Data mining

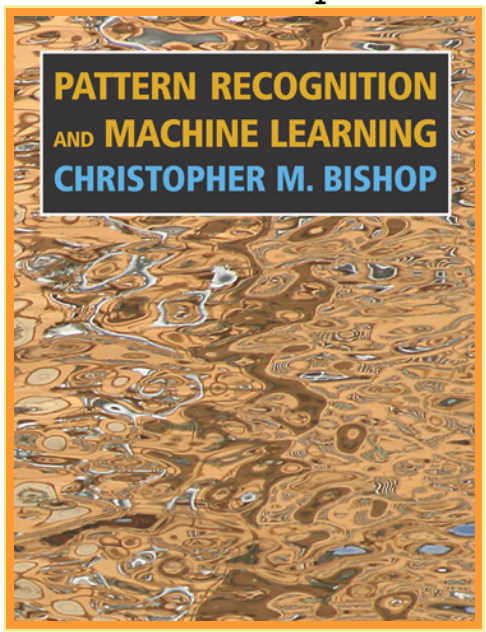
Neuronové sítě

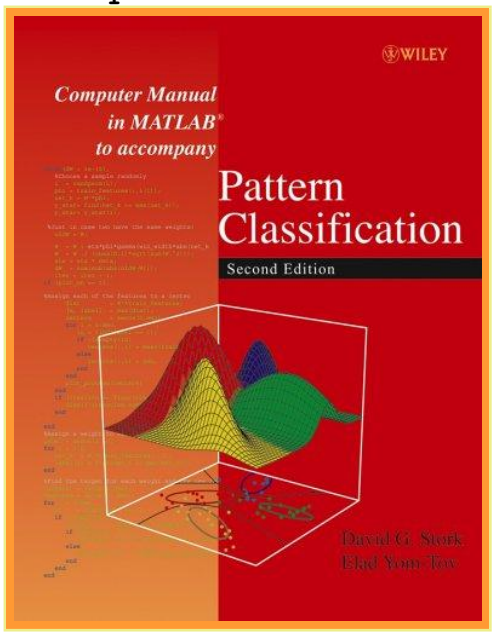
Gerstnerova laboratoř
katedra kybernetiky
České vysoké učení technické v Praze

Daniel Novák

Literatura, dema

- Duda, Hart, Stork: Pattern Classification `http://www.crc.ricoh.com/~stork/DHS.html`
- Ch. Bishop, Pattern Recognition and Machine Learning `http://research.microsoft.com/en-us/um/people/cmbishop/prml/`
- Kotek, Vysoký, Zdráhal: Kybernetika 1990
- Classification toolbox
  `http://stuff.mit.edu/afs/sipb.mit.edu/user/arolfe/matlab/`
- Statistical Pattern Recognition Toolbox
  `http://cmp.felk.cvut.cz/cmp/software/stprtool/`





percepton - Lineární forma klasifikátoru

- Předpokládejme binární klasifikační problém $S = \{s_1, s_2\}$,$\vec{x}$ je n-dimenzionální příznak
- Odhadujeme $\vec{b}, c$ přímo ze zadaného vzorku $D = \{(\vec{x}_1, y_1), (\vec{x}_2, y_2) \dots (\vec{x}_m, y_m)\}$.
- Požadujeme $\left(\vec{b}^t \vec{x}_i + c\right) > 0$ pokud $y_i = s_1$ a $\left(\vec{b}^t \vec{x}_i + c\right) < 0$ jinak.
- Totéž, jako požadovat $\left(\vec{b}^t \vec{z}_i + c\right) > 0$ pro všechna z_i, kde $z_i = x_i$ pokud $y_i = s_1$ a $z_i = -x_i$ jinak.
- Formálně, nechť $z_i^0 = 1 \; \forall i$ a $\vec{w} = [\vec{b}, c]$ (přidejme c jako poslední složku $\vec{w}$).
- Nyní můžeme jednoduše psát $g(\vec{z}) = \vec{w}^t \vec{z}$ a požadovat $\vec{w}^t \vec{z}_i > 0$ pro všechna z_i .
- Nechť

$$J(\vec{w}) = \sum_{\vec{z}_i \in M} -\vec{w}^t \vec{z}_i$$

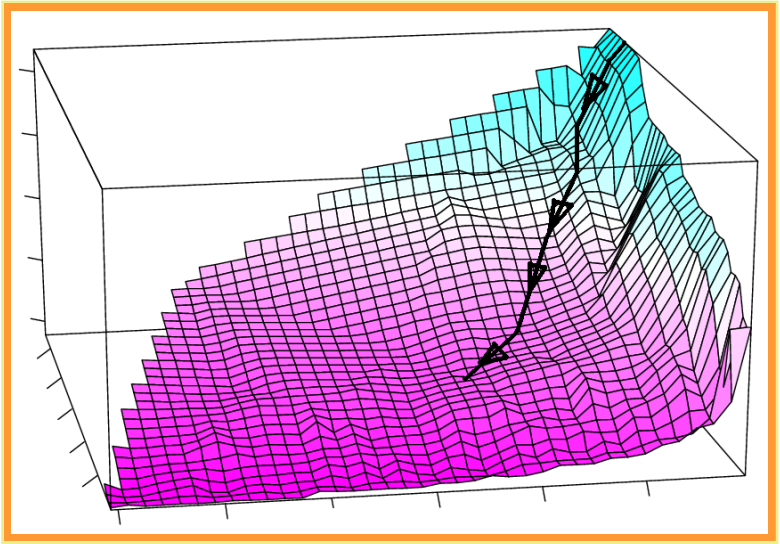
kde M je množina všech **nesprávně klasifikovaných** $\vec{z}_i$.

Perceptron

- $J(\vec{b}, c)$ je vždy nezáporná.
- Pokud $J(\vec{w}) = 0$, tak všechny příklady v D jsou správně klasifikovány a D je **lineárně separabilní**. Chceme najít minimum $J(\vec{w})$.
- $J(\vec{w})$ je po částech lineární. Pro hledání minima lze použít **gradientního algoritmu**.
- Gradientní algoritmus: přibližuj se k minimu pomocí malých diskrétních krorků v $\Re^{n+1}$ ve směru proti gradientu $J(\vec{w})$.

$$\nabla(J(\vec{w})) = \left(\frac{\partial J(\vec{w})}{\partial w_1}, \frac{\partial J(\vec{w})}{\partial w_2}, \dots \frac{\partial J(\vec{w})}{\partial w_{n+1}}\right) = \sum_{z_i \in M} -\vec{z}$$

