



**České vysoké učení technické v Praze**



**Fakulta elektrotechnická**



**Katedra kybernetiky  
Katedra počítačů**



# Vytěžování dat – přednáška 10

## Neuronové sítě s učitelem

# Osnova přednášky

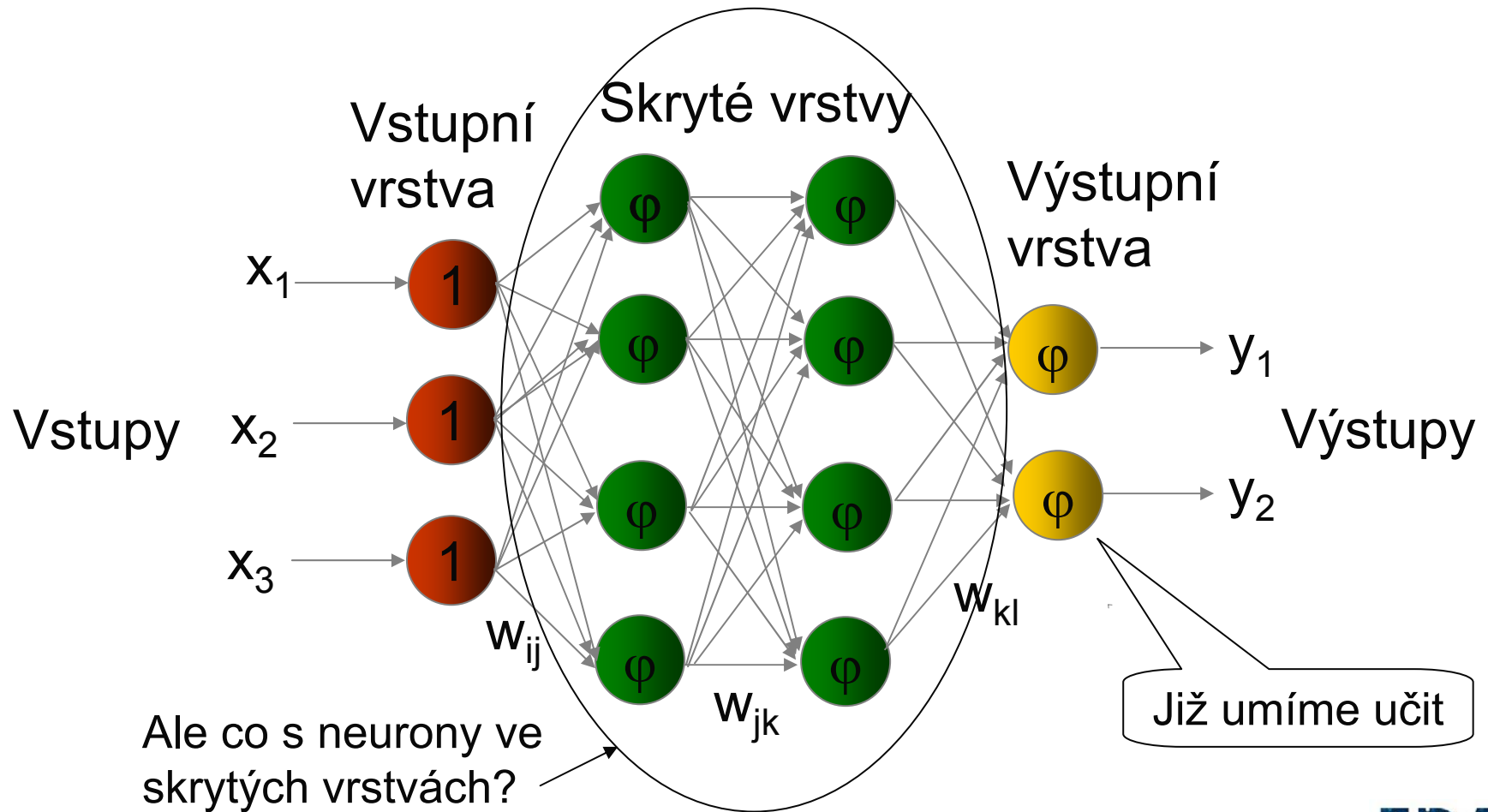
- Učení dopředných neuronových sítí s učitelem
  - Učení MLP - zpětná propagace chyby
  - Učení GMDH
  - Učení RBFN

# Pamatujete si z minula

- Problém – perceptron nedokáže uspokojivě řešit lineárně neseparabilní problémy
- Průlom I – diferenciovatelná aktivační funkce neuronů
- Průlom II – vícevrstvé sítě
- Průlom III – algoritmus učení ... o tom dnes

# Vícevrstvá síť s dopředným šířením

## ■ MLP (MultiLayered Perceptron)



# MP – algoritmus učení (opakujeme)

Pro každý učicí vzor uprav váhy perceptronu:

$$w_i(t+1) = w_i(t) + \eta e(t) x_i(t),$$

$$e(t) = [d(t) - y(t)] ,$$

↘ Chyba, nebo odchylka

V závislosti na chybě uprav hodnoty vah perceptronu.

Odpovědnost za chybu musí převzít i neurony ve skrytých vrstvách!

# Změny ve výstupní vrstvě:

Vstup je zároveň výstup neuronu ze skryté vrstvy

$$\Delta w_{ij}^o(t) = \eta \delta_i^o(t) y_j^h(t)$$

$$\Delta \Theta_i^o(t) = \eta \delta_i^o(t)$$

kde

Chyba, odchylka

*pozor, změna značení (dříve e)*

$$\delta_i^o(t) = (y_i^o - d_i) \cdot S'(\varphi)$$

Derivace sigmoidy, vysvětlím později

Váhy a prahy výstupní vrstvy tedy při učení upravujeme podle vztahů

$$w_{ij}^o(t+1) = w_{ij}^o(t) + \Delta w_{ij}^o(t)$$

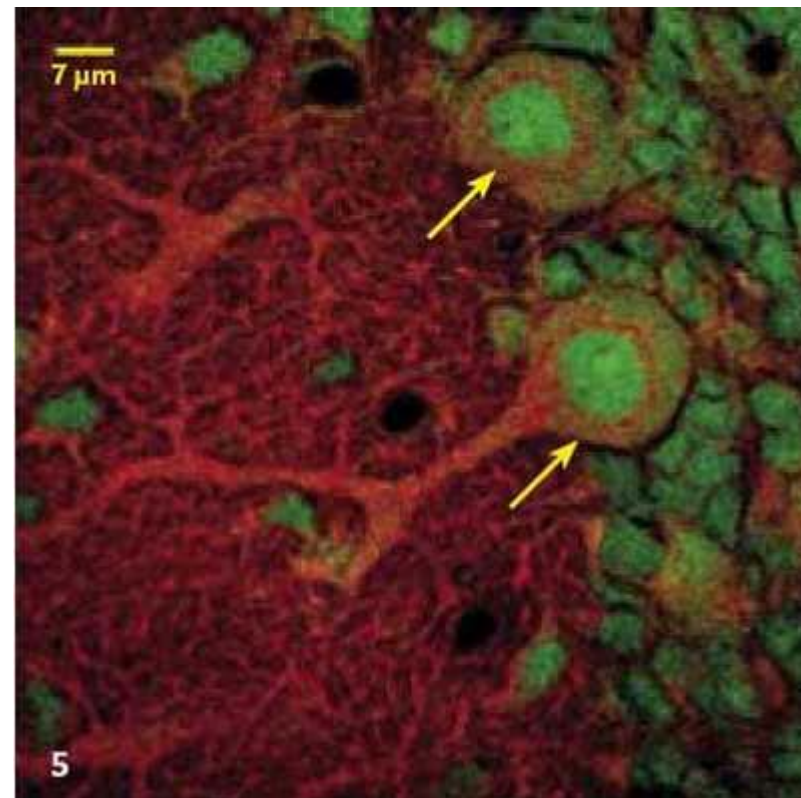
$$\Theta_i^o(t+1) = \Theta_i^o(t) + \Delta \Theta_i^o(t)$$

Jako v případě ML perceptronu



# Biologická inspirace

Název mozeček (lat. cerebellum) není jen zdrobnělina mozku, tedy jakýsi nedostatečně narostlý mozek. Je to významná část centrálního nervového systému, která je uložena v zadní jámě lební za prodlouženou míchou a Varolovým mostem. Stejně jako velký mozek je i mozeček rozdělen na dvě hemisféry, má členitou kůru, pod ní bílou hmotu tvořenou nervovými vlákny a v ní ostrůvky šedé hmoty – mozečková jádra. Ačkoli je mozeček mnohem menší než přední nebo-li velký mozek, jeho kůra obsahuje více neuronů (v porovnání s kůrou velkého mozku), a navíc je propojen se všemi důležitými mozkovými strukturami i s míchou. Proto se významně podílí na řadě nervových funkcí, dokonce i na těch, které byly dříve přisuzovány výhradně velkému mozku.



## Purkyňovy buňky mozečku zdravé myši.

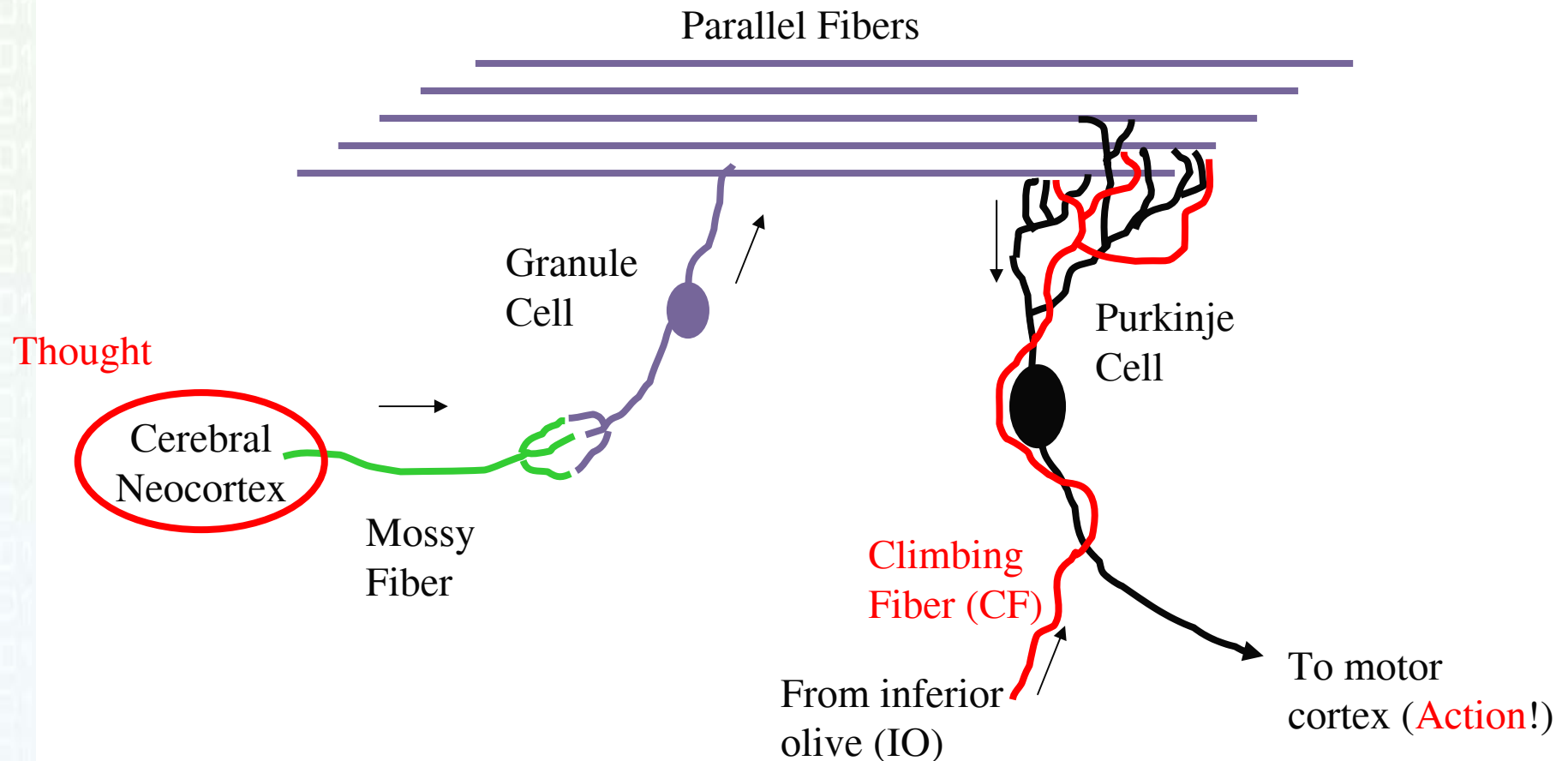
Obrázky barveny dvojitým fluorescenčním barvením, snímány konfokálním mikroskopem Olympus Fluoview. Snímky © Josef Reischig, Referenční pracoviště optické mikroskopie firmy Olympus C&S se sídlem v Ústavu biologie LF UK v Plzni

<http://www.vesmir.cz/clanek.php3?CID=5880>



# Učení s učitelem v mozečku

- IO dostává senzorní informace z kůže, svalů, receptorů, atd. Tyto informace reflektují namáhání těla v důsledku motorických pohybů.
- CF nervová vlákna od IO zajišťují zpětnou vazbu (například signalizují chybu), která přímo vede např. k útlumu-zeslabení synapsí Purkyňových buněk.



# Back Propagation historie



- Paul Werbos,
  - 1974,
  - Harvard,
  - PhD práce.
  
- Dodnes populární metoda, mající mnoho modifikací.

