


# Hodnocení úspěšnosti učícího algoritmu

jak můžeme zjistit, zda  $h \approx f$ ?

# Hodnocení úspěšnosti učícího algoritmu

jak můžeme zjistit, zda  $h \approx f$ ? 

- dopředu – použít věty Teorie počítačného učení
- po naučení – kontrolou na jiné trénovací sadě

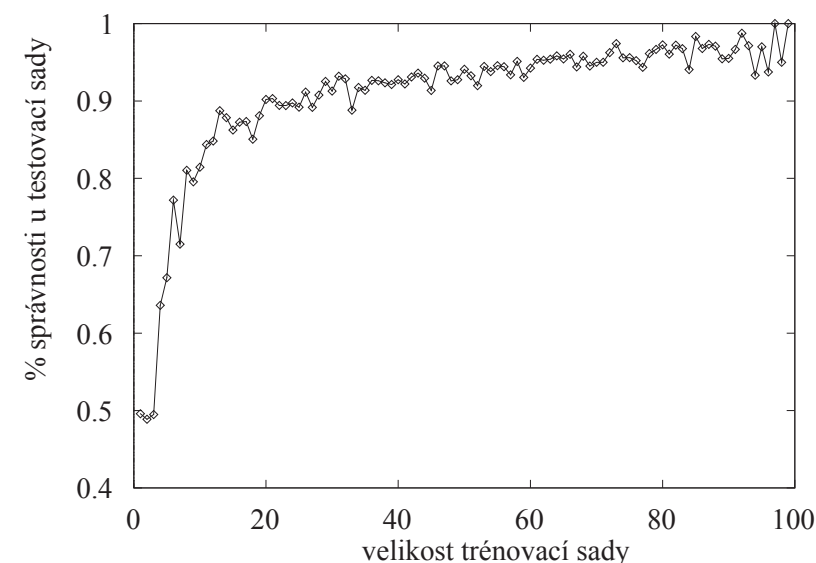
# Hodnocení úspěšnosti učícího algoritmu

jak můžeme zjistit, zda  $h \approx f$ ?  $\left\{ \begin{array}{l} \text{dopředu – použít věty Teorie kom-} \\ \text{putačního učení} \\ \text{po naučení – kontrolou na jiné trénovací} \\ \text{sadě} \end{array} \right.$

používaná **metodologie** (cross validation):

1. vezmeme velkou množinu příkladů
2. rozdělíme ji na 2 množiny –  
**trénovací** a **testovací**
3. aplikujeme učící algoritmus na  
**trénovací** sadu, získáme hypotézu  $h$
4. **změříme** procento příkladů v  
**testovací** sadě, které jsou správně  
klasifikované hypotézou  $h$
5. opakujeme kroky 2–4 pro různé  
velikosti trénovacích sad a pro  
náhodně vybrané trénovací sady