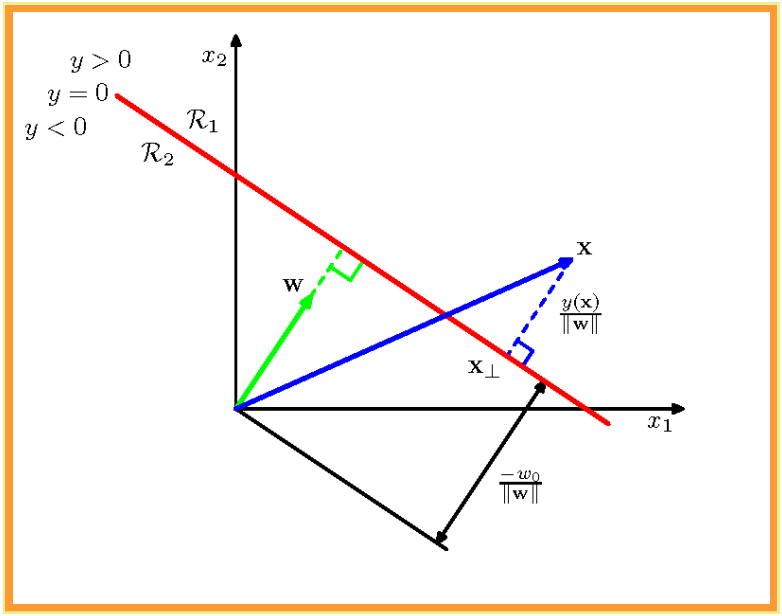
- **Perceptronový gradientní algoritmus**:
  1. $k = 0$. Zvol náhodně $\vec{w}$.
  2. $k \leftarrow k + 1$
  3. $\vec{w} \leftarrow \vec{w} + \eta(k) \sum_{z_i \in M_k} \vec{z}$
  4. pokud $|\sum_{z_i \in M_k} \vec{z}| > \theta$ jdi na 2
  5. vrať $\vec{w}$
- η - **rychlost učení**, θ - mez přijatelné chyby.



Příklad pro $\vec{w} \in \Re^2$, $J(\vec{w})$ na svislé ose.

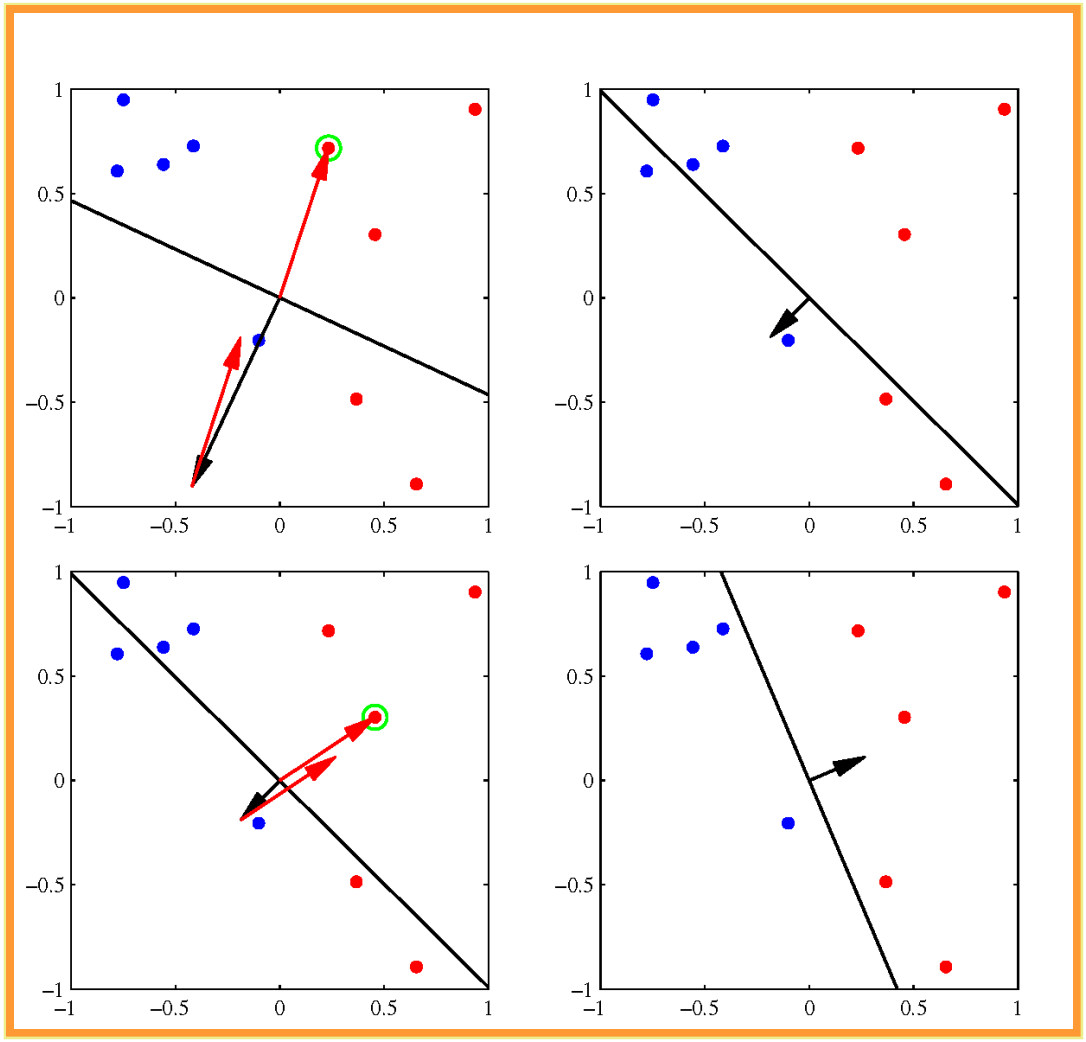
Percepton II

- Sjednocení terminologie se cvičením - rozšíření na K tříd
- grafické znázornění rozhodovací hranice (změna symbolů)
  - $y(\vec{x}) = \vec{w}^t\vec{x} + w_0$, $y(\vec{x_a}) = y(\vec{x_b})$
  - $\vec{x_a}$ a $\vec{x_b}$ leží na rozhodovací hranici, tedy $\vec{w}^t(\vec{x_a} - \vec{x_b}) = 0)$
  - w je ortonormální na rozhodovací hranici, $\omega_0(b)$ je posunutí [Bishop]



Percepton - grafické znázornění učení

- grafické znázornění [Bishop]