Ve skrytých vrstvách budeme váhy upravovat obdobně:

$$\Delta w_{ij}^h(t) = \eta \delta_i^h(t) x_j(t)$$

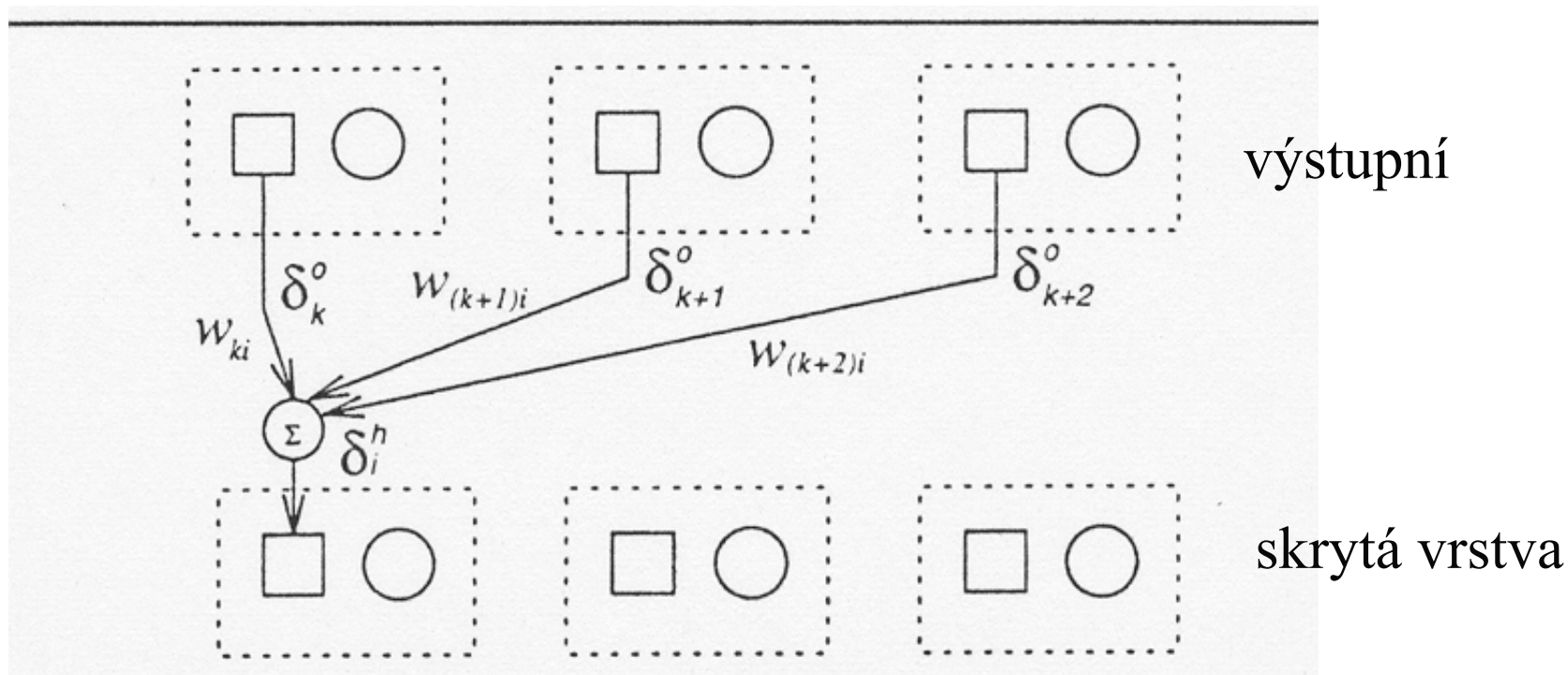
$$\Delta \Theta_i^h(t) = \eta \delta_i^h(t)$$

kde

$$\delta_i^h = y_i^h (1 - y_i^h) \sum_{k=1}^n w_{ki}^o \delta_k^o$$

Jak jsme na to přišli?

# Vysvětlení sumy



Prostřednictvím větvení přispívá tento neuron k celkové energii (chybě) na několika výstupech.

A provedeme změny vah a prahů ...

$$w_{ij}^h(t+1) = w_{ij}^h(t) + \Delta w_{ij}^h(t)$$

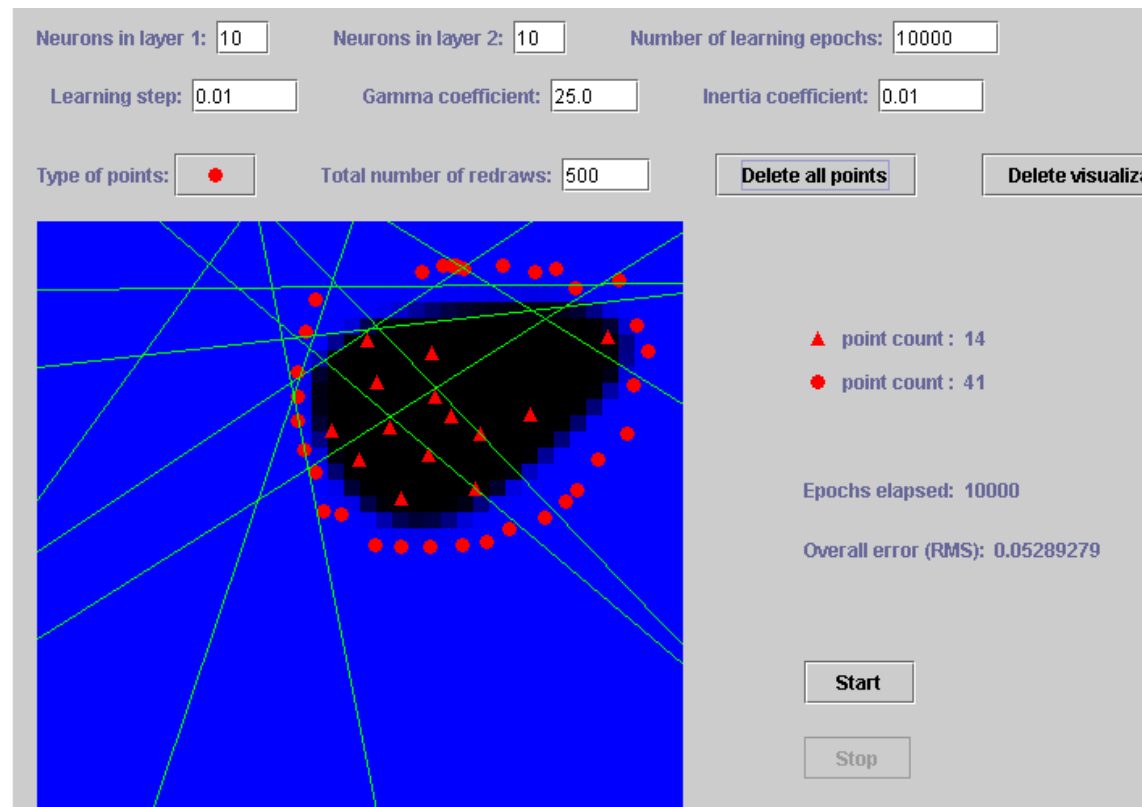
$$\Theta_i^h(t+1) = \Theta_i^h(t) + \Delta \Theta_i^h(t)$$

A co to všechno umí?

- Je z něj patrné, jak se chyba z výstupu šíří celou sítí na vstup.
- Odtud název učicího algoritmu - **zpětné šíření chyby - Back-propagation.**

# MLP - příklad

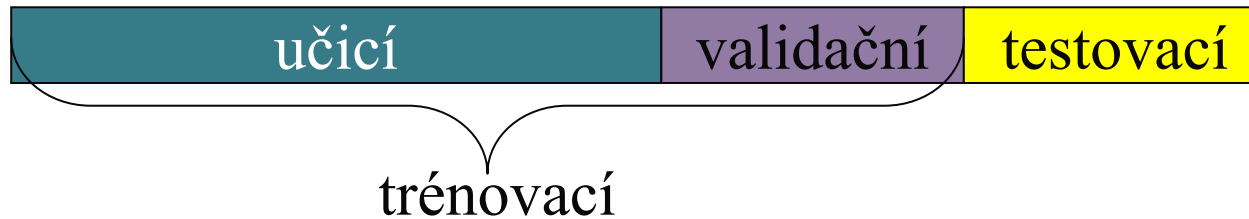
Autor: Jan Horňáček



<http://neuron.felk.cvut.cz/~kordikp/mlp/index.htm>



# Trénovací, validační a testovací množina ... terminologie



- Množina všech nebo vybraných uspořádaných dvojic vstupní vektor - výstupní vektor. Množiny musí **reprezentativním** způsobem popisovat prostor vstupních dat.
- V případě, že je množina vstupních vektorů příliš velká, vybere se (viz výše) podmnožina.
- V průběhu učení lze použít validační množinu pro zastavení učení v optimu (a zabránit tak přeučení).
- Kvalita naučení se kontroluje množinou testovací.

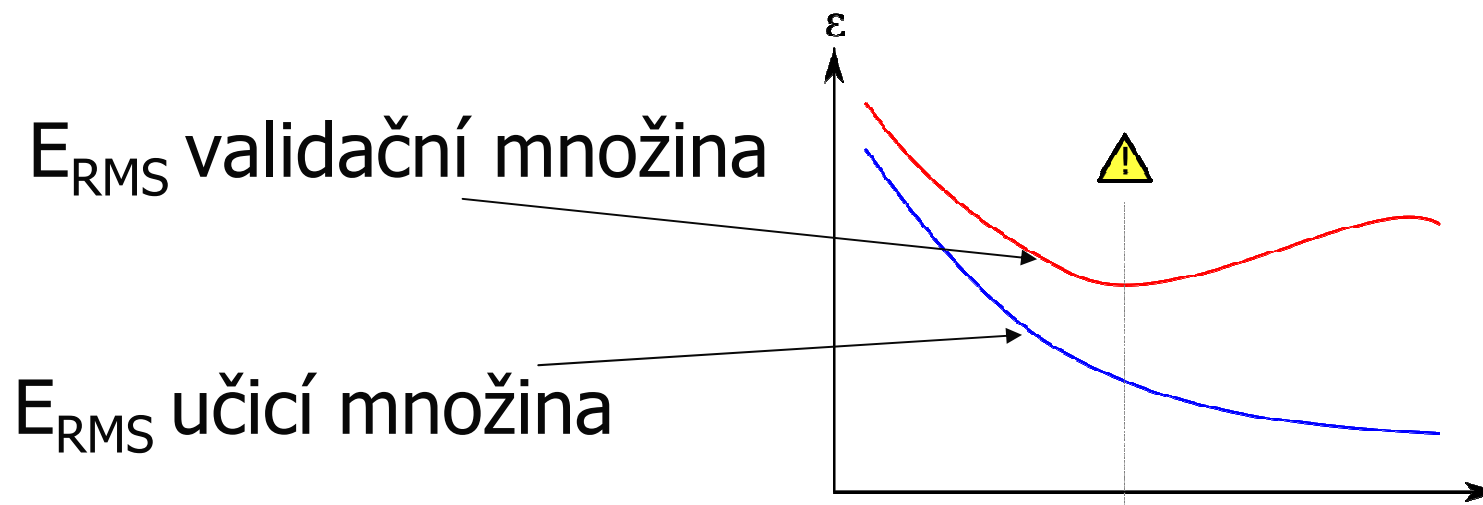
# Ještě jednou a lépe!

- Chceme naučit MLP na učití množině dat tak, abychom minimalizovali její chybu na testovací množině – k tomu nám dopomáhej validační množina.
- Co to znamená minimalizovat chybu?
  - Střední kvadratická odchylka (RMS - Root Mean Squared), počítaná přes všechny vzory množiny

$$E_{RMS} = \sqrt{\sum_n (y_n - d_n)^2},$$

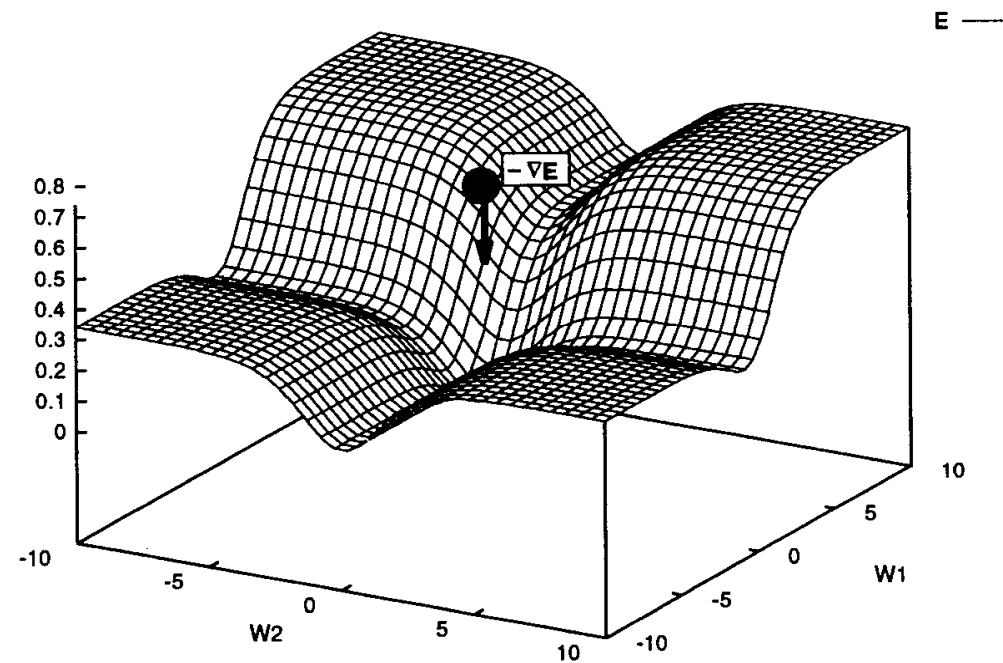
# Cíl: $\min E_{\text{RMS}}$

- Ale na jakých datech?
  - Nejlépe na testovacích, ale ta máme schované
  - Na trénovacích? Může dojít k přeučení
  - Budeme minimalizovat na učicích a sledovat na validačních – včasné ukončení učení



# Cíl: $\min E_{\text{RMS}}$ , ale jak na to?

- Budeme měnit hodnoty vah a prahů neuronů
- Jak? Ve směru nejstrmějšího poklesu  $E_{\text{RMS}}$
- $E_{\text{RMS}}$  se také někdy říká energie sítě, nenaučenost.
- Jen 2 váhy – 2D
- Prostor chyby je složitější kvůli sigmoidě



# „Lidská“ interpretace

Tuto funkci si lze představit jako zakřivenou plochu v hyperprostoru připomínající třídimenziální pohoří. Okamžitá hodnota vah a prahů je souřadnicí, energie bodem na této ploše. Z tohoto bodu se snažíme postupně dosáhnout minima (energie).