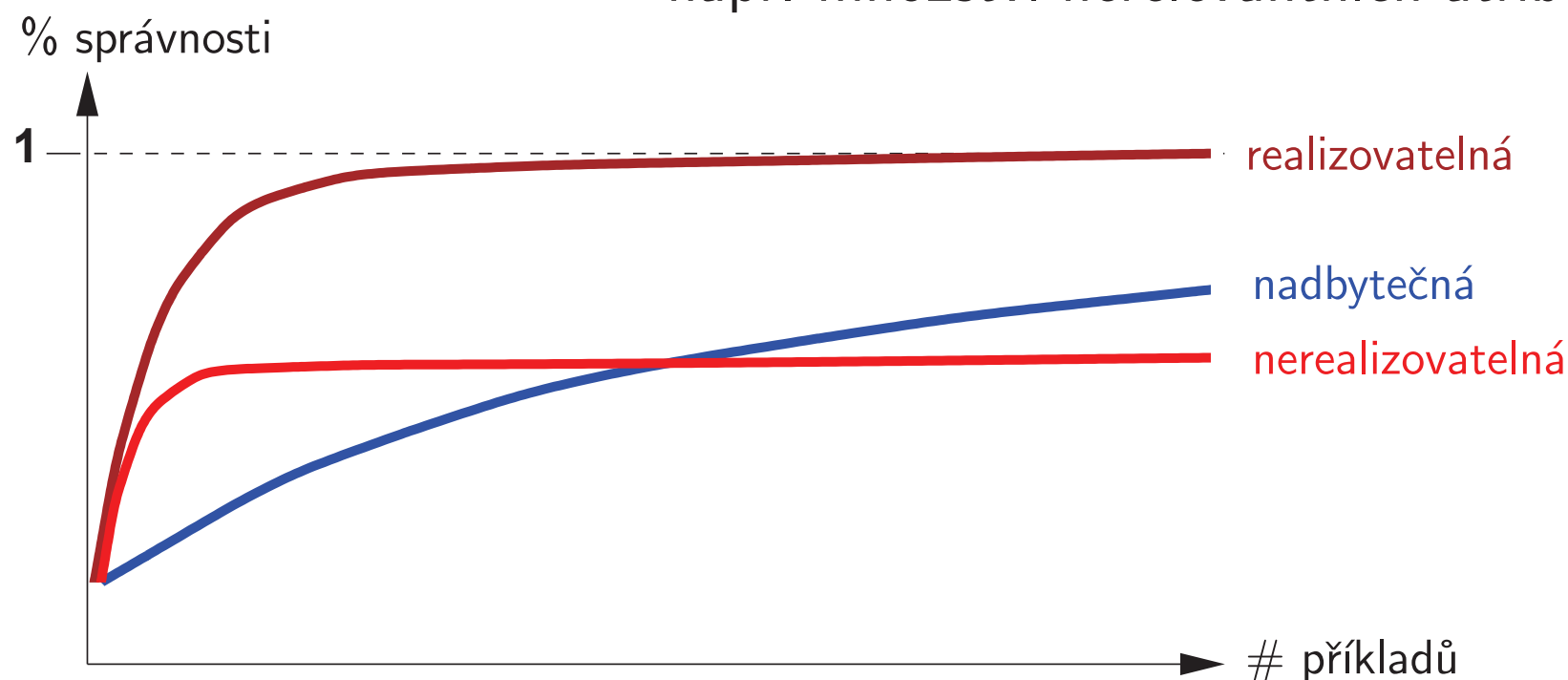
**křivka učení** – závislost ve-  
likosti trénovací sady na  
úspěšnosti



# Hodnocení úspěšnosti učícího algoritmu – pokrač.

tvár křivky učení závisí na

- je hledaná funkce realizovatelná  $\times$  nerealizovatelná  
funkce může být nerealizovatelná kvůli
  - chybějícím atributům
  - omezenému prostoru hypotéz
- naopak nadbytečné expresivitě  
např. množství nerelevantních atributů



# Induktivní učení – shrnutí

- **učení** je potřebné pro **neznámé prostředí** (a líné analytiky ☺)
- **učící se agent** – **výkonnostní komponenta** a **komponenta učení**
- **metoda** učení závisí na **typu výkonnostní komponenty**, dostupné **zpětné vazbě**, **typu** a **reprezentaci** části, která se má učením zlepšit
- u **učení s dohledem** – cíl je najít nejjednodušší hypotézu přibližně konzistentní s trénovacími příklady
- učení formou **rozhodovacích stromů** používá **míru informace**
- **kvalita učení** – přesnost odhadu změřená na testovací sadě