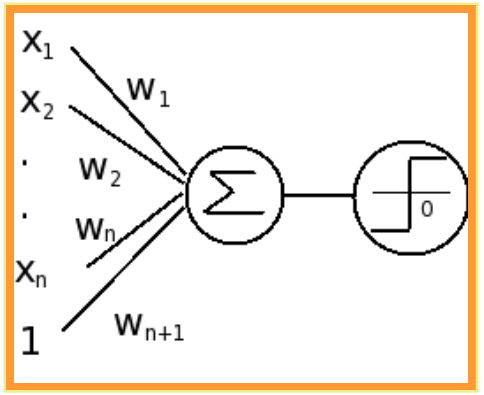


Perceptron: lineární diskriminace

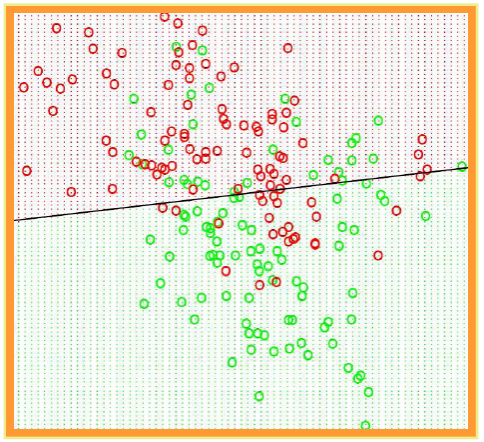
- Perceptron je využíván obecně pro úlohy lineární klasifikace, nejen za předpokladu normálně rozdělených příznaků.
- Je-li zadaný vzorek dvou tříd lineárně separabilní, perceptronový algoritmus skončí v konečném čase s nulovou chybou (pro $\theta = 0$ a dostatečně malé $\eta(k)$).
- Vzorek lineárně neseparabilní v $\Re^n$ může být separabilní po *transformaci* do $\Re^{n'}$ $n' > n$. Např. pro $\vec{x} \in \Re^n$, $T(\vec{x}) =$

$$[\underbrace{x(1), \ldots, x(n)}_{=\vec{x}}, x^2(1), x(1)x(2), x(1)x(3), \ldots x^2(n)]$$

- Tj. přidány kvadratické členy. Lineární diskriminace v transformovaném prostoru odpovídá nelineární (zde kvadratické) diskriminaci v původním prostoru $\Re^n$.
- Lineární diskriminační metody, jako je perceptron, mohou být tedy pomocí tranformace využity i pro hledání nelineární diskriminace.
- DEMO - iterace rozhodovací hranice ve 2D případu



Schema perceptronu


Lineárně neseparabilní problém

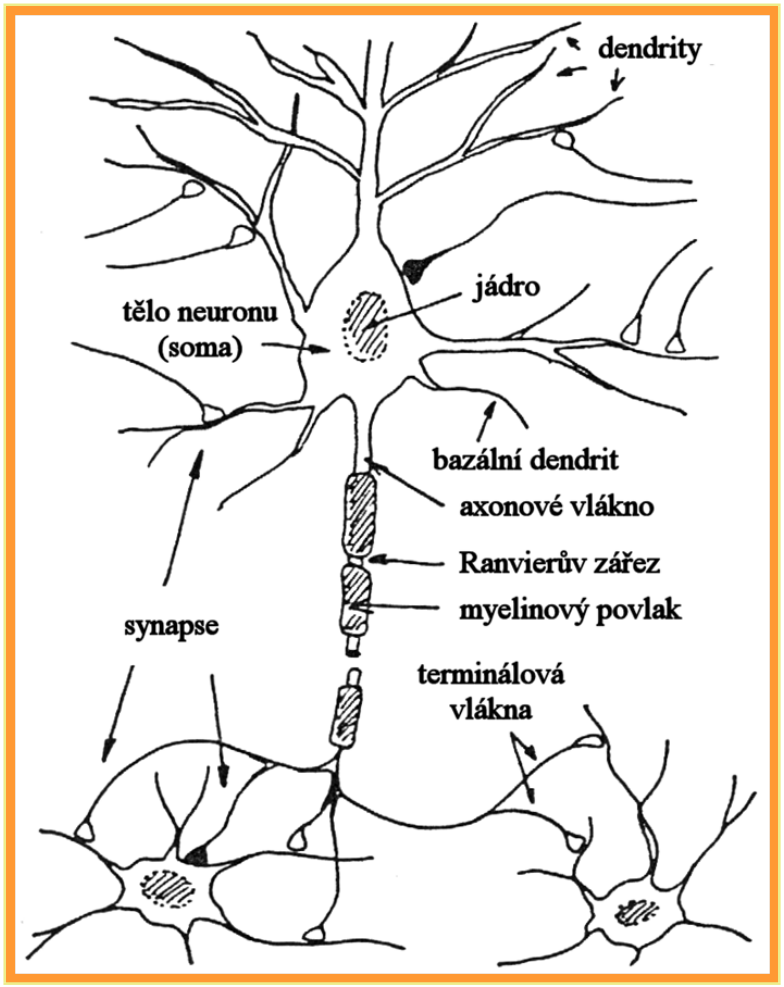
Percepton - historie

- Frank Rosenblatt - HW realizace perceptonu v roce 1958

- Učení jednoduchých písmen a tvarů - inspirace neuronovými sítěmi v mozku
- Světla osvětlují objekt, foceno na pole 20 x 20 cadmium-sulfádových buněk, dostáváme tedy obrázek o 400 pixelech
- Přepojovací deska - možnost různé konfigurace vstupů
- Adaptivní váhy - potenciometr ovládány elektrickým motorem - automatické nastavování vah učícím algoritmem
- Implementace na počítači Mark 1 (Hardvard-IBM spolupráce) - 765000 součastek, 16 m dlouhý 2.4 m vysoký, 2 m široký, 3 operace za sekundu, násobení trvalo 6 vteřin