Tuto situaci můžeme připodobnit chůzi po horském masívu za husté mlhy, kdy vidíme jen několik metrů před sebe. Když budeme chtít dojít do údolí (minimum energie) půjdeme logicky s kopce, tedy proti směru největšího spádu (gradientu). Stejným způsobem tento problém řeší i metoda BP.

# Cíl učení a metoda k jeho dosažení:

Pro dané hodnoty na vstupech sítě vykazuje energetická funkce určitou hodnotu energie. Naším cílem je tuto hodnotu minimalizovat, a to modifikací váhových koeficientů a prahů. Proto musíme určit o kolik a s jakým znaménkem se změní hodnota energie, když změníme určitou váhu o  $\Delta w$ .

To je možné snadno určit výpočtem z parciální derivace  $\partial E/\partial w$ . Určujeme tedy citlivost změny energie (chyby) na změny vah, popř. prahů. Postupujeme tedy **gradientní metodou**.


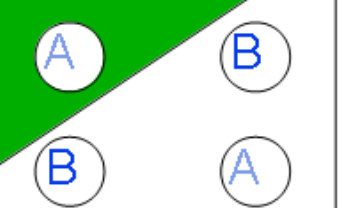


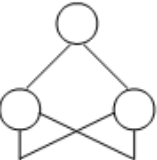
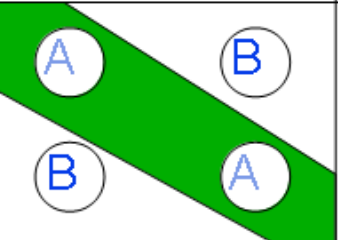
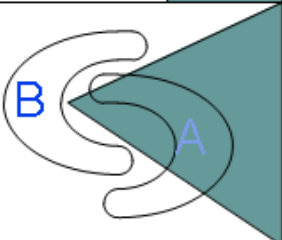
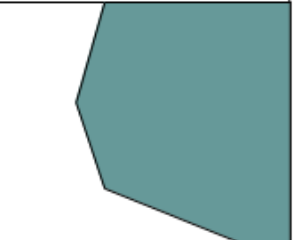
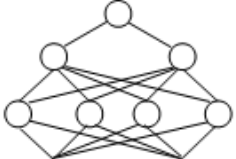
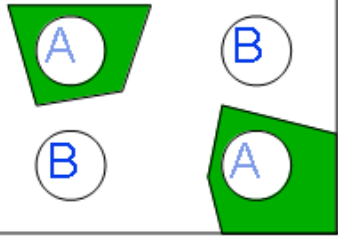

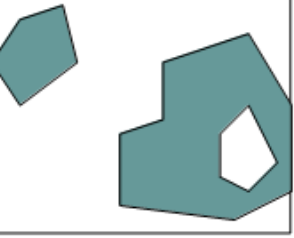
# Co si hlavně zapamatovat?

- Co že se to zpětně šíří?
  - **Chyba!**
- Kam se zpětně šíří?
  - **Z výstupu přes skrytou/té vrstvu/y na vstup!**
- Co upravujeme abychom snížili chybu?
  - **Váhy a prahy neuronů**
- Jak?
  - **Gradientní metodou!**

Pro zájemce pokračování algoritmu učení BP – slajdy na konci prezentace – nebude vyžadováno u zkoušky



# Separáčnı schopnosti MLP v závislosti na počtu jeho vrstev

<i>Structure</i>	<i>Types of Decision Regions</i>	<i>Exclusive-OR Problem</i>	<i>Classes with Meshed regions</i>	<i>Most General Region Shapes</i>
<b>Single-Layer</b> 	<i>Half Plane Bounded By Hyperplane</i>			
<b>Two-Layer</b> 	<i>Convex Open Or Closed Regions</i>			
<b>Three-Layer</b> 	<b>Arbitrary</b> (Complexity Limited by No. of Nodes)			

# Mám data, jakou strukturu MLP zvolit?

Datově závislý problém ...

# GMDH

- Group method of Data Handling.
- Autorem je ukrajinský akademik A.G. Ivachněnko, 1968,
- od ostatních sítí, které zde popisujeme, se výrazně liší:
  - síť vzniká indukcí, někdy říkáme samoorganizací,
- je vícevrstvá s dopředným šířením,
- učí se s učitelem.



## A. G. a G. A. Ivachněkovi





# Gluškovův institut v Kyjevě



# Indukce x dedukce

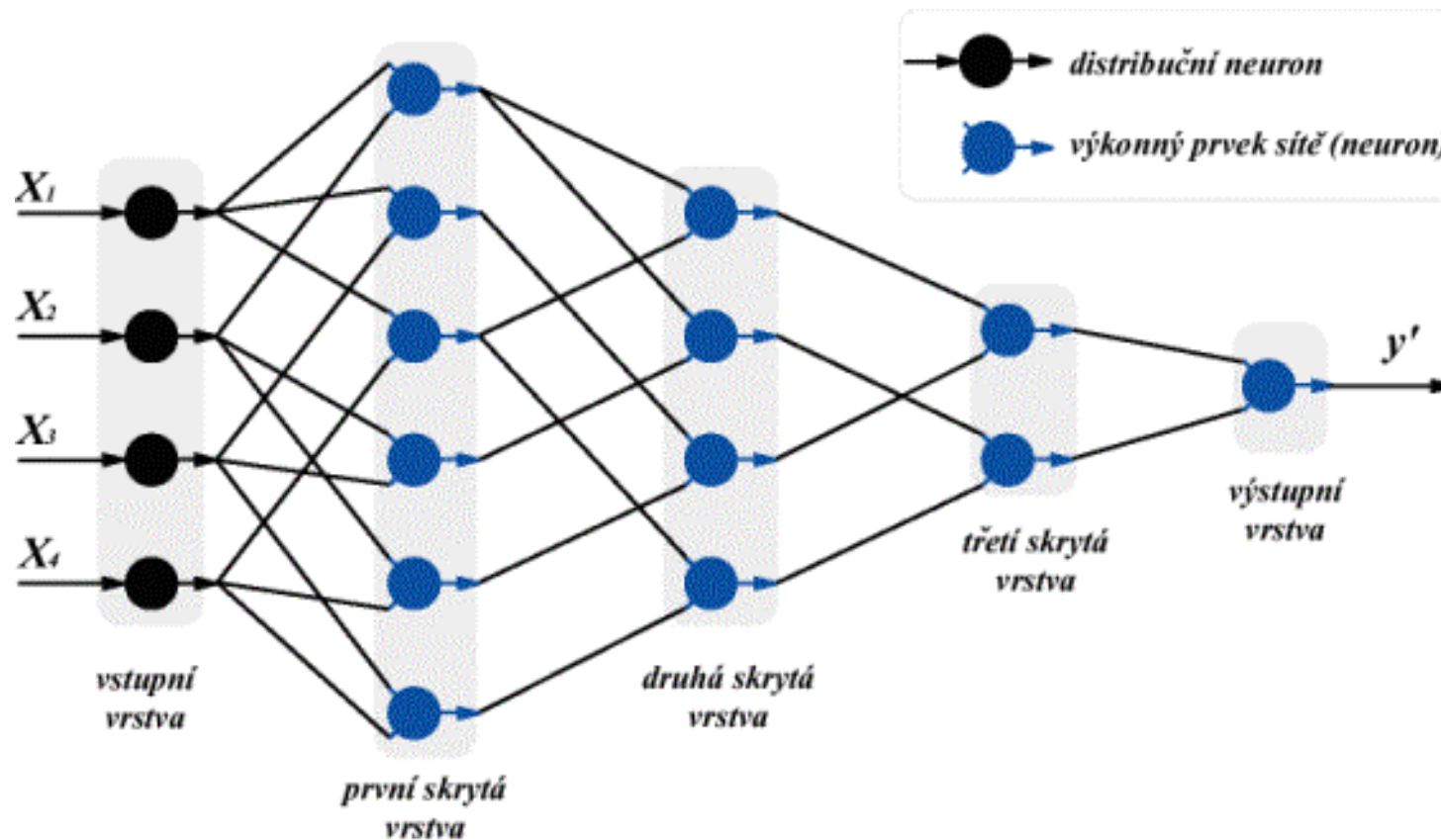
- **Indukce** neboli úsudek je postup od jednotlivostí k obecným závěrům. Používá se např. v matematice jako metoda důkazu.
- **Dedukce** (opak indukce) spočívá v odvozování dílčího, zvláštního ze všeobecného.
- Induktivní modelování = model roste z dat a nepoužívá žádné další externí informace o povaze problému.

# Původní GMDH = GMDH typu **MIA**

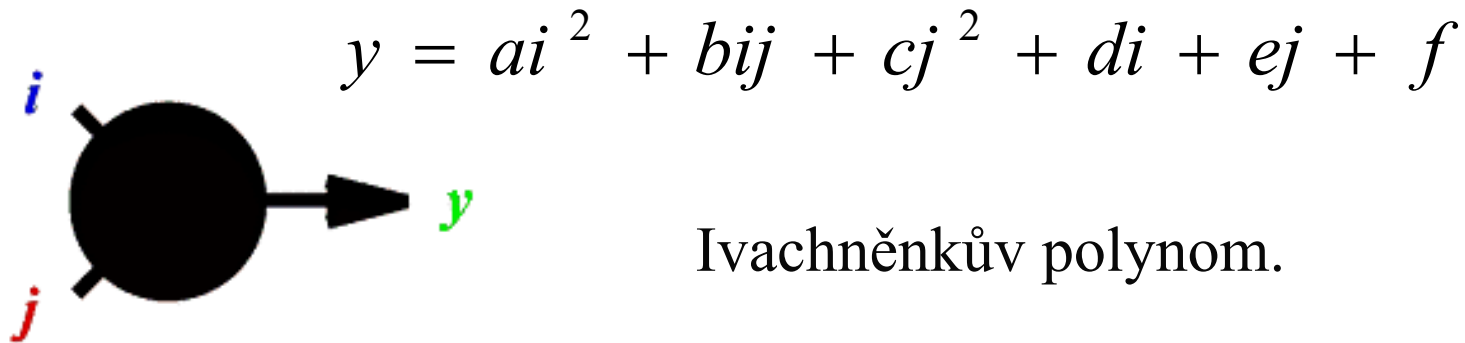
- Patří mezi parametrické sítě
  - MIA (Multilayer Iterative Algorithm), založen na postupné indukci,
  - roste z minimální formy,
  - během učení se nastavují její parametry,
  - perspektivní neurony (jednotky) přežívají,
  - tvorba sítě ukončena, když už přidávání dalších vrstev nezlepšuje přesnost sítě.



# Architektura GMDH typu MIA obecně



# Neuron GMDH typu MIA



$$y = ai^2 + bij + cj^2 + di + ej + f$$

Ivachněnkův polynom.

Příklady jiných Ivachněnkových polynomů:

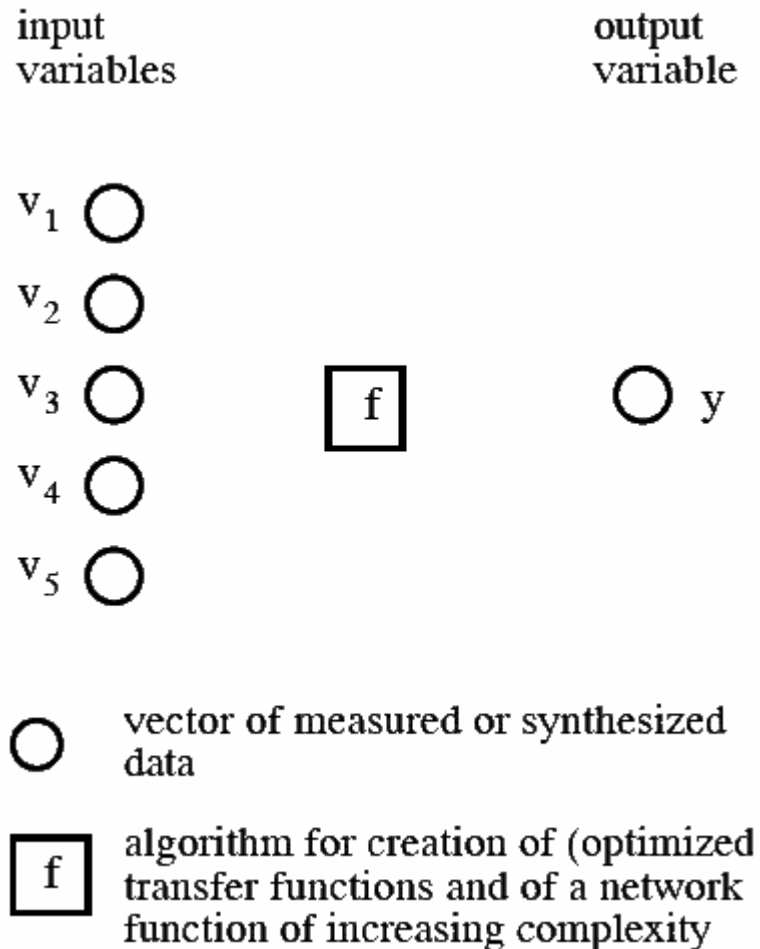
$$y = a + bi + cj,$$

$$y = ai + bjk.$$

# Další charakteristiky

- Sít' MIA se učí s učitelem.
- MIA užívá jediný typ neuronů.
- Průběh učení:
  - sít' vzniká vrstvu po vrstvě,
  - při učení se přidávají skryté vrstvy tak dlouho,
  - dokud není splněno kritérium kvality výstupu.
- Průběh vybavování
  - je jednoduchý, sít' má jediný výstup.

