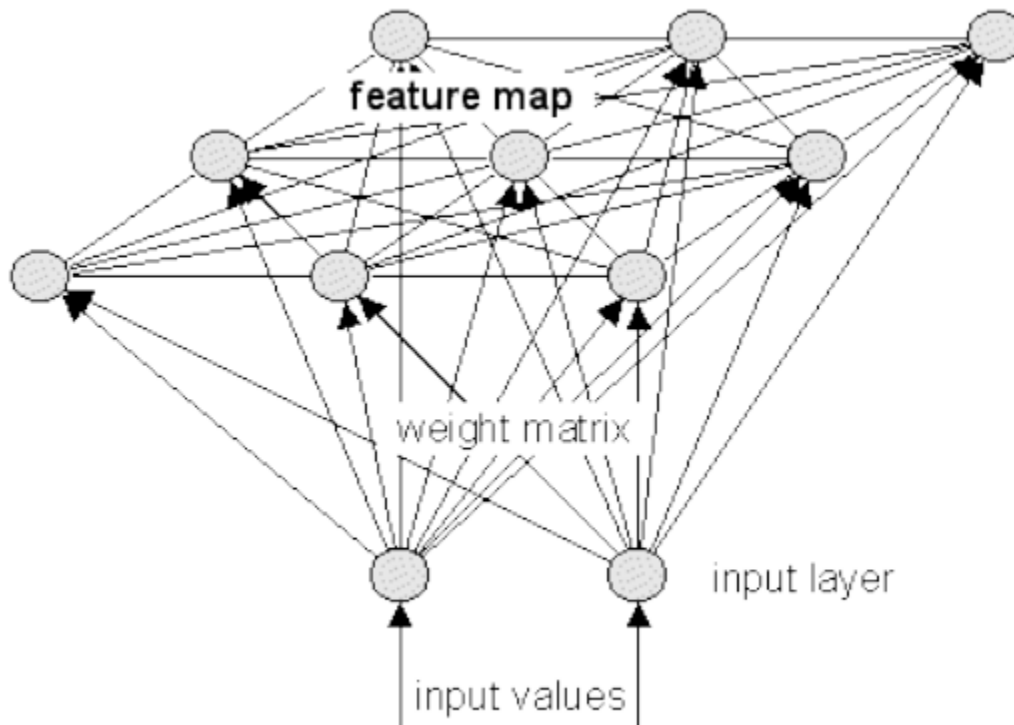
# Kohonenova mapa

---

- Jednovrstvá neuronová síť
- Využívá učení bez učitele
- Autor finský profesor Teuvo Kohonen v roce 1982
- Pro klasifikaci lze využít po doučení pomocí učení s učitelem s využitím tzv. vektorové kvantizace učením tzv. Learning Vector Quantization (LVQ algoritmus)
- Samotná mapa se využívá pro aproximaci pravděpodobnostního rozložení dat



# Kohonenova mapa



# Kohonenova mapa

- Náhodně se rozmístí  $N$  neuronů v příznakovém prostoru
- Neurony jsou uspořádány v mřížce

$$\|x - m_c\| = \min_i \{\|x - m_i\|\},$$

where  $x$  is an input,  $m_c$  the nearest neuron and  $m_i$  is a  $i$ th neuron.

- Je nalezen nejbližší neuron a ten je adaptován pomocí vztahu:

$$w_i(t + 1) = w_i(t) + \alpha(t)[x(t) - w_i(t)]$$

- Kromě tohoto neuronu jsou adaptovány i neurony z okolí, které se v průběhu času zmenšuje  $\rightarrow$  to zajišťuje aproximaci distribuce dat

# Learning vector quantization (LVQ1)

---

- Klasifikace podle metody nejbližšího souseda
- Učení
  - Nejprve se musí určit třídy neuronů
  - Pro každý tréninkový subjekt se najde nejbližší neuron, tento neuron je označen za reprezentanta té třídy, podle toho, pro které subjekty z trénovací množiny byl vícekrát nejbližším neuronem
  - Např. neuron č. 1 je nejbližší pro 5 pacientů a 10 kontrol → je zvolen jako reprezentant kontrol

# Learning vector quantization (LVQ1)

- LVQ1 algoritmus má tyto kroky:
  - Ukaž trénovací vzor  $x$
  - Najdi nejbližší neuron
  - Pokud je trénovací vzor  $x$  klasifikován správně  $\rightarrow$  přiblíž neuron
  - Pokud je trénovací vzor  $x$  klasifikován špatně  $\rightarrow$  oddal neuron
  - S ostatními neurony nehýbej
- V algoritmu vystupuje rychlost učení  $\alpha$ , která určuje velikost posunu neuronu v euklidovském prostoru (může se v průběhu času snižovat)

- a. if  $x$  is classified correctly, use:

$$w_i(t + 1) = w_i(t) + \alpha(t)[x(t) - w_i(t)]$$

- b. if  $x$  is classified incorrectly:

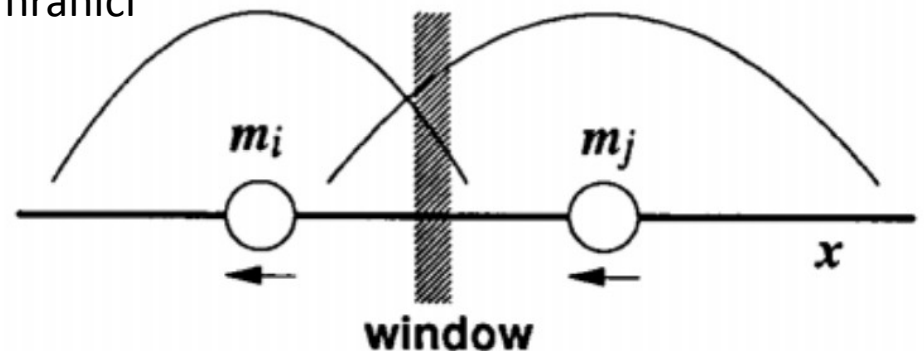
$$w_i(t + 1) = w_i(t) - \alpha(t)[x(t) - w_i(t)]$$

- c. if  $x$  is not the closest neuron:

$$w_k(t + 1) = w_k(t)$$

# Learning vector quantization (LVQ2)

- LVQ2 algoritmus má tyto kroky:
  - Ukaž trénovací vzor  $x$
  - Najdi 2 nejbližší neurony, pro které platí:
    - Neurony jsou kompetitivní
    - Nejbližší neuron musí klasifikovat špatně (LVQ2.1 – toto pravidlo vynechává)
    - Vstup musí být situovaný v pásmu uprostřed mezi kompetitivními neurony a nesmí být příliš blízko jednomu z nich
  - Posuň správně klasifikující neuron směrem k trénovacímu vzoru  $x$  a špatně klasifikující neuron od trénovacího vzoru  $x$
  - Pohybuje se přímo s rozhodovací hranicí



# Zdroj

---

- <https://sci2s.ugr.es/keel/pdf/algorithm/articulo/1990-Kohonen-PIEEE.pdf>