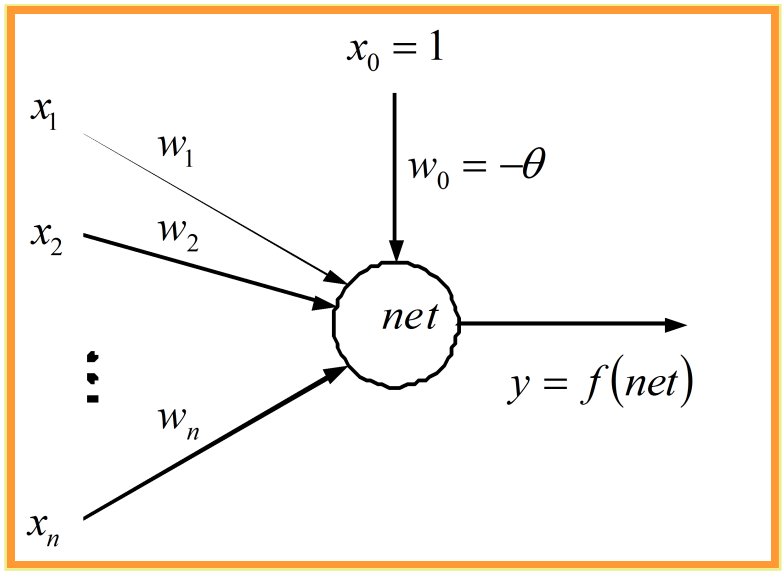
Neuronové sítě

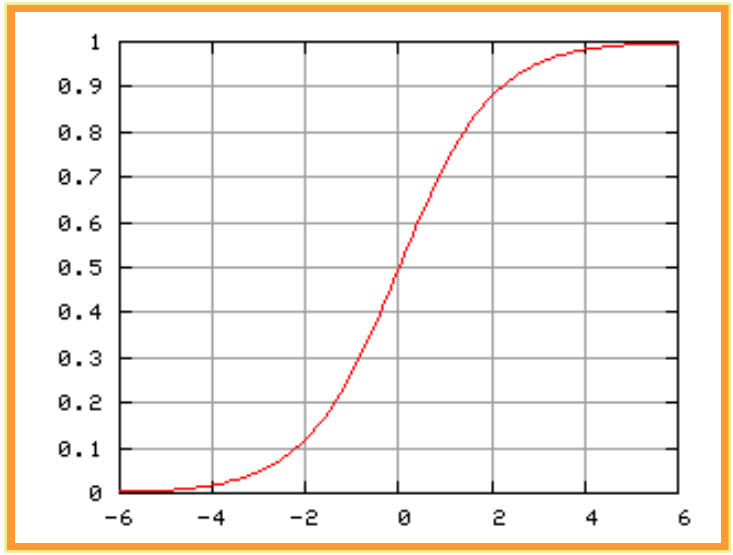
- Umělá neuronová sít je distribuovaný výpočetní systém sestávající z dílčích podsystémů (neuronů), který je inspirován neurofyziologickými poznatky o struktuře a činnosti neuronu a nervových systému živých organizmů, a který je ve vetší či menší míře realizuje.
- Neurony mezi sebou komunikují na základě přenášení elektrického potenciálu pomocí synapsí
- Schopnost extrahovat a reprezentovat závislosti v datech, které nejsou zřejmé
- Schopnost učit se, schopnost zevšeobecňovat
- Schopnost rešit silně nelineární úlohy



Definice neuronu

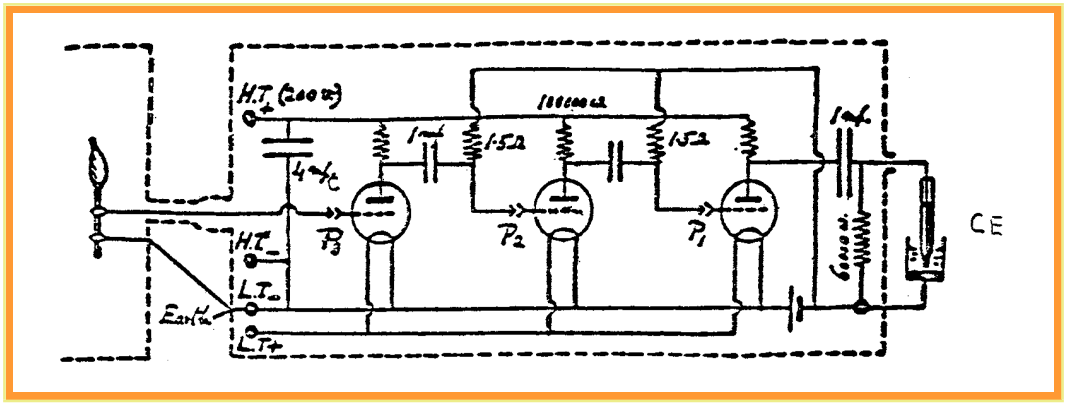
- Neuron je základní výpočetní jednotkou neuronových sítí
- Vstupy x_i jsou váhovány parametry ω_i
- $net = \sum_{i=1}^{n} x_i \omega_i + w_0 = \sum_{i=0}^{n} \vec{w}^t \vec{x}$
- Zavedení nelinearity do neuronové sítě $y = f(net)$, nejčastěji sigmoid třida, např, $\tanh$ či logistická funkce $y = f(net) = \frac{1}{1+\exp^{-\lambda * net}}$

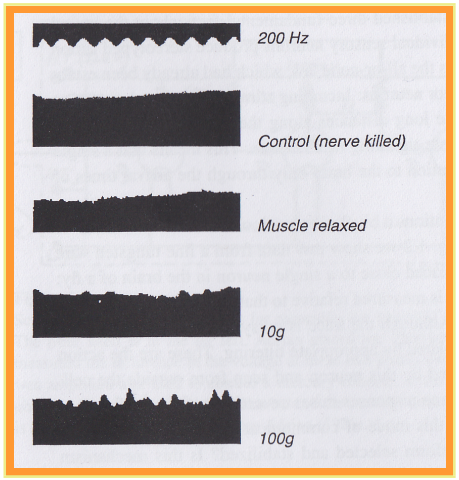


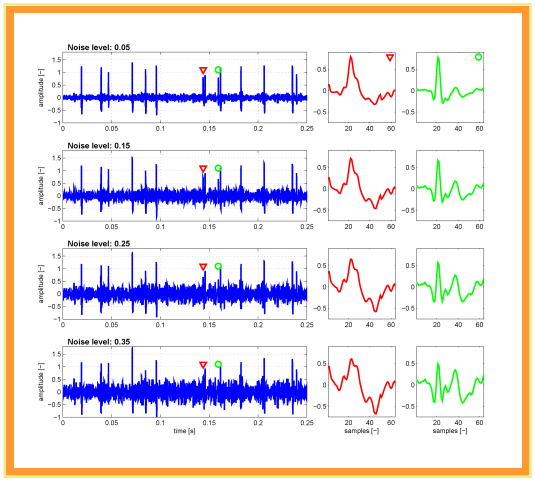


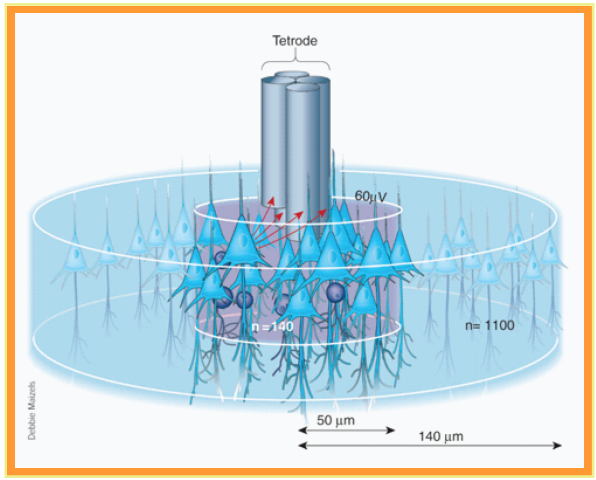
Fyziologie

- Pionýrská práce barona Edgarda Adriana (Nobelova cena za medicínu v roce 1932)
  `http://nobelprize.org/nobel_prizes/medicine/laureates/1932/adrian-bio.html#`