# Proces tvorby GMDH MIA



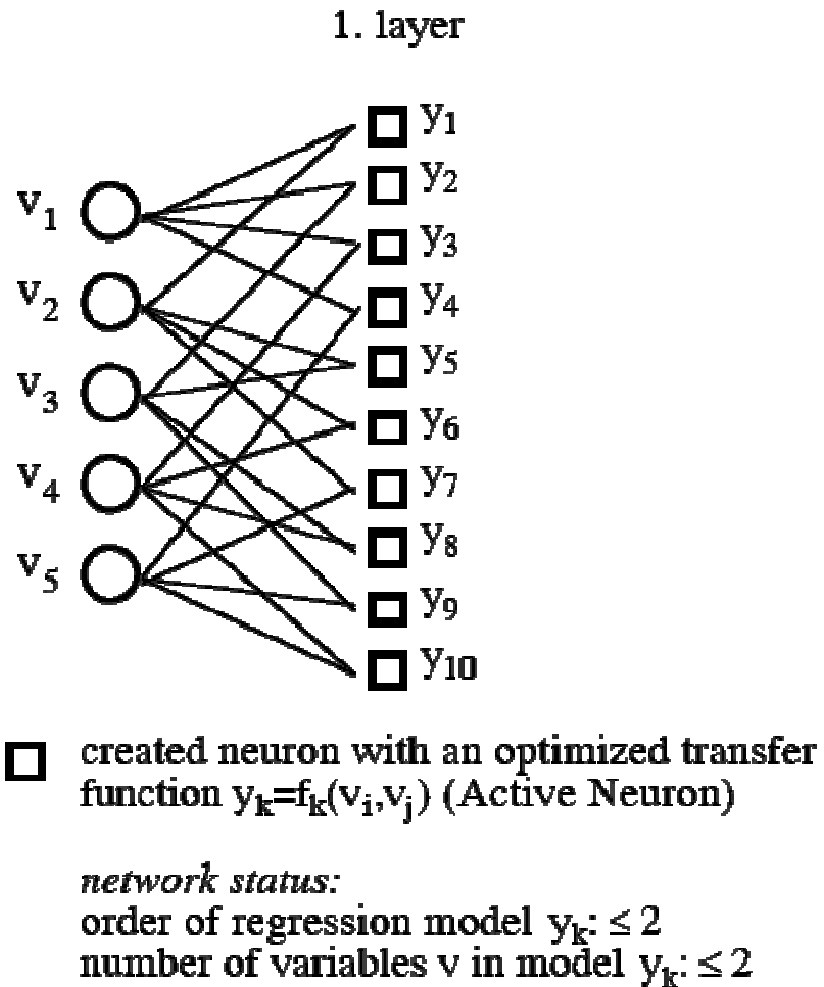
- Máme data v obvyklé formě
  - pro každý vektor hodnot vstupních proměnných  $v_1, \dots, v_5$
  - známe výstup  $y$  (učení s učitelem).
- Konstruujeme síť s přenosovou funkcí  $f$ , která bude modelem systému – pro každý vstupní vektor poskytne výstup blížíící se  $y$

zdroj: Mueller, Lemke: kniha

Self-organising Data Mining

Y336VD Vytěžování dat

# Proces tvorby GMDH MIA



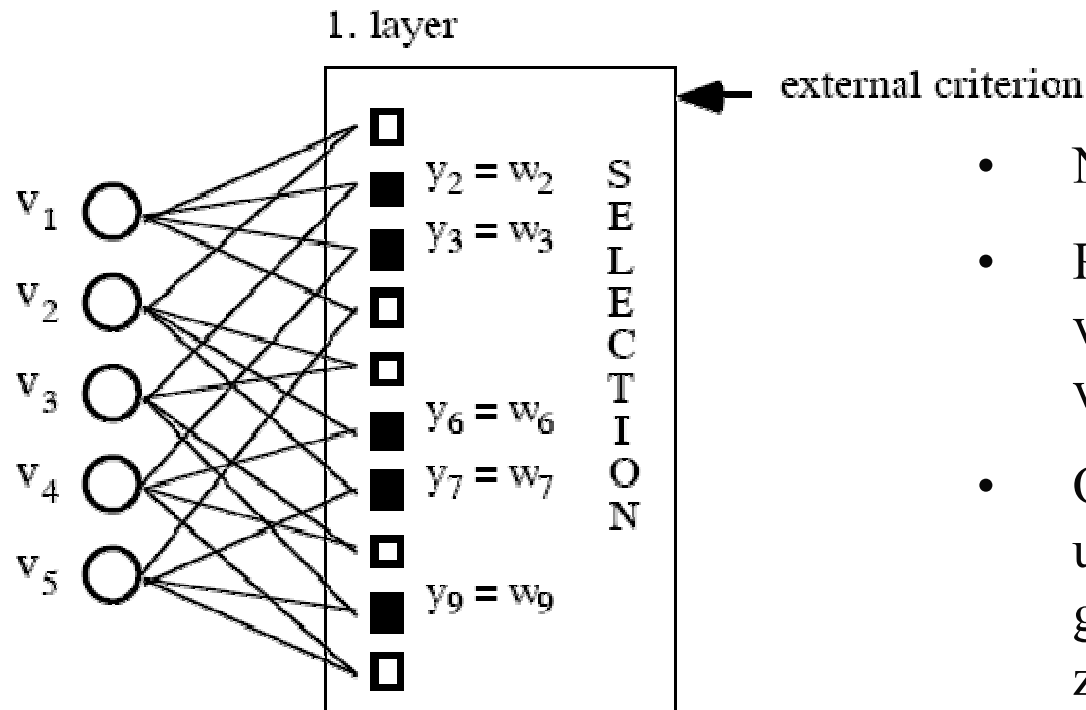
- Tvoříme první vrstvu.
- Každý neuron (jednotka) se snaží co nejlépe určit  $y$  ze dvou vstupů, ke kterým je připojen.
- Neurony které vygenerujeme nazveme počáteční populace.
- Generuje tolik neuronů, kolik je všech možných kombinací dvojic vstupů (pairwise combinations).
- Co když je vstupů 500?

zdroj: Mueller, Lemke: kniha

Self-organising Data Mining

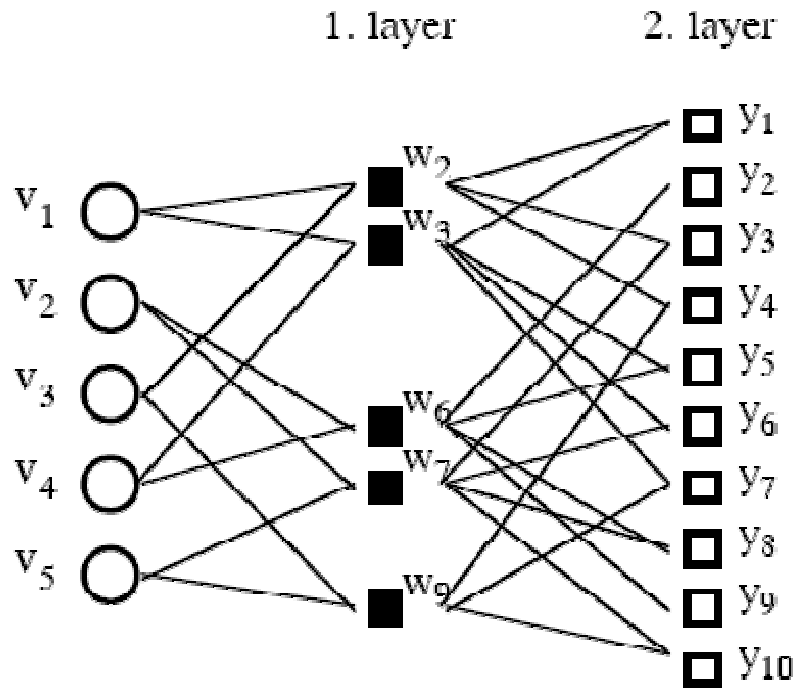
Y336VD Vytěžování dat

# Proces tvorby GMDH MIA



- Následuje proces selekce.
- Podle kritéria (viz dále) vybereme neurony, které ve vrstvě zachováme.
- Ostatní neurony zrušíme – umírají. Analogie s genetickými algoritmy, ale zde je jen jedna generace.
- Vybrané neurony ve vrstvě zmrazíme – dále už se nebudou měnit.

# Proces tvorby GMDH MIA



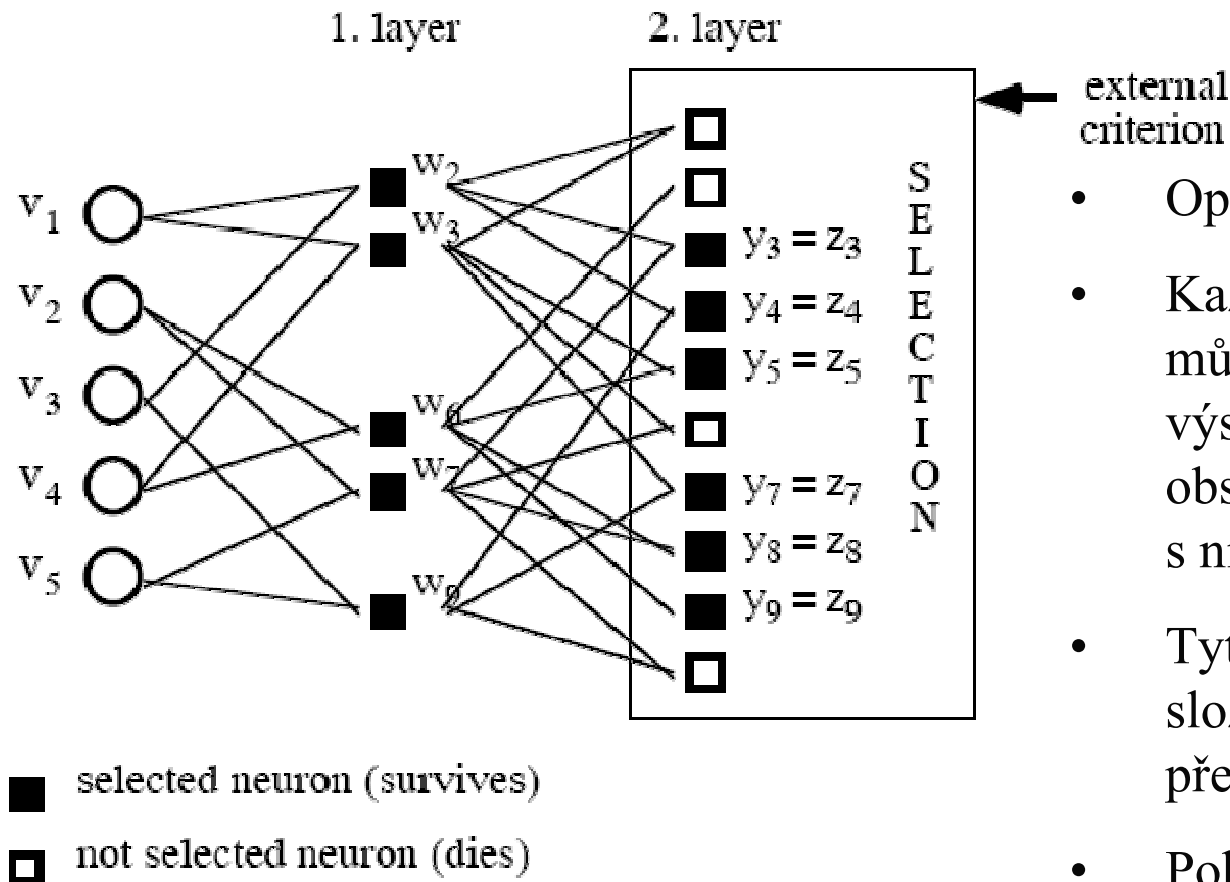
Přenosová funkce modelů se zesložitňuje...

Každý neuron může být připojen až ke 4 různým vstupním neuronům.

- Přidáme další vrstvu.
- Vytvoříme počáteční populaci připojenou k neuronům předchozí vrstvy.
- Tyto neurony pro nás vlastně předzpracovávají data ze vstupní vrstvy.
- Každý neuron 2. vrstvy můžeme chápat jako výstup modelu obsahujícího neurony sítě, s nimiž je spojen.

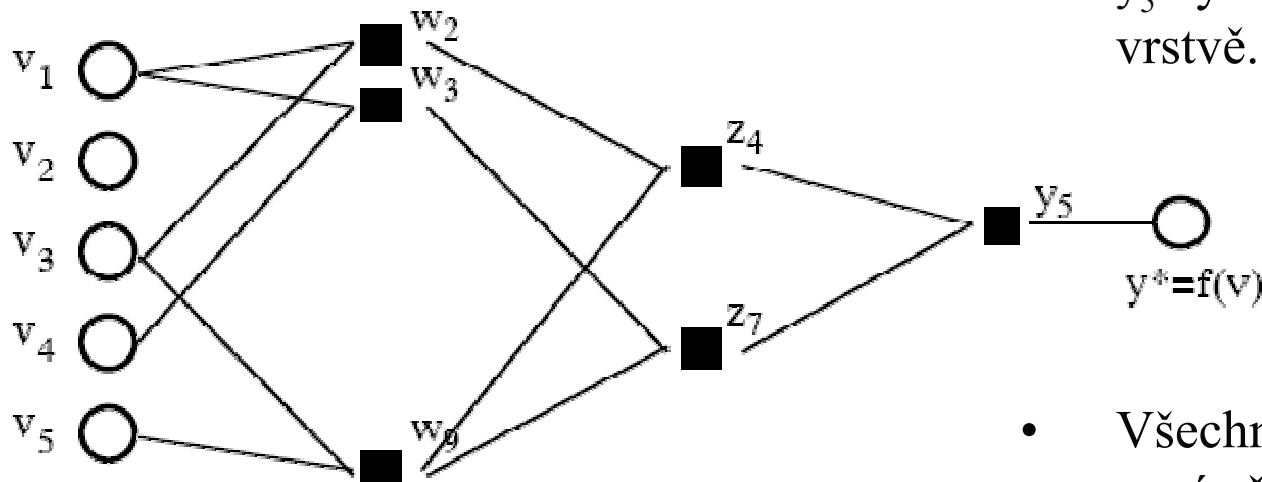


# Proces tvorby GMDH MIA



- Opět selekce ...
- Každý neuron 2. vrstvy můžeme chápat jako výstup modelu obsahujícího neurony sítě, s nimiž je spojen.
- Tyto modely – stejně složité – spolu bojují o přežití.
- Pokračujeme v přidávání vrstev, dokud je to výhodné (viz dále).

# Proces tvorby GMDH MIA



- Optimální model – model  $y_5$  vyhrál v poslední vrstvě.
- Všechny neurony, k nimž není připojen, jsou smazány.

explicit analytically available optimal complex model.