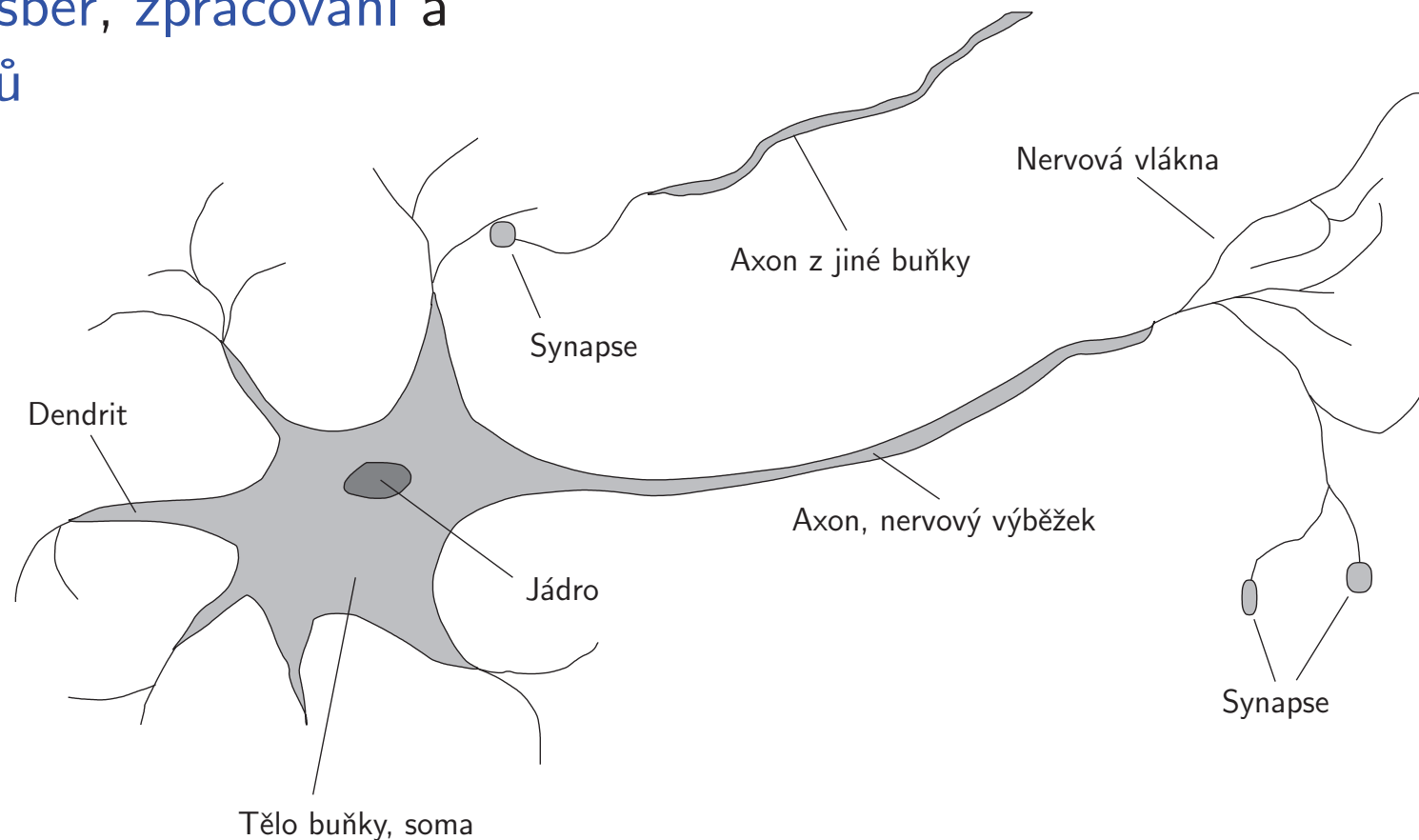
# Obsah

- 1 Učení
  - Učící se agent
  - Komponenta učení
  - Induktivní učení
- 2 Rozhodovací stromy
  - Atributová reprezentace příkladů
  - Rozhodovací stromy
  - Vyjadřovací síla rozhodovacích stromů
  - Prostor hypotéz
  - Učení ve formě rozhodovacích stromů
- 3 Hodnocení úspěšnosti učícího algoritmu
  - Induktivní učení – shrnutí
- 4 Neuronové sítě
  - Neuron
  - Počítačový model – neuronové sítě
  - Aktivační funkce
  - Logické funkce pomocí neuronové jednotky

# Neuron

mozek –  $10^{11}$  neuronů  $> 20$  typů,  $10^{14}$  synapsí, 1ms–10ms cyklus  
nosiče informace – signály = “výkyvy” elektrických potenciálů (se šumem)