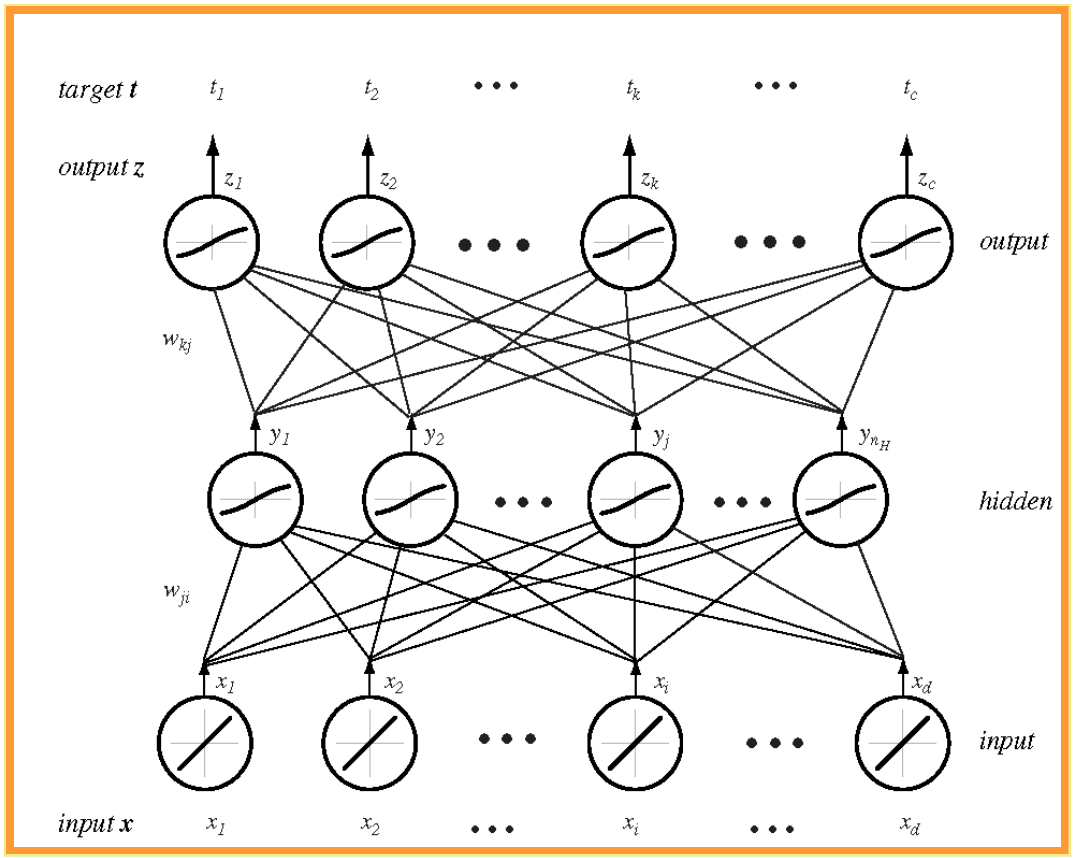






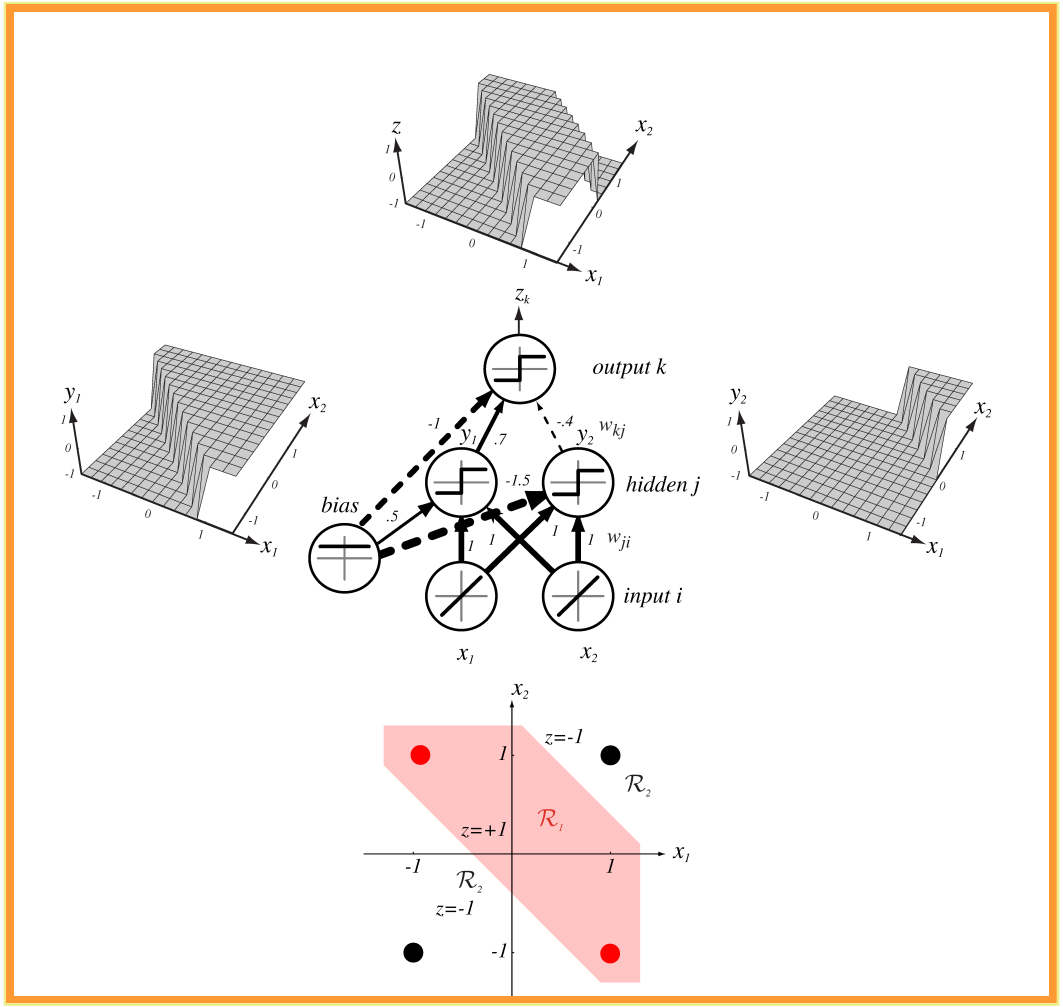
Třívstvá neuronová síť ($d - n_H - c$)