*network status:*

order of regression model  $y^*$ :  $\leq 8$

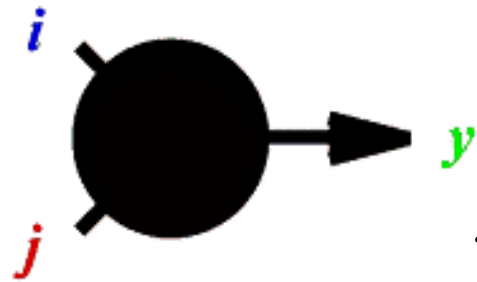
number of variables  $v$  in model  $y^*$ : 4 (from initially 5 variables)

# Ukončení učení

$F$  = chyba nejlepšího neuronu vrstvy na validačních datech



# ? Otázka ?



Ivachněnkův polynom.

$$y = ai^2 + bij + cj^2 + di + ej + f$$

- Jakým způsobem vypočteme každému neuronu šestici koeficientů tak, aby co nejlépe modeloval  $y$ ?

(pro jednoduchost uvažujte nejdříve neuron v první vrstvě)

# Selekce neuronů aktuální vrstvy

- Šestice koeficientů vyhovuje těm vstupním vektorům, z nichž jsme je počítali,
- dramaticky ale nemusí vyhovovat jiným.
- Proto neurony např. seřadíme sestupně, podle dosažené RMS na validačních datech,
- V každé vrstvě ponecháme jen několik „nejlepších“.

# Dají se neurony vybírat i podle jiných kritérií?

- Ano, existuje jich mnoho, můžeme brát v úvahu chyby na učící, validační množině, rozptyl výstupu vůči šumu v datech, podobnost s ostatními neurony ...
- viz dále ..

# Kritéria výběru neuronů ve vrstvě

$$\frac{N}{N-p} s_e^2 \quad S_{e,p}^2 \quad s_e^2 + s_y^2 \ln N \quad \frac{p}{N} \quad \text{Schwarz}$$

$$AB = \frac{1}{N_B} \sum_{i=1}^{N_B} [y_i - \hat{y}_i(A)]^2 \rightarrow \min. \quad \text{Kritérium regularity}$$

$$\frac{N+p}{N-p} s_e^2 \quad FPE \quad N \ln s_e^2 + p \ln N \quad \text{Rissanen}$$

$$\delta^2 = \frac{\sum_{i=1}^N (y_i - \hat{y}_i)^2}{\sum_{i=1}^N (y_i - \bar{y})^2} < 1.0 \quad \text{Diskriminační kritérium}$$

$$s_e^2 + 2\sigma_p^2 \frac{p}{N} \quad PSE \quad \frac{2ps_e^2}{N-p} - Ns_{y^m}^2 \quad \text{Kocerka}$$

Validační kritérium

$$\frac{Ns_e^2}{\sigma_c^2} + 2p - N \quad \text{Mallow } C_p \quad \frac{\sum_{t=1}^N (\bar{y}_t^m - y_t)^2}{\sum_{t=1}^N y_t^2} \quad i^2(N)$$

$$R_{ij} = 1 - \sqrt{\frac{\frac{1}{N-p_i} \sum_{t=1}^N (y_t - y_t^i)^2}{\frac{1}{N-p_j} \sum_{t=1}^N (y_t - y_t^j)^2}}$$

$$N \ln s_e^2 + 2p + c \quad \text{AIC}$$

$s_e^2$

$$N \ln s_{e,p}^2 + p \ln \left( \frac{s_{y^m}^2}{s_{e,p}^2} \frac{N}{p} \right) \quad \text{BIC}$$

$$\left[ 1 - \lambda \sqrt{\frac{k(\ln(\frac{N}{k}) + I) - \ln \alpha}{N}} \left( 1 - \frac{I}{4} \frac{\ln(\frac{N}{k}) + I - \ln \alpha}{N} \right) \right] \quad \text{Vapnik}$$

$$s_e^2 = \frac{1}{N} \sum_{t=1}^N (y_t - y_t^m)^2 = \frac{1}{N} \sum_{t=1}^N e_t^2; \quad s_y^2 = \frac{1}{N} \sum_{t=1}^N (y_t - \bar{y})^2; \quad s_{y^m}^2 = \frac{1}{N} \sum_{t=1}^N (y_t^m)^2$$

zdroj: Mueller, Lemke: kniha



## 1. Постановка задачи

$$y(D,k) = \overset{\circ}{y}(D,k) + \overset{\circ}{\xi}(D,k) = \sum_{j=1}^{m(D,k)} \overset{\circ}{\theta}_j(k) \overset{\circ}{x}_j(D,k) + \overset{\circ}{\xi}(D,k), k = 1, 2, \dots, h$$

$$\overset{\circ}{y}(D,k) = \overset{\circ}{\mathbf{y}}(D,k) + \overset{\circ}{\xi}(D,k) = \overset{\circ}{\mathbf{X}}(D,k) \overset{\circ}{\boldsymbol{\theta}}(k) + \overset{\circ}{\xi}(D,k), k = 1, 2, \dots, h$$

$$E\{\overset{\circ}{\xi}(D,k)\} = \mathbf{0}_{n(D)}, E\{\overset{\circ}{\xi}(D,k) \overset{\circ}{\xi}^T(D,k)\} = \sigma_{kk} \cdot \mathbf{I}_{n(D)}, k = 1, 2, \dots, h$$

$$E\{\overset{\circ}{\xi}(D,k) \overset{\circ}{\xi}^T(D,q)\} = \sigma_{kq} \cdot \mathbf{I}_{n(D)}, k, q = 1, 2, \dots, h, k \neq q$$

$$E\{\overset{\circ}{\xi}_{i_1}(D,k) \overset{\circ}{\xi}_{i_2}(D,k)\} = 0, i_1, i_2 = 1, 2, \dots, n(D), E\{\overset{\circ}{\xi}(I,k) \overset{\circ}{\xi}^T(II,q)\} = 0, I, II = 1, 2, \dots, h, I \neq II$$

Konference IWIM, Gluškovův institut v Kyjevě

# Nejčastěji se používá v moderních implementacích GMDH:

*Kritérium regularity*

$$AB = \frac{1}{N_B} \sum_{i=1}^{N_B} [y_i - \hat{y}_i(A)]^2 \rightarrow \min.$$

$\hat{y}_i(A)$  ... výstup neuronu, jehož koeficienty byly naučeny na množině A

Seřadím neurony ve vrstvě podle jejich chyby na validačních datech (B).

To, které neurony použiji, rozhodnu podle diskriminačního kritéria.

*Diskriminační kritérium*

$$\delta^2 = \frac{\sum_{i=1}^N (y_i - \hat{y}_i)^2}{\sum_{i=1}^N (y_i - \bar{y})^2} < 1.0$$

$y_i$  ... očekávaný výstup

$\bar{y}$  ... průměrný výstup

$\hat{y}_i$  ... výstup modelu

Chyba modelu ku rozptylu dat -> min

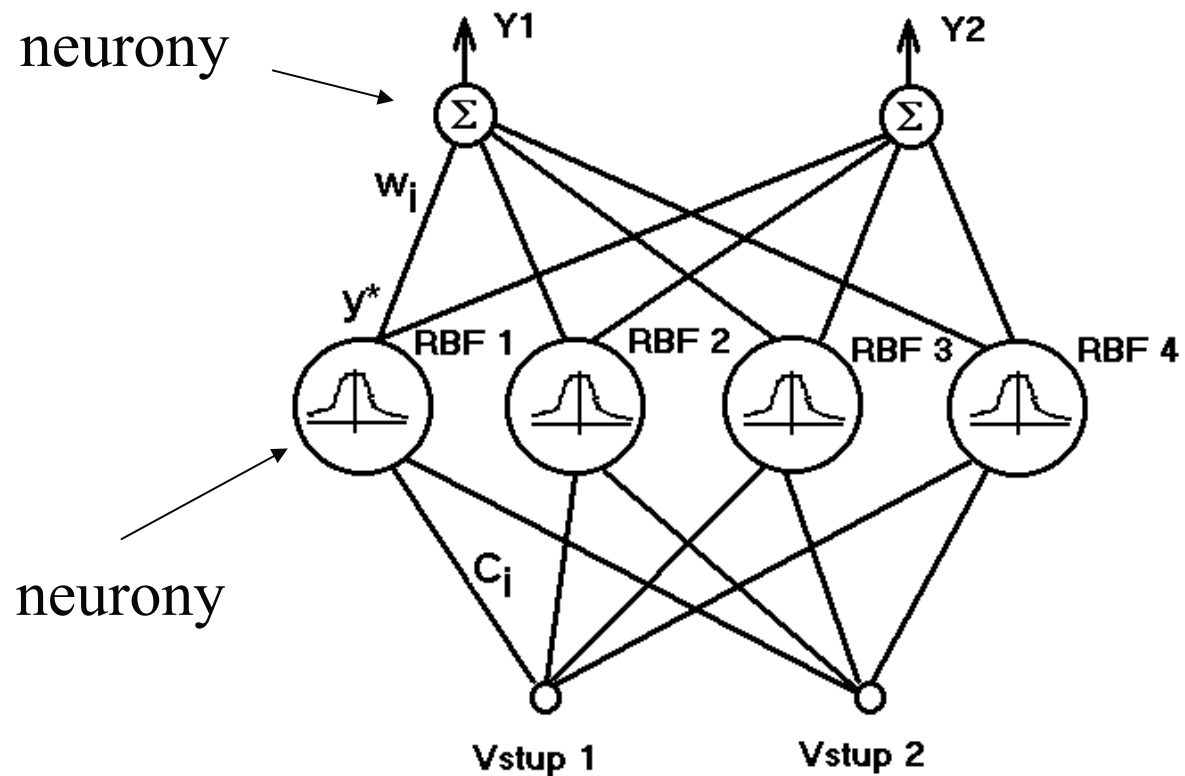
# GMDH není jen MIA

- Za téměř 40 let existence byla teorie GMDH rozšiřována vědci „východního bloku“ – Ivachněnkové, Štěpaško, Mueller, atd.
- Vyvinuté metody respektují základní filozofii GMDH – indukce, samorganizace, růst z minimální formy, regularizační kritéria.
- Mezi **parametrické metody** GMDH patří:
  - **COMBI** (Combinatorial Algorithm)
  - **MIA** (Multilayered Iterative Algorithm)
  - TMNN (Twice-Multilayered Neural Nets)
- Jsou však i jiné varianty GMDH, třeba ty **neparametrické**:
  - **OCA** (Objective Cluster Analysis),
  - **AC** (Analog Complexing),
  - Fuzzy (Self-Organizing Fuzzy Rule Induction).

# RBFN

- Radial Basis Function Network
- 1988, Bromhead, Lowe
- Neuronová síť
- Učení s učitelem
- Použití:
  - Klasifikace
  - Regrese
- Hlavní rozdíl oproti MLP – lokální jednotky (vysvětlíme dále)

# Architektura RBF sítě



# Jak vypadá „sféra vlivu“

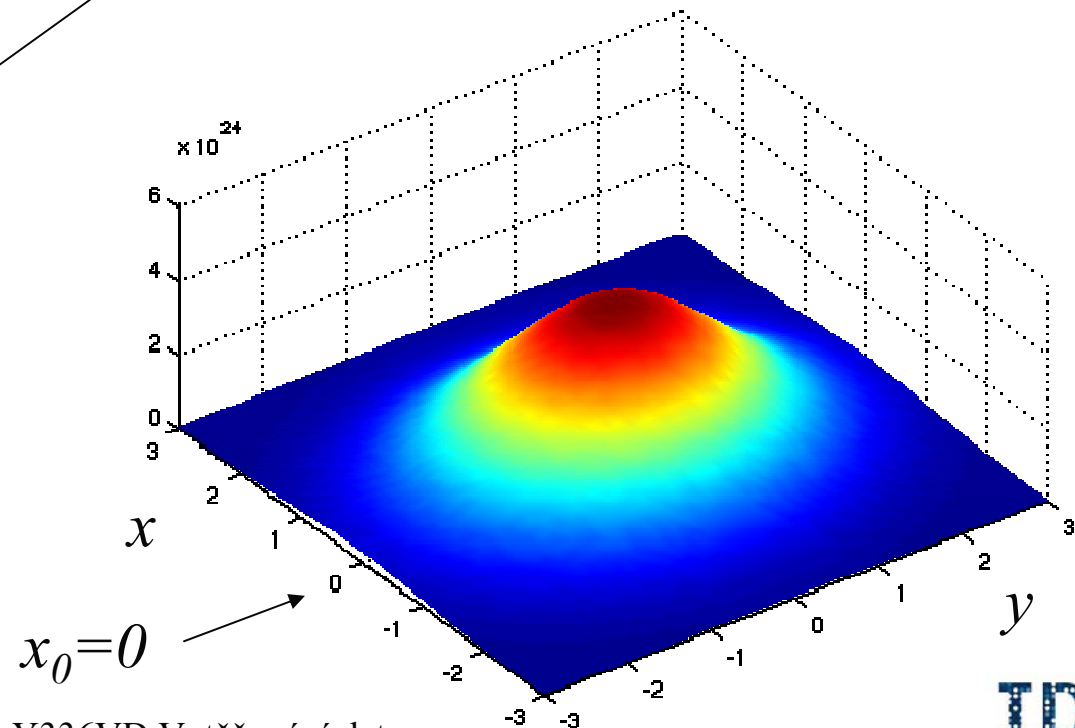
- Většinou gaussovská funkce (kernel)

$$f(x, y) = Ae^{-\left(\frac{(x-x_0)^2}{2\sigma_x^2} + \frac{(y-y_0)^2}{2\sigma_y^2}\right)}$$