neuron – mozková buňka, která  
má za úkol sběr, zpracování a  
šíření signálů

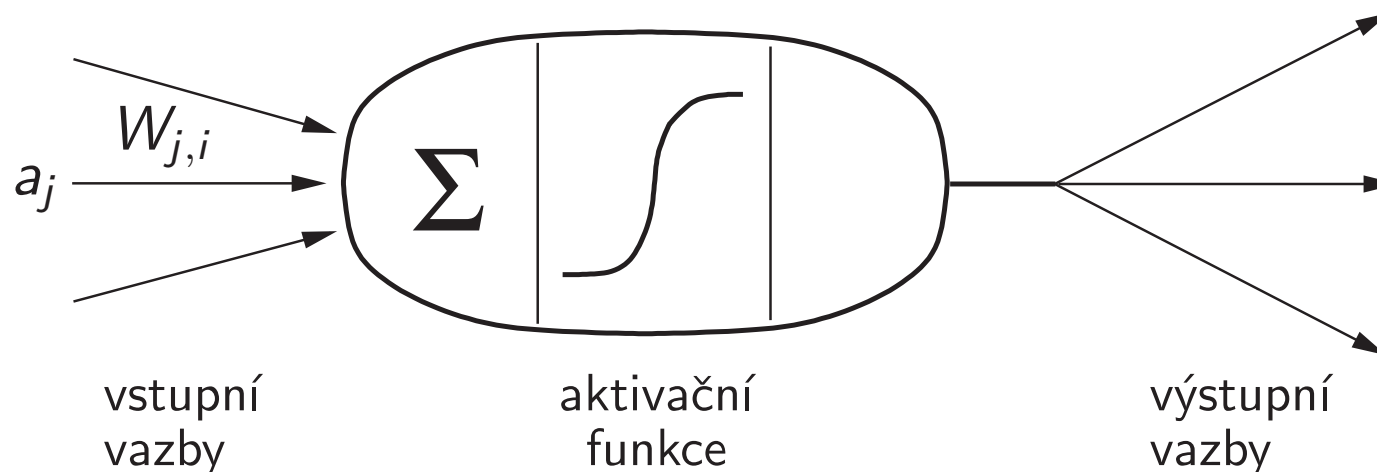


# Počítačový model – neuronové sítě

1943 – McCulloch & Pitts – matematický model neuronu  
spojené do neuronové sítě – schopnost tolerovat šum ve vstupu a učit se

jednotky v neuronové síti – jsou propojeny vazbami (*links*)  
(*units*)

- vazba z jednotky  $j$  do  $i$  propaguje aktivaci  $a_j$  jednotky  $j$
- každá vazba má číselnou váhu  $W_{j,i}$  (síla+znaménko)





# Počítačový model – neuronové sítě

1943 – McCulloch & Pitts – matematický model neuronu  
spojené do neuronové sítě – schopnost tolerovat šum ve vstupu a učit se

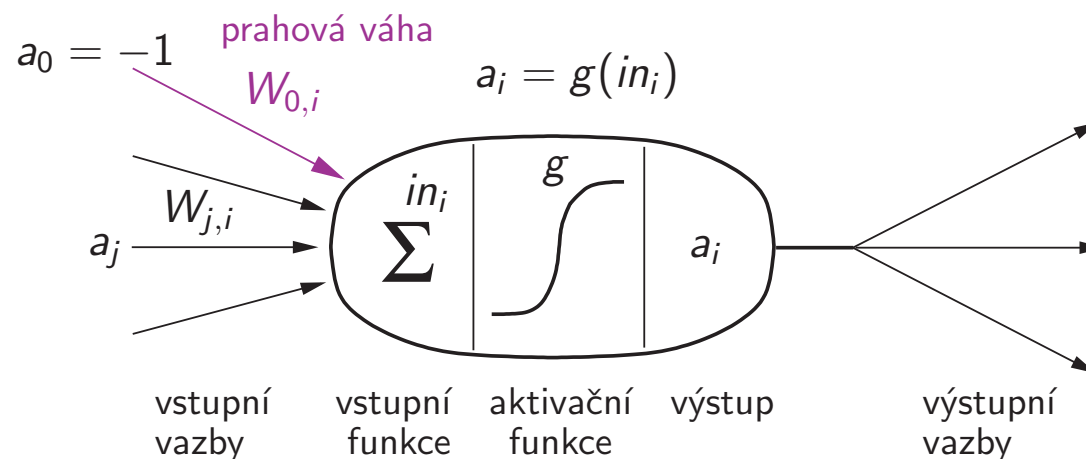
jednotky v neuronové síti – jsou propojeny vazbami (links)  
(units)

- vazba z jednotky  $j$  do  $i$  propaguje aktivaci  $a_j$  jednotky  $j$
- každá vazba má číselnou váhu  $W_{j,i}$  (síla+znaménko)

funkce jednotky  $i$ :

1. spočítá váženou  $\sum$  vstupů =  $in_i$
2. aplikuje aktivační funkci  $g$
3. tím získá výstup  $a_i$

$$a_i = g(in_i) = g\left(\sum_j W_{j,i} a_j\right)$$



# Aktivační funkce

účel **aktivační funkce**:

- jednotka má být **aktivní** ( $\approx +1$ ) pro pozitivní příklady, jinak **neaktivní**  $\approx 0$
- aktivace musí být **nelineární**, jinak by celá síť byla lineární

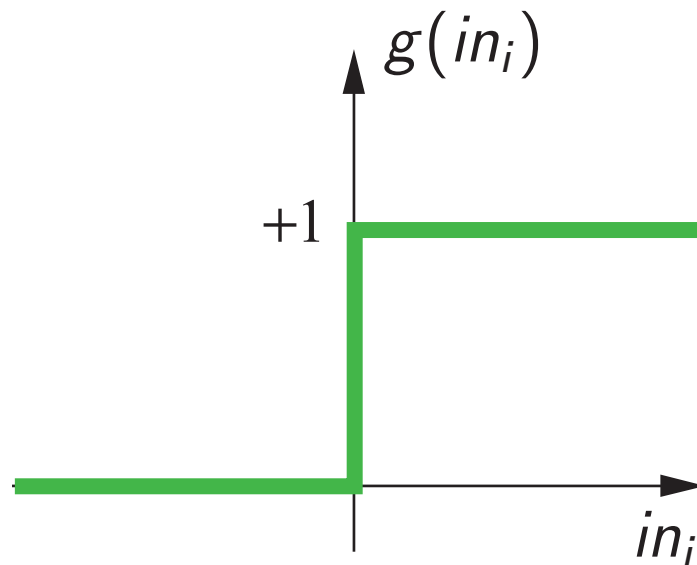
# Aktivační funkce

účel **aktivační funkce**:

- jednotka má být **aktivní** ( $\approx +1$ ) pro pozitivní příklady, jinak **neaktivní**  $\approx 0$
- aktivace musí být **nelineární**, jinak by celá síť byla lineární

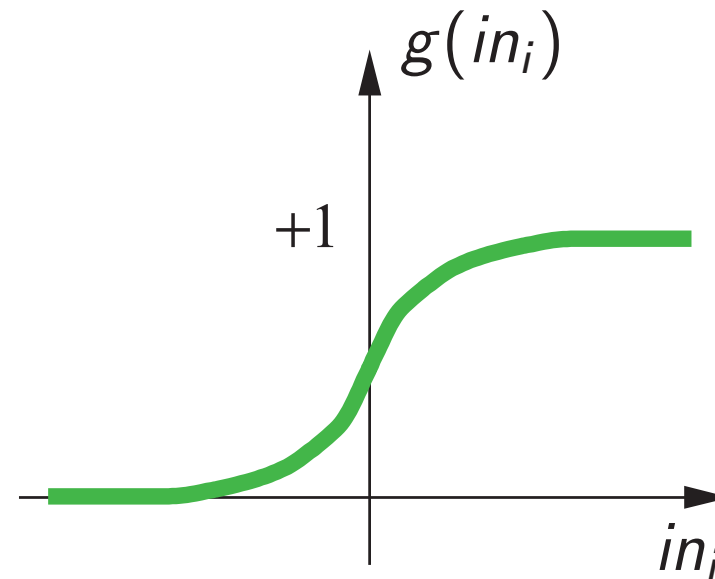
např.

a)



prahová funkce

b)