- První vrstva je vstupní vrstva, akční funkce je zde lineární, počet neuronů se rovná dimenzi vstupního vektoru, $1 \ldots d$
- Druhá vrstva je skrytá, libovolný počet neuronů, $1 \ldots n_H$
- Třetí vrstva je výstupní vrstva, počet neuronů je nejčastěji roven počtu tříd, $1 \ldots c$



Příklad - třívstvá neuronová sít´ - řešení XOR problému

- $0 \bigoplus 0 = 0,\ 1 \bigoplus 1 = 0,\ 1 \bigoplus 0 = 1,\ 0 \bigoplus 1 = 1$
- $-1 \bigoplus -1 = -1,\ 1 \bigoplus 1 = -1,\ 1 \bigoplus -1 = 1,\ -1 \bigoplus 1 = 1$



Řešení XOR problému

- rozhodovací hranice skrytého neuronu y_1

$$x_1 + x_2 + 0,5 = 0 \begin{cases} \geq 0 & \text{if } y_1 = +1 \\ < 0 & \text{if } y_1 = -1 \end{cases}$$

- rozhodovací hranice skrytého neuronu y_2

$$x_1 + x_2 - 1.5 = 0 \begin{cases} \geq 0 & \text{if } y_2 = +1 \\ < 0 & \text{if } y_2 = -1 \end{cases}$$

- neuron ve výstupní vrstvě z

$$0.7y_1 - 0.4y_2 - 1 = 0 \begin{cases} \geq 0 & \text{if } z = +1 \\ < 0 & \text{if } z = -1 \end{cases}$$

Aktivace neuronu

- aktivace net_j skryté vrstvy $net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji} = \vec{w_j}^t \vec{x}$
- i indexuje neuron ve vstupní vrstvě, j skrytou vrstvu, w_{ji} je váha neuron j ve skryté vrstvě, který je spojen se vstupním neuronem i (synapse).
- Výstup neuronu ve skryté vrstvě $y_j = f(net_j)$
- XOR problém

$$f(net) = sgn(net) \begin{cases} 1 & \text{net } \geq 0 \\ -1 & \text{net } < 0 \end{cases}$$

- funkce $f(.)$ se nazývá aktivační funkcí.
- Podobně, aktivační funkce net_k neuronu ve výstupní vrstvě je dána jako $net_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj} = \vec{w_k}^t \vec{y}$
- k indexuje neuron ve výstupní vrstvě, n_H je počet skrytých neuronů
- Výstup neuronu ve výstupní vrstvě $z_k = f(net_k)$
- V případě c tříd (výstupů), síť počítá c diskriminačních funkcí $z_k = g_k(\vec{x})$ a klasifikuje vstup $\vec{x}$ podle největší diskriminační funkce $g_k(\vec{x}) \qquad \forall k = 1, \ldots c$

Vybavování sítě - dopředná operace

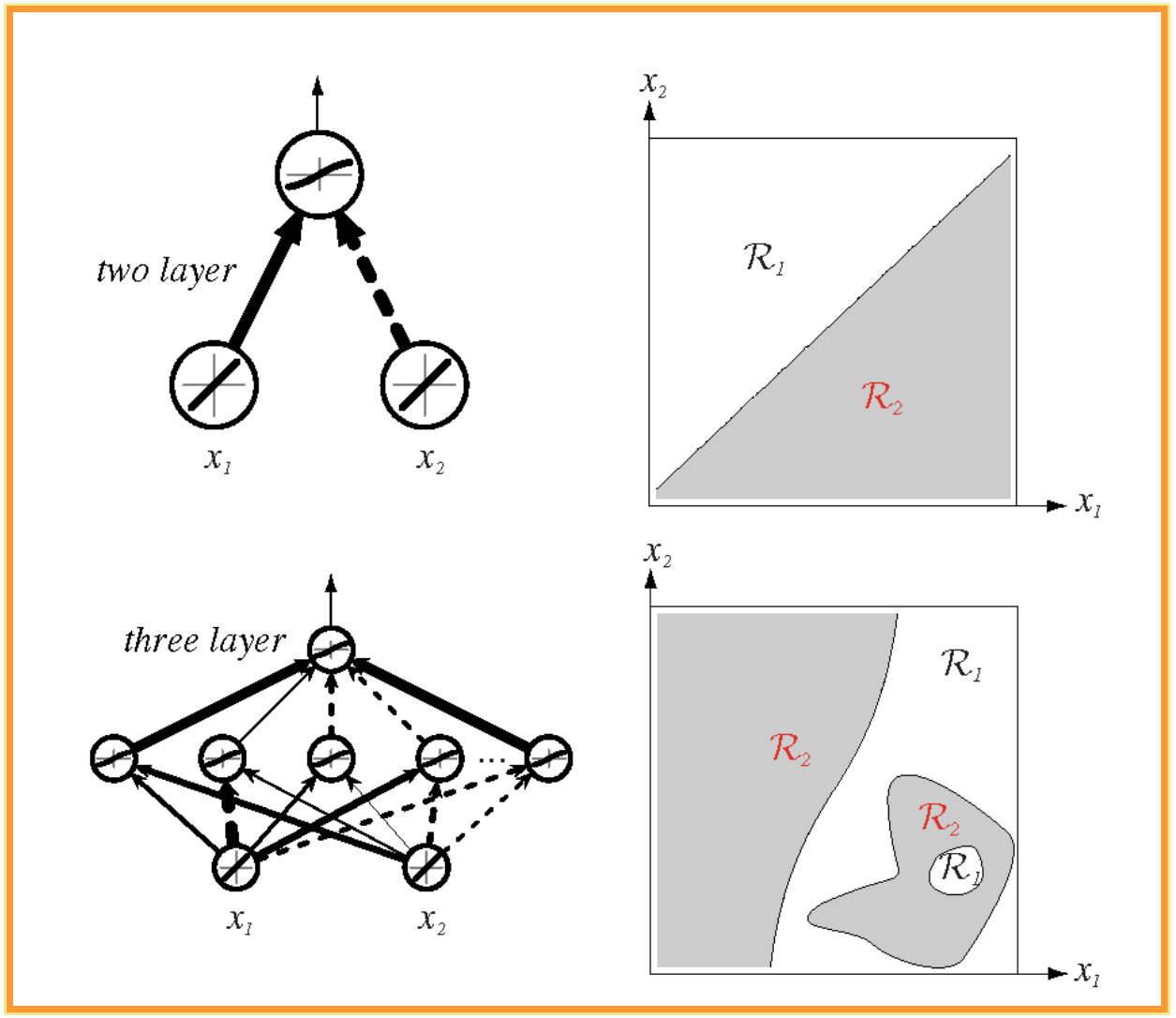
- Výstup sítě

$$g_k(\vec{x}) = z_k = f(\sum_{j=1}^{n_H} w_{kj} f(\sum_{i=1}^{d} w_{ji} x_i + w_{j0}) + w_{k0}) \qquad \forall k = 1 \dots c$$