amplituda

rozptyl

posuv

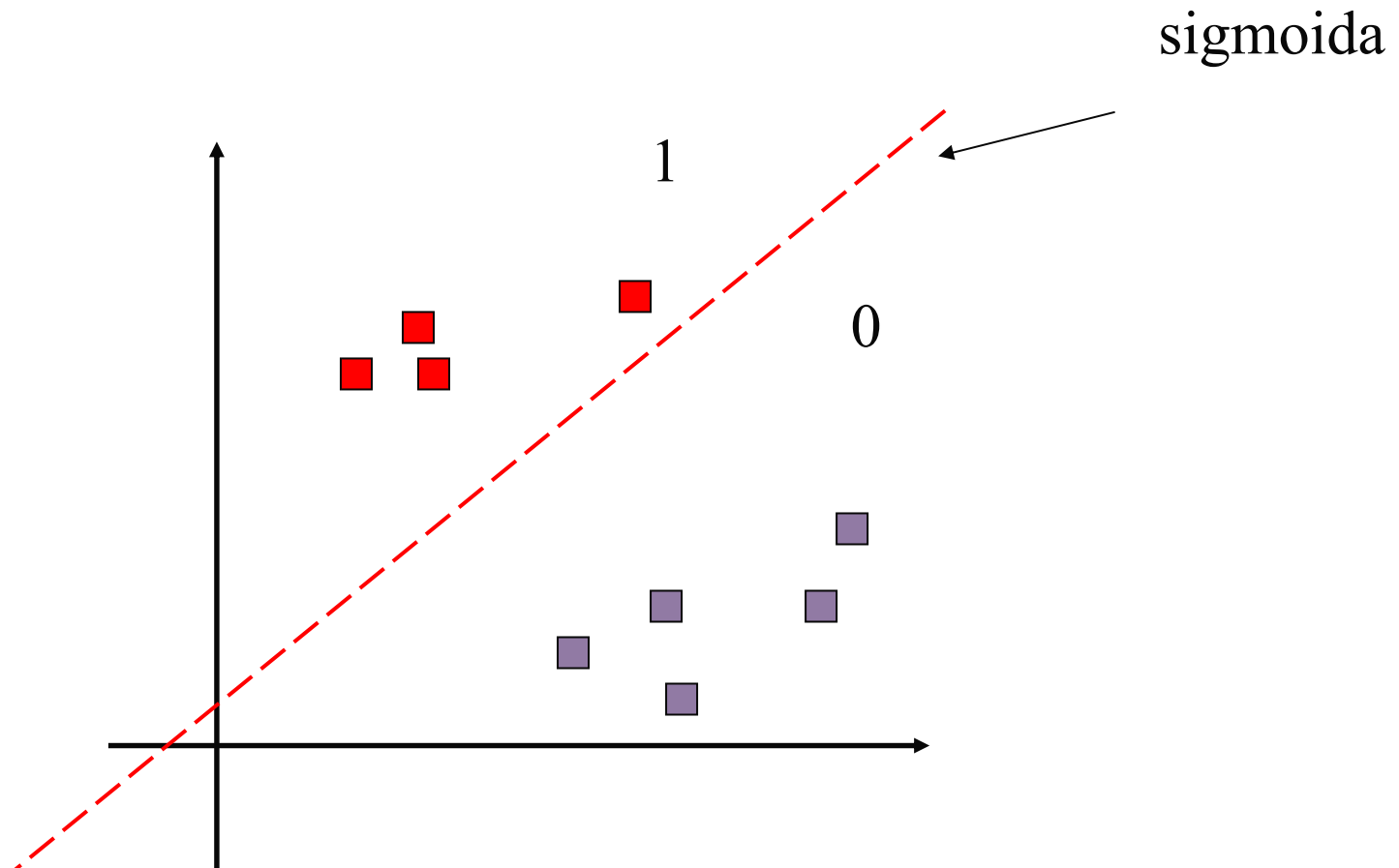


# Lokální jednotky

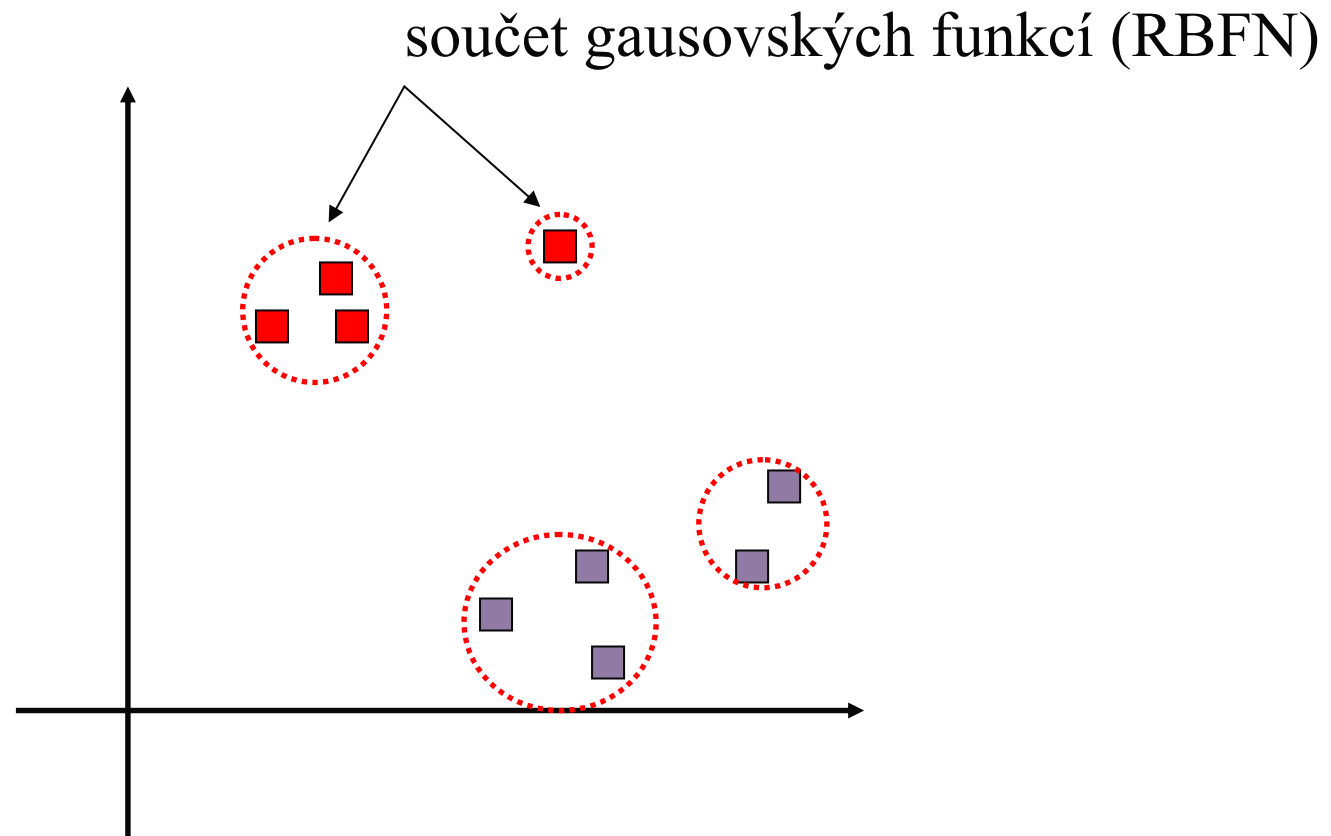
- Co to znamená?
  - pokrývají jen část definičního oboru
  - jsou nenulové jen v jistém úseku
- Globální versus lokální jednotky:
  - gausovská funkce – lokální
  - sigmoida – globální
  - lineární funkce – globální
  - polynom – globální, ale ve speciálních případech může fungovat jako lokální



# Klasifikace pomocí globálních jednotek

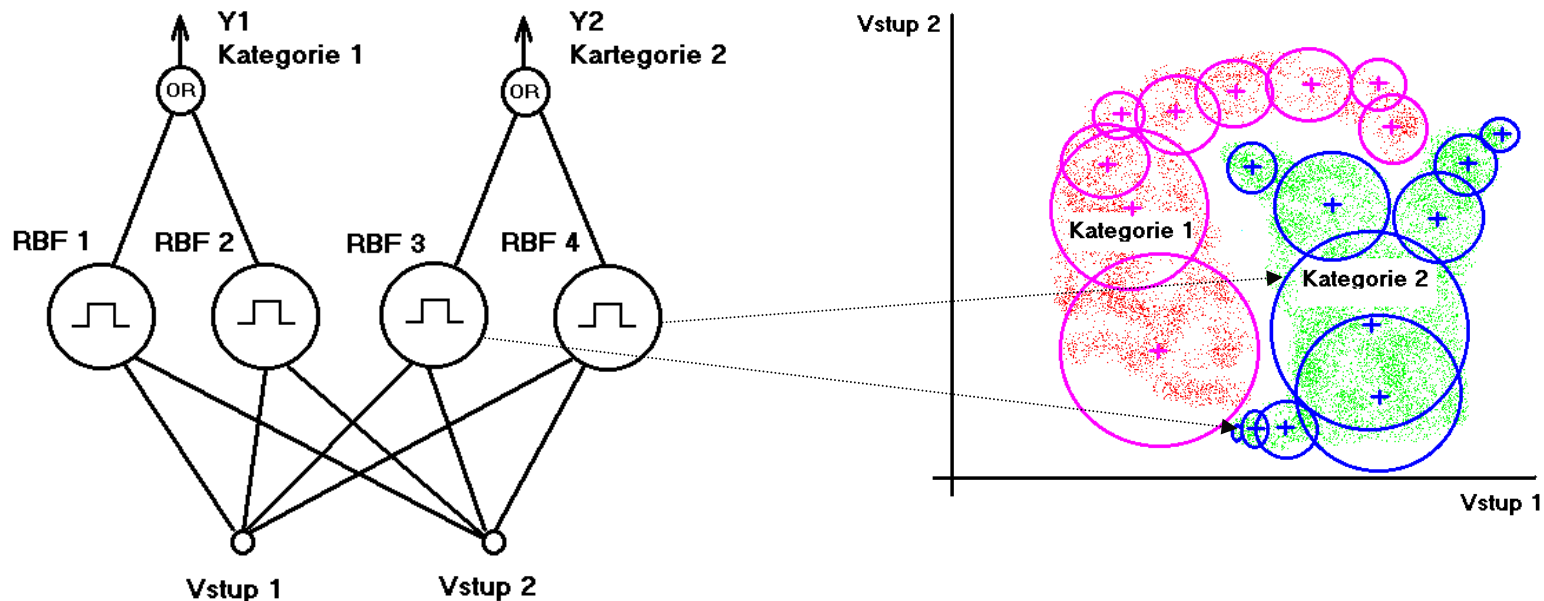


# Klasifikace pomocí lokálních jednotek



20013627 표현아

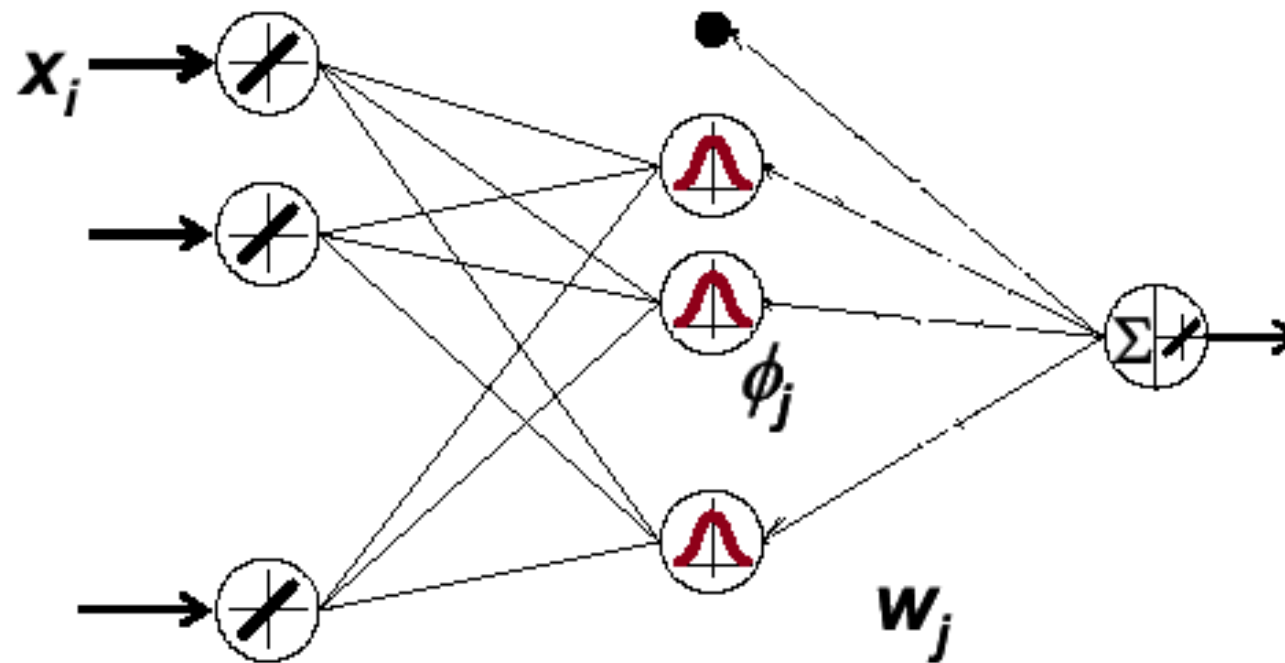
# RBFN jako klasifikátor



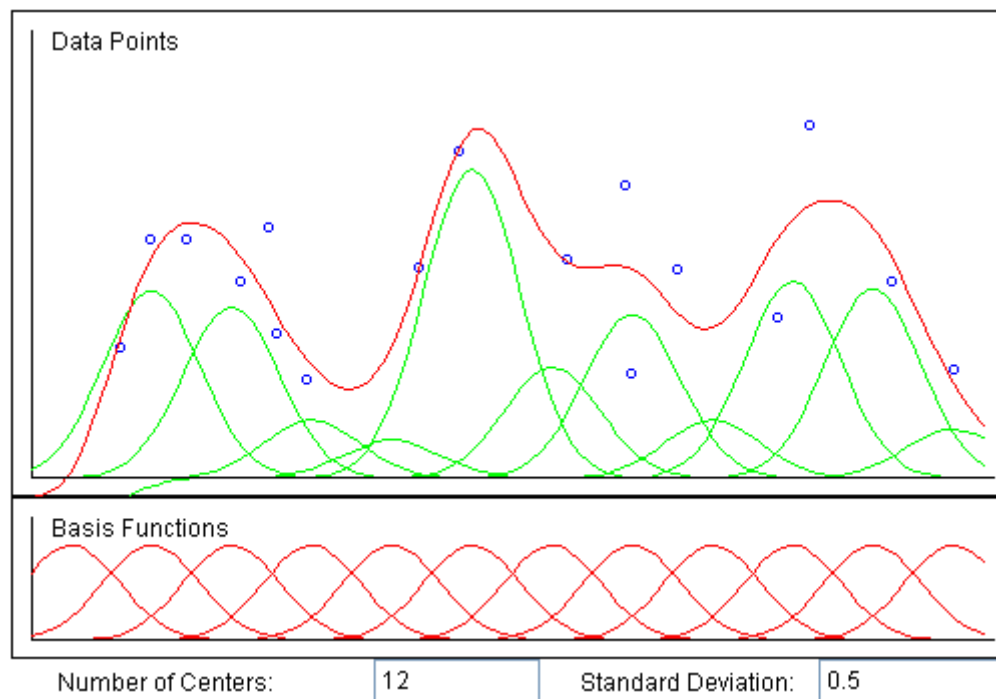
Každý neuron ve vnitřní vrstvě má „sféru vlivu“

Ty se ve výstupní vrstvě vážení sčítají pro každou třídu zvlášť

# RBFN pro aproximaci (regrese)



# RBF síť jako univerzální aproximátor



<http://diwww.epfl.ch/mantra/tutorial/english/rbf/html/>

# Neurony RBF sítě

- **Skrytá vrstva,**

- vnitřní potenciál

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

- lokální nelineární aktivační funkce

$$y = f(\phi), \text{ např. gaussovská}$$

- **výstupní vrstva,**

- lineární přenosová funkce  
(vážený součet)

$$y = \sum_{i=1}^n w_i y_i^*$$

# Diskuse architektury

- RBF neurony:
  - vnitřní potenciál je mírou vzdálenosti vstupního vektoru a **středu** (reprezentovaného vahami neuronu),
  - aktivační funkce vymezuje **sféru vlivu**.
- Výstupní neurony:
  - nasčítávají přírůstky, tak aby požadovaná aproximace byla co nejpřesnější.