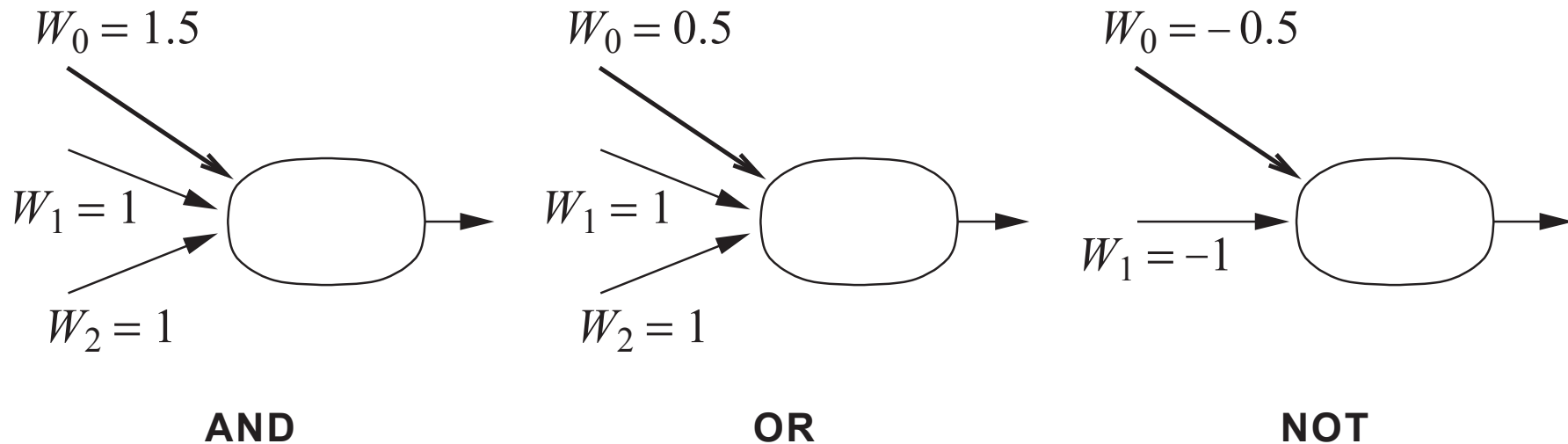
sigmoida

$$1/(1 + e^{-x})$$

je derivovatelná – důležité pro **učení**

změny **prahové váhy**  $W_{0,i}$  nastavují nulovou pozici – nastavují **práh** aktivace

# Logické funkce pomocí neuronové jednotky



jednotka McCulloch & Pitts sama umí implementovat **základní Booleovské funkce**

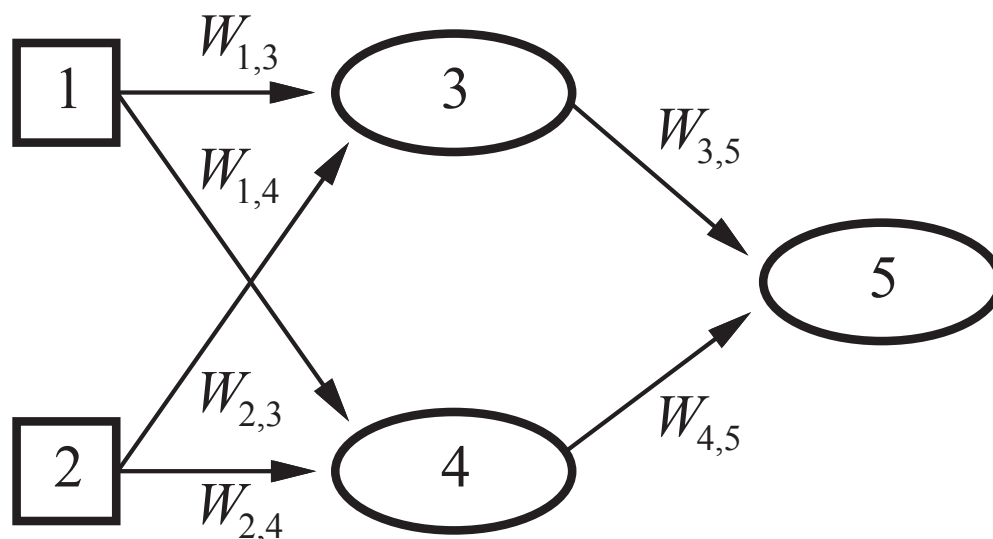
⇒ kombinacemi jednotek do sítě můžeme implementovat **libovolnou Booleovskou funkci**

# Struktury neuronových sítí

- sítě s předním vstupem (*feed-forward networks*)
  - necyklické
  - implementují funkce
  - nemají vnitřní paměť
- rekurentní sítě (*recurrent networks*)
  - cyklické
  - vlastní výstup si berou opět na vstup
  - složitější a schopnější
  - výstup má (zpožděný) vliv na aktivaci = paměť
  - Hopfieldovy sítě – symetrické obousměrné vazby; fungují jako asociativní paměť
  - Boltzmannovy stroje – pravděpodobnostní aktivační funkce

# Příklad sítě s předním vstupem

síť 5-ti jednotek – 2 vstupní jednotky, 1 skrytá vrstva (2 jednotky), 1 výstupní jednotka