- Skrytá vrstva umožnuje realizovat komplikované nelineární funkce
- Aktivační funkce - v každé vrstvě můžeme mít rozdílnou aktivační funkci, dokonce každý neuron může mít svoji vlastní aktivační funkci
- Pro zbytek přednášky předpokládáme, že máme jeden typ aktivační funkce
- **OTÁZKA: Může třívrstvá neuronová sít´ aproximovat jakoukoliv nelineární funkci?**
- **ODPOVĚD: ANO - díky A.Kolmogorovi**
  Jákákoliv spojitá funkce může být implementovaná třívrstvou sítí za předpokladu dostatečného počtu n_H skrytých neuronů,vhodných nelinearit a vah w.

Příklad rozhodovací hranice

- Porovnání 2-vrstvé a 3-vrstvé sítě



Andrej Kolmogorov

- Již na střední škole sestrojil "perpetuum mobile", jeho učitel nepřišel na použitý trik
- Studoval nejdříve historii na Moskevské státní univerzitě (nastoupi v 17 letech)
- První vědecký článek publikoval na téma vlastnictví nemovitostí v Novgorodu v období 15. a 16. století
- Největší přínos v teorii pravděpodobnosti a výpočetní složitosti

Jak sít´naučíme ????

- Náš cíl je nastavit váhy na základě trénovacích dat a požadovaného výstupu t_k
- Odvodíme si metodu zpětného šíření chyby
- Necht´ t_k je k skutečný výstup a z_k necht´je vypočtený výstup, kde $k = 1, \ldots, c$. Definujeme chybu jako

$$J(\vec{w}) = \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2 = \frac{1}{2} \parallel \vec{t} - \vec{z} \parallel$$

- Algoritmus zpětného šíření chyby je založen na gradietním algorimu (viz percepton). Váhy jsou náhodně inicializovány a jsou měněny ve směru poklesu chyby

$$\triangle\vec{w} = -\eta\frac{\partial J}{\partial \vec{w}}$$

- η je parametr ovlivnující rychlost učení - určuje relativní změnu vah

$$\vec{w}(m+1) = \vec{w}(m) + \triangle\vec{w}$$

- kde m je m-tý použitý vzor $(\vec{x}_m, t_m)$

Odvození

- Chyba pro váhy (skrytá-výstupní)

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}} = \delta_k \frac{\partial net_k}{\partial w_{kj}}$$

- Platí $net_k = \sum_{j=0}^{n_H} y_j w_{kj} = \vec{w}_k{}^t \vec{y}$, tedy $\frac{\partial net_k}{\partial w_{kj}} = y_j$
- kde sensitivita k-tého neuronu je definována jako $\delta_k = -\frac{\partial J}{\partial net_k}$ a popisuje, jak se celková chyba mění s aktivací parametru net_k neuronu, $\frac{\partial z_k}{\partial net_k} = f'(net_k)$

$$\delta_k = -\frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial net_k} = (t_k - z_k) f'(net_k)$$

- Váhy (skrytá-výstupní) se aktualizují jako

$$\boxed{\triangle w_{kj} = \eta \delta_k y_j = \eta (t_k - z_k) f'(net_k) y_j}$$

- Chyba pro váhy (vstupní-skrytá)

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}}$$

- Platí

$$\frac{\partial J}{\partial y_j} = \frac{\partial}{\partial y_j}\left[\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2\right] = -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial y_j} =$$
$$-\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial net_k}\frac{\partial net_k}{\partial y_j} = -\sum_{k=1}^{c}(t_k - z_k)f^{'}(net_k)w_{kj}$$

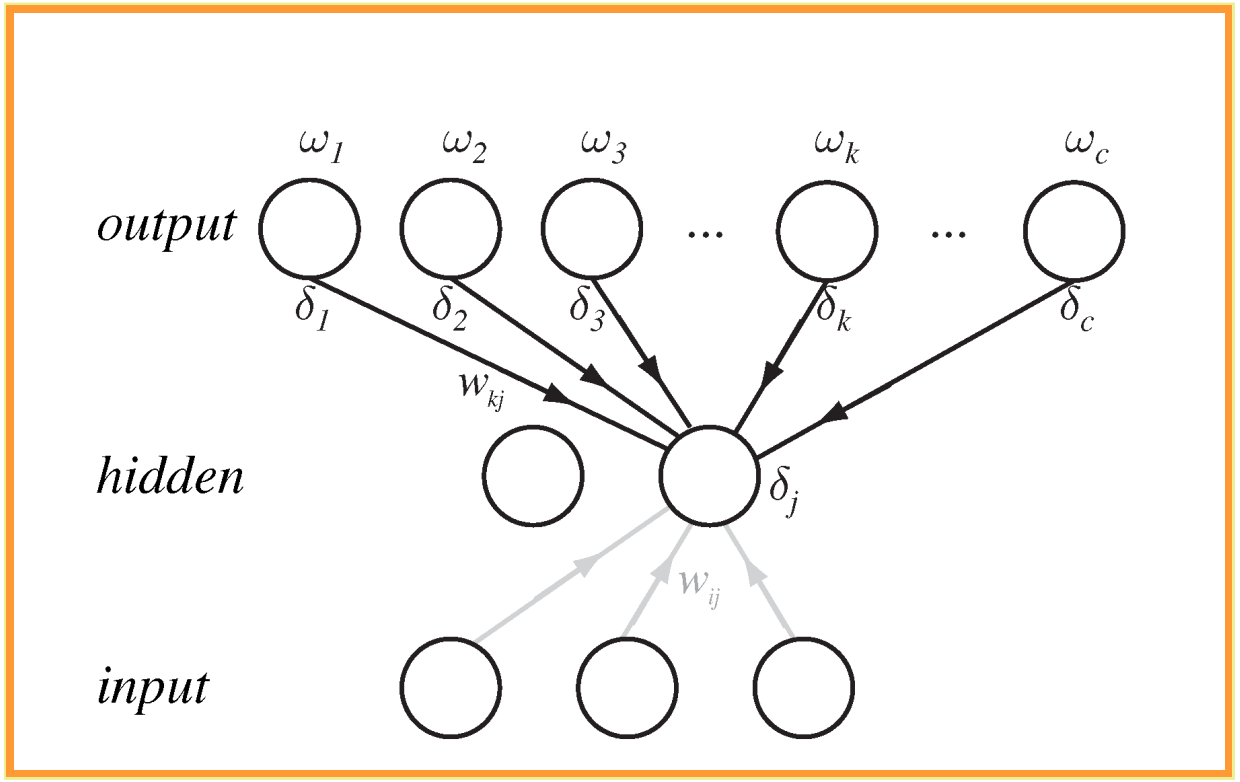
- Platí $\frac{\partial net_k}{\partial y_j} = w_{kj}$, protože $net_k = \sum_{j=0}^{n_H} y_j w_{kj} = \vec{w}_k{}^t \vec{y}$
- Takže $\frac{\partial J}{\partial y_j} = -\sum_{k=1}^{c}\delta_k w_{kj}$, protože $\delta_k = (t_k - z_k)f^{'}(net_k)$
- Zavedeme $\frac{\partial y_j}{\partial net_j} = f^{'}(net_j)$
- Zbývá vyčíslit $\frac{\partial net_j}{\partial w_{ji}} = x_i$, protože platí $net_j = \sum_{i=0}^{d} x_i w_{ji} = \vec{w}_j{}^t \vec{x}$
- Dosadíme a definujeme sensitivitu pro skrytou jednotku. Sensitivita je váhovaný součet výstupních sensitivit, znásoben aktivační funkcí skrytého neuronu