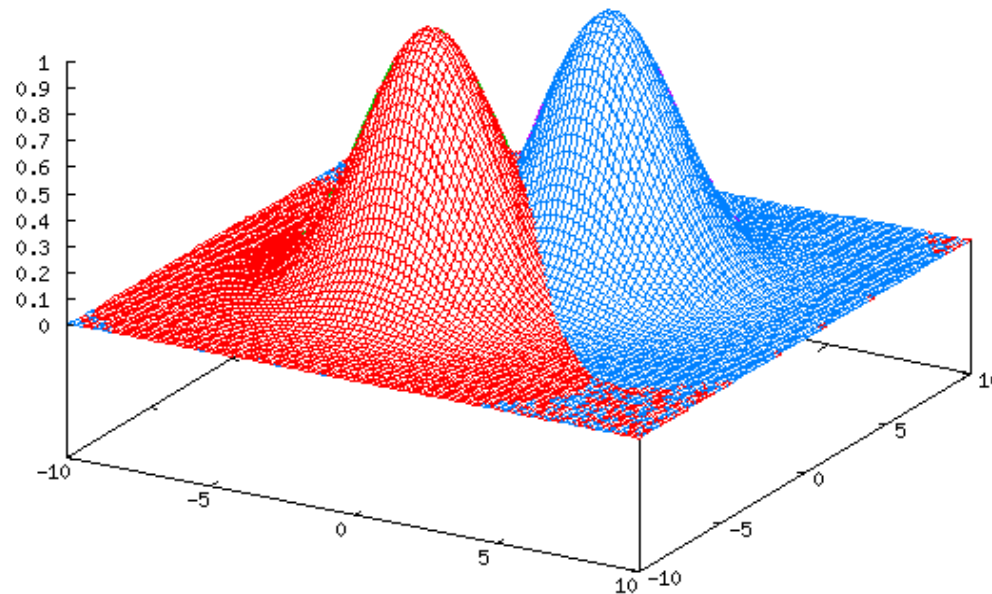
# Sféra vlivu

- Hyperkoule se středem  $C$  a poloměrem  $R$ ,
- RBFN používá pro její určení Eukleidovskou metriku,
- prototyp reprezentuje jistou podmnožinu vstupních dat ve tvaru shluku,

# Sféra vlivu - určení

- Nejčastěji se používá Gaussova funkce známá ze statistiky.
  - Pokud je vstupní vektor totožný s prototypem (tj.  $\varphi = 0$ ), nabývá tato funkce maxima, které dosahuje hodnoty jedna. To je také maximální hodnota aktivity neuronu.
- Se zvětšující se vzdáleností od prototypu aktivita neuronu klesá. Parametr  $\sigma$ , jenž je analogií rozptylu normálního rozdělení, určuje strmost aktivační funkce.

# Sféra vlivu - geometrická představa



# Diskuse

- Gaussova funkce vyjadřuje míru příslušnosti vzoru ke středu.
  - Je-li výstup neuronu blízký jedničce, pak je také vzor velmi podobný středu.
- Podobnost vyhodnocujeme pomocí metrik, které už důvěrně známe

# MLP vs RBFN

Globální plocha	Lokální oblasti
Méně neuronů	Většinou více neuronů
Rychlá vybavovací fáze	Pomalejší vybavování
Pomalá učicí fáze	Rychlé učení

Vhodná data pro MLP a RBFN?

# Učení RBF neuronových sítí

- Připomenutí:
  - jedná se o učení s učitelem, existují tedy dvojice
    - vzor x kategorie (klasifikátor),
    - argument funkce x funkční hodnota (aproximátor).
- Dvě fáze učení:
  - učení prototypů,
  - učení výstupních neuronů.

# Učení středů I

- Předem odhadneme počet shluků ve vstupních datech,
- definujeme funkci příslušnosti  $m$  vzoru ke shluku,
- odhadneme souřadnice všech  $p$  vektorů  $C_p$  které jsou středy shluků.



# Učení prototypů I - pokračování

## Kroky K-means algoritmu:

- . Náhodně inicializuj středy RBF neuronů  $C$ .
- . Vypočítej  $m()$  pro všechny vzory z trénovací množiny.
- . Vypočítej nové středy  $C$  jako průměr všech vzorů, které náležely ke středu  $k$  podle funkce příslušnosti.
- . Ukonči, jestliže se  $m()$  nemění, jinak pokračuj bodem 2

# Učení prototypů II - pokračování

## Kroky adaptivního K-means algoritmu:

- . Náhodně inicializuj středy RBF neuronů  $C$ .
- . Přečti vzor  $X$ .
- . Urči k němu nejbližší nejbližší střed a změň jeho polohu podle pravidla:

$$\bar{C}_k^{(t+1)} = \bar{C}_k^t + \eta(\bar{X}^{(t)} - \bar{C}_k^t)$$

kde  $\eta$  je rychlost adaptace, která se postupně snižuje s počtem iterací.

- . Ukonči, pokud  $\eta = 0$  nebo po určitém počtu kroků. Jinak pokračuj bodem 2

# Učení prototypů III

- Pokud neumíme odhadnout počet shluků v datech, vycházíme z jejich nulového počtu. Postup v tomto případě:

- Přečti vzor

- Vyhledej nejbližší shluk  $k$ . Pokud je vzdálenost menší než  $r$ , modifikuj střed shluku podle

$$\bar{C}_k^{(t+1)} = \bar{C}_k^t + \eta(\bar{X}^{(t)} - \bar{C}_k^t)$$

- Pokud je vzdálenost větší než  $r$ , založ nový střed na pozici vzoru  $X$ , tj. .

$$\bar{C}_k^{t+1} = \bar{X}^{(t)}$$

- Ukonči, pokud  $\eta = 0$ , nebo po určitém počtu kroků. Jinak pokračuj bodem 2.

# Určení parametru $\sigma$

- Parametr  $\sigma$  je možno určit jako střední kvadratickou vzdálenost vzorů od středu shluku.

$$\sigma_k = \sqrt{\frac{1}{Q} \sum_{i=1}^Q \|\bar{C}_k - \bar{X}_q\|^2}$$

- kde  $X_q$  je  $q$ -tý vzor náležející ke shluku se středem  $C_k$ .

# Učení vah výstupních neuronů

- Váhy ve výstupní vrstvě budeme opakovaně upravovat tak, abychom minimalizovali energetickou funkci:

$$\Delta \bar{w}^{(t)} = -\eta \nabla E^{(t)} = \eta (D^{(t)} - Y^{(t)}) Y^{*(t)}$$

- Vzpomínáte si? Co vám to připomíná?

Energetickou funkcí  
je v tomto případě

$$E = \frac{1}{2} \sum_{t=1}^m \sum_{i=1}^n \left( d_i^{(t)} - y_i^{(t)} \right)^2$$

Pro odvození vztahu pro úpravu vah jsme použili gradientní algoritmus.

# Gradientní učení RBFN

- V magisterském studiu ...

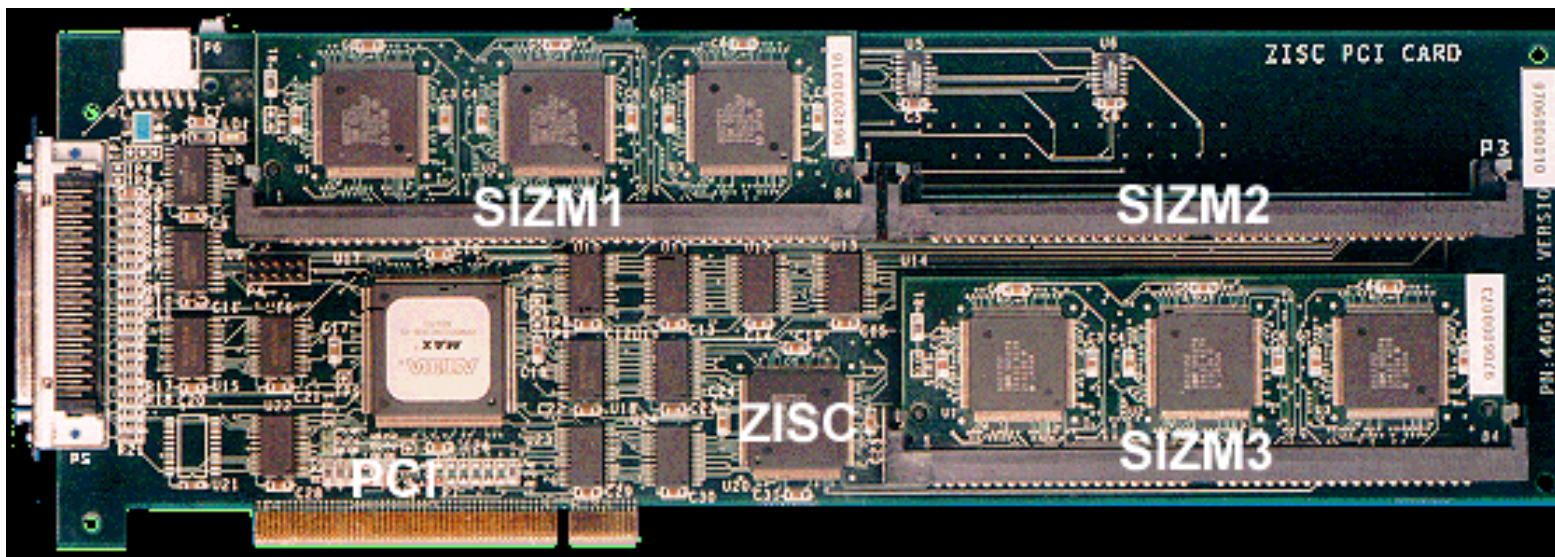
# Implementace sítě RBF – neuročip ZISC 36

- Neuročip ZISC (Zero Instruction Set Computer) vyrábí firma IBM.
- Jedná se o jednoúčelový procesor s pevně danou funkcí, který lze omezeně konfigurovat, ale nikoliv programovat.
- Číslo 36 v názvu udává počet neuronů implementovaných v jednom pouzdře.

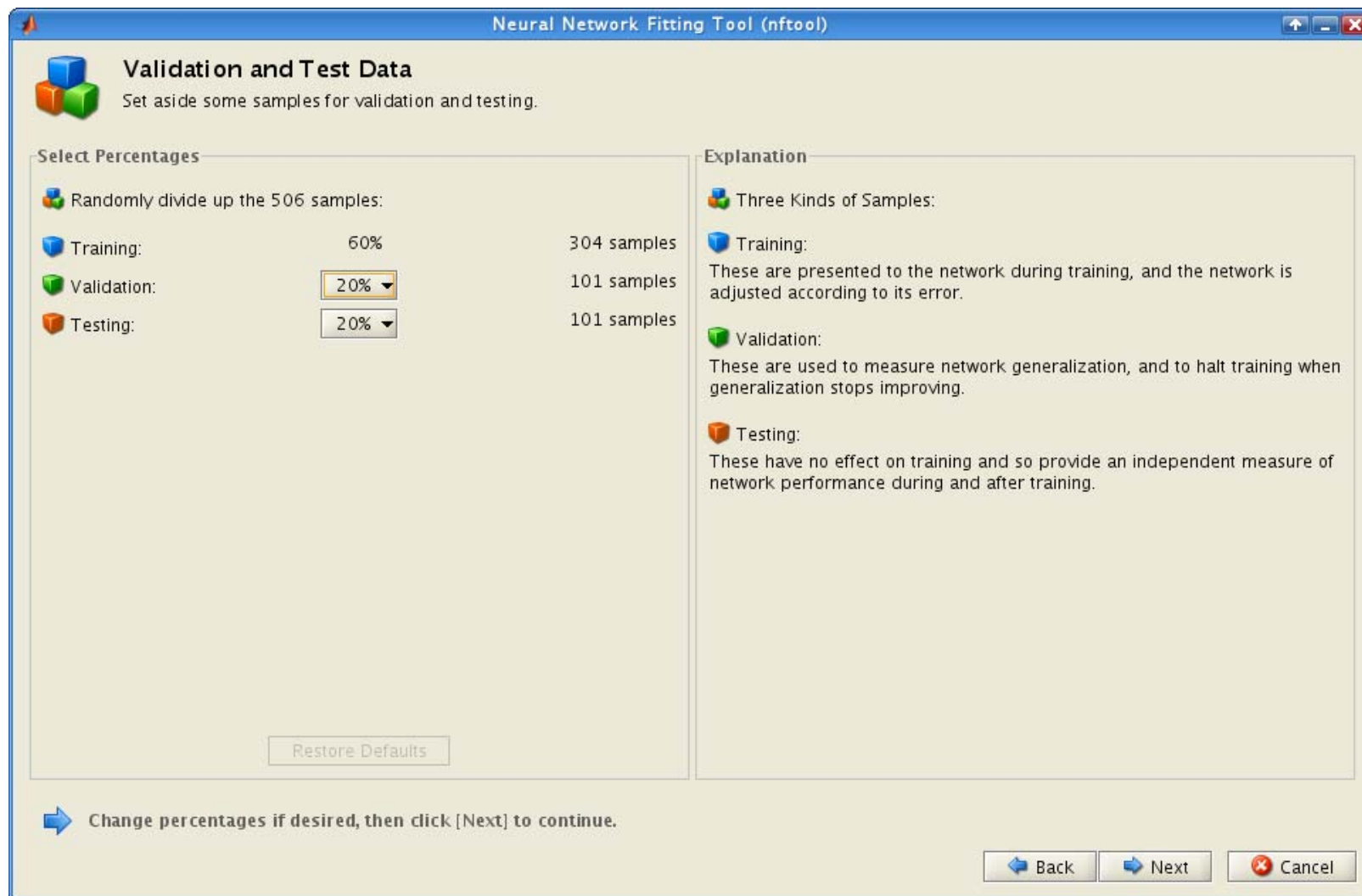


# Neuročip přes WEB

- <http://axon.felk.cvut.cz/zisc/zisc.php>



# Možné použití v semestrálce – Matlab neural network toolbox



# Jak určit gradient energetické funkce?

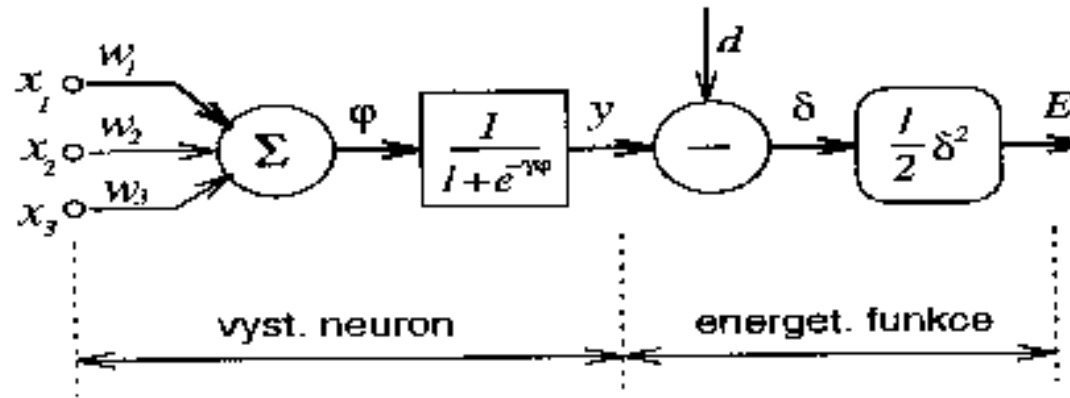
Pro zájemce pokračování algoritmu učení BP

Vypočítáme-li parciální derivace

$$\partial E / \partial w_{ij}^o$$

pro všechny váhy, dostaneme vektor, který určuje **lokální gradient** energetického prostoru.

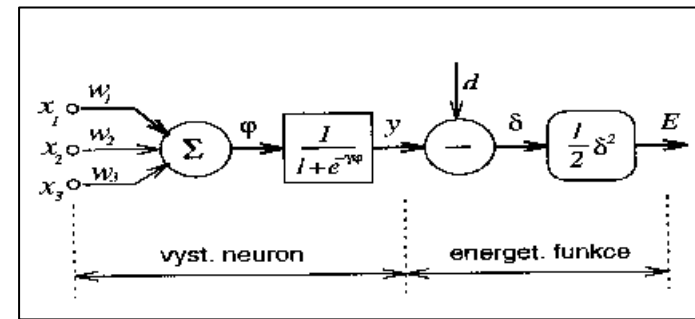
# Studie: neuron ve výstupní vrstvě



Energetická funkce  $g$  je funkcí složenou. Proto můžeme partiální derivaci rozepsat

$$\frac{\partial E}{\partial w_{ij}^0} = \frac{\partial E}{\partial y_i^0} \frac{\partial y_i^0}{\partial \varphi_i} \frac{\partial \varphi_i}{\partial w_{ij}^0}$$

# Jednotlivé položky v předchozím vztahu přitom mají fyzikální význam:



$$\frac{\partial E}{\partial y_i^o}$$

ukazuje závislost energie na výstupu sítě,

$$\frac{\partial y_i^o}{\partial \varphi_i^o}$$

je derivací sigmoidy a

$$\frac{\partial \varphi_i^o}{\partial w_{ij}^o}$$

odráží závislost vnitřního potenciálu na vahách.

# Už víte, proč má BP neuron sigmoidu?

- Aby tato derivace existovala, musí být tato nelineární funkce spojitá a diferencovatelná: tedy nesmí to být skoková nelinearita (ale může se k ní libovolně blížit).
- Tuto vlastnost má právě sigmoida.

# Takže konkrétně:

1. Derivace energie:

$$E = \frac{1}{2} \sum_{i=1}^n (y_i - d_i)^2 \quad \frac{\partial E}{\partial y_i^o} = (y_i^o - d_i)$$

2. Derivace sigmoidy

$$S(\varphi) = \frac{1}{1 + e^{-\varphi}} \quad \frac{\partial y_i^o}{\partial \varphi_i^o} = y_i^o (1 - y_i^o) .$$

3. Derivace  
vnitř. potenciálu

$$\varphi_i^o = \sum_{j=1}^n w_{ij}^o y_j^h + \Theta_i^o \quad \frac{\partial \varphi_i^o}{\partial w_{ij}^o} = y_j^h$$



Shrneme-li tyto dílčí výsledky,  
dostaneme

$$\frac{\partial E}{\partial w_{ij}^o} = (y_i^o - d_i) w_{ij}^o (1 - y_i^o) y_j^h = \underbrace{\delta_i^o}_{S'(\phi)} y_j^h \cdot$$

„odchylka“

x vstup

Symbolem  $\delta$  označujeme lokální gradient.



# Co tento vztah znamená?

- Určuje gradient energetického prostoru, podle kterého budeme upravovat váhy.

Váhy proto můžeme upravovat podle vztahu ( $\eta$  určuje koeficient učení, velikost kroku. Postupujeme proti gradientu):

$$\Delta \vec{W} = -\eta \nabla E$$

# Změny ve výstupní vrstvě:

$$\Delta w_{ij}^o(t) = \eta \delta_i^o(t) y_j^h(t)$$

$$\Delta \Theta_i^o(t) = \eta \delta_i^o(t)$$

Teď už je to jasné

kde

$$\delta_i^o(t) = (y_i^o - d_i) \cdot S'(\varphi)$$

Váhy a prahy výstupní vrstvy  
tedy při učení upravujeme podle  
vztahů

$$w_{ij}^o(t+1) = w_{ij}^o(t) + \Delta w_{ij}^o(t)$$

$$\Theta_i^o(t+1) = \Theta_i^o(t) + \Delta \Theta_i^o(t)$$

a máme zaručeno, že směřujeme do minima energetické funkce.

Pro skrytou vrstvu tedy dostáváme

$$\Delta w_{ij}^h(t) = \eta \delta_i^h(t) x_j(t)$$

$$\Delta \Theta_i^h(t) = \eta \delta_i^h(t)$$

kde

$$\delta_i^h = y_i^h (1 - y_i^h) \sum_{k=1}^n w_{ki}^o \delta_k^o$$

a změny vah a prahů provedeme takto:

$$w_{ij}^h(t+1) = w_{ij}^h(t) + \Delta w_{ij}^h(t)$$

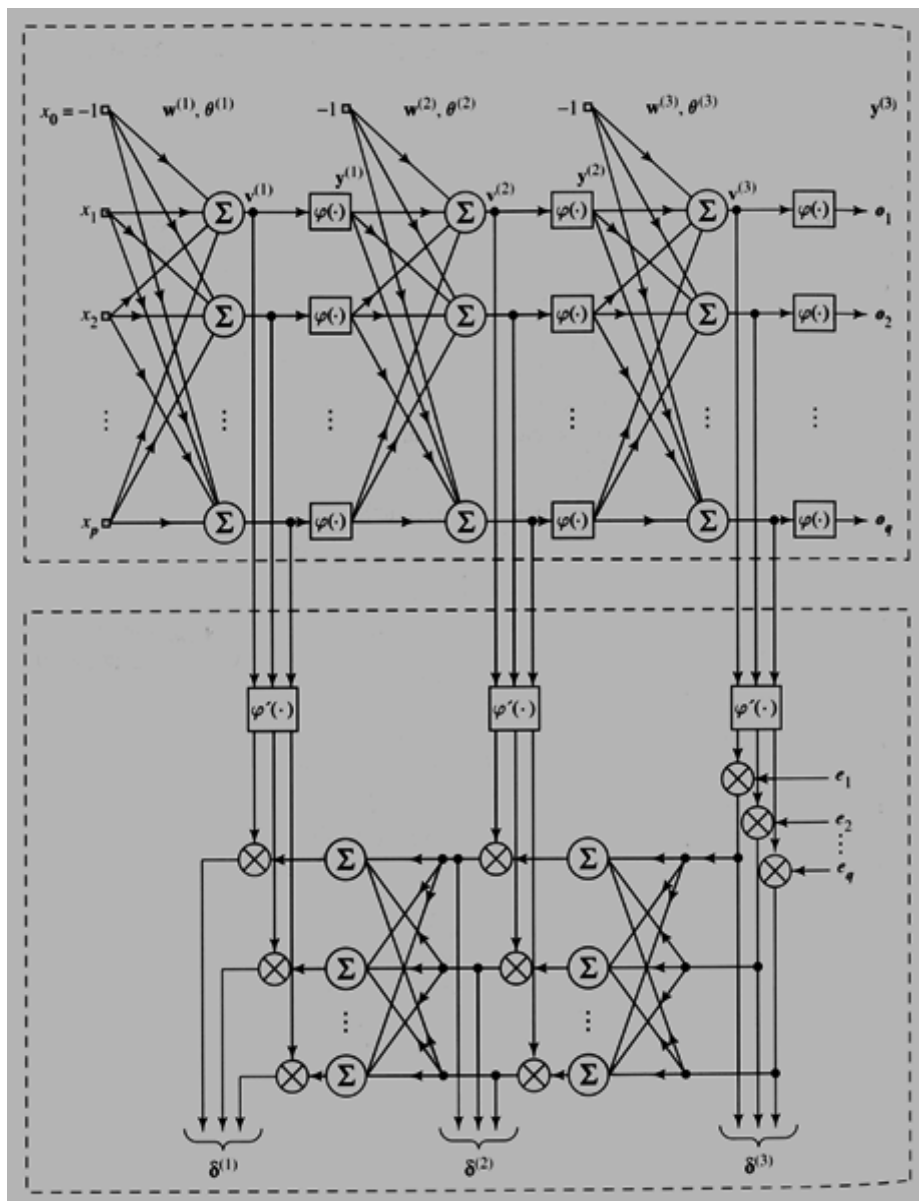
$$\Theta_i^h(t+1) = \Theta_i^h(t) + \Delta \Theta_i^h(t)$$

Výpočet  $\delta$  lze pro celou síť zobecnit následovně:

$$\delta_i^{h-1} = y_i^{h-1} (1 - y_i^{h-1}) \sum_{k=1}^n w_{ki}^h \delta_k^h$$

Postup výpočtu nových vah a prahů shrnuje následující obrázek

Vlastní  
neuronová  
sít'



Zpětné  
šíření  
chyby

Připomenutí:

$$\delta_i^o(t) = (y_i^o - d_i) \cdot S'(\varphi)$$

$$\delta_i^h = y_i^h (1 - y_i^h) \sum_{k=1}^n w_{ki}^o \delta_k^o$$

# Modifikace standardního algoritmu

## ■ Setrvačnost

$$\Delta w_{ij}^o(t) = \eta \delta_i^o(t) y_j^h(t) + \alpha \Delta w_{ij}^o(t-1)$$

$$\Delta \Theta_i^o(t) = \eta \delta_i^o(t) + \alpha \Delta \Theta_i^o(t-1)$$

Co se v těchto vztazích změnilo?



# Modifikace standard. algoritmu II

- Normalizované kumulované delta pravidlo,
  - změny až najednou,
- Delta-bar-delta pravidlo,
  - nejen současný, ale i minulý gradient,
- rozšířené,
  - obojí plus gradient,
- Quickprop, Maxprop,
- rychlé učení.

# Modifikace standard. algoritmu II

- Resilient backpropagation
  - gradient určuje znaménko změny vah
  - pro stabilní gradient zvýš rychlost učení
  - pro proměnlivý gradient sniž rychlost učení

```
public void modifyWeightsAndBias() {
    for( int i = 0; i < inputs; i++) {
        gradient[i] = (squareError * sign[i].outValue);
        if(Math.abs(gradient[i]) > 0.0001) {
            delta[i] = Math.min(nPlus*delta[i],maxDelta);
            dlt[i] = -(this.sign(gradient[i]))*delta[i];
            previousGradient[i] = gradient[i];
            a[i] += dlt[i];
        } else
        if(previousGradient[i]*gradient[i]<0){
            delta[i] = Math.max(nMinus*delta[i],minDelta);
            a[i] -= dlt[i];
            previousGradient[i] = 0;
        } else if(previousGradient[i]*gradient[i]==0){
            dlt[i] = -(this.sign(gradient[i])) * delta[i];
            previousGradient[i] = gradient[i];
            a[i] += dlt[i];
        }
        previousWeigthChange[i] = dlt;
        dlt = -learningSpeed * squareError * f[i].outValue;
        ba[i] += (dlt + bdelta[i] * inertiaCoefficient);
        bdelta[i] = dlt + bdelta[i] * inertiaCoefficient;
        dlt = learningSpeed * squareError;
        bias = dlt + lastBiasDelta * inertiaCoefficient;
        lastBiasDelta = dlt;
    }
}
```