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} = \underbrace{-\sum_{k=1}^{c}\delta_k w_{kj} f^{'}(net_j)}_{\delta_j} x_i$$

Proč zpětné šíření chyby (back-propagation)?

- Pravidlo aktualizace vah (vstupní-skrytá) je

$$\triangle w_{ji} = \eta x_i \delta_j = \eta \sum (w_{kj}\delta_k) f'(net_j) x_i$$



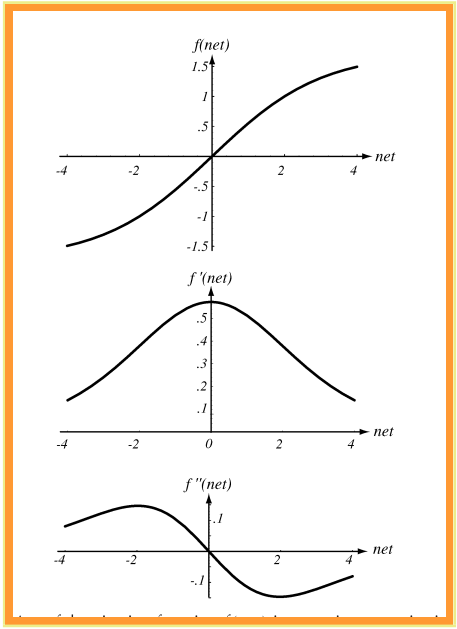
Pseudo-kód

- pseudokód (inkrementální učení, stochastické)

```
1 begin initialize  network topology (# hidden units), w, criterion θ, η, m ← 0
2   do m ← m + 1
3       x^m ← randomly chosen pattern
4       w_ij ← w_ij + η δ_j x_i;   w_jk ← w_jk + η δ_k y_j
5   until ∇J(w) < θ
6 return w
7 end
```

– Implementace v Matlabu: *Backpropagation_Stochastic.m*. Použita funkce tanh, $a = 1.716$, $b = \frac{2}{3}$, aby $f'(0) \simeq 1$.

$$f(net) = a \tanh(b \star net) = \frac{2a}{1 + \exp^{b \star net}} - a$$



Batch učení

- Mám n vstupů $\vec{x_i}$, vyjádřím celkovou chybu přes všechny vzorky jako

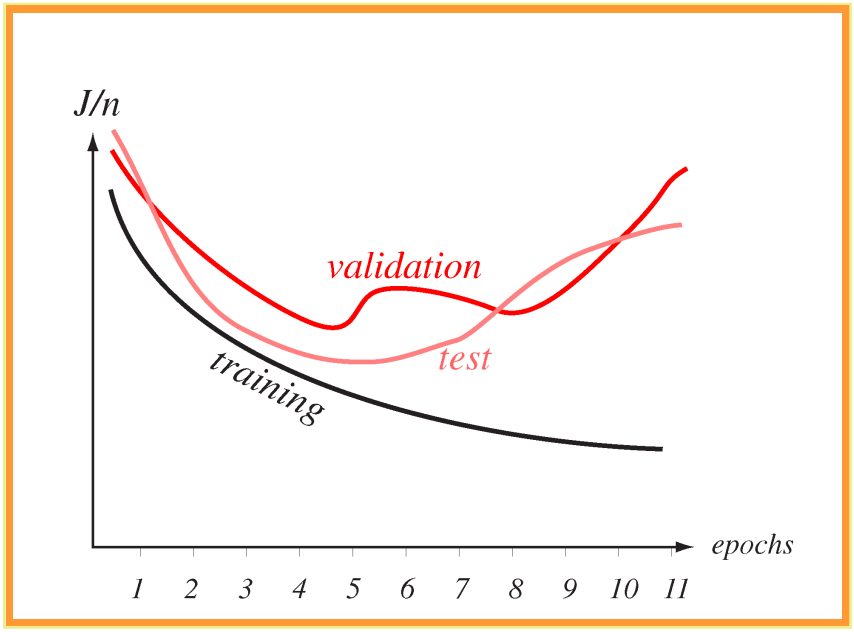
$$J = \sum_{p=1}^{n} J_p$$

- Není nutné vybírat vstupy postupně
- Epocha je jedna prezentace všech vstupů, krok 2: $r = r + 1$

```
1  begin initialize  network topology (# hidden units), w, criterion θ, η, r ← 0
2    do r ← r + 1 (increment epoch)
3       m ← 0; Δw_ij ← 0; Δw_jk ← 0
4       do m ← m + 1
5          x^m ← select pattern
6          Δw_ij ← Δw_ij + ηδ_j x_i;   Δw_jk ← Δw_jk + ηδ_k y_j
7       until m = n
8       w_ij ← w_ij + Δw_ij;   w_jk ← w_jk + Δw_jk
9    until ∇J(w) < θ
10 return w
11 end
```

Validace

- Chyba trénovací množiny je monotonní klesající funkce díky gradietnímu algoritmu (klesám ve směru poklesu chyby)
- Rozdělení dat na trénovací,testovací a validační množinu. Validaci používám jako zastavovací kritérium (první minimum nebo celkové minimum pro konstantní počet epoch)



- DEMO - Neural Network Toolbox v Matlabu
  `http://www.mathworks.com/products/neuralnet/`
- Data pocházejí z UCI Machine Learning Repository
  `http://mlearn.ics.uci.edu/MLRepository.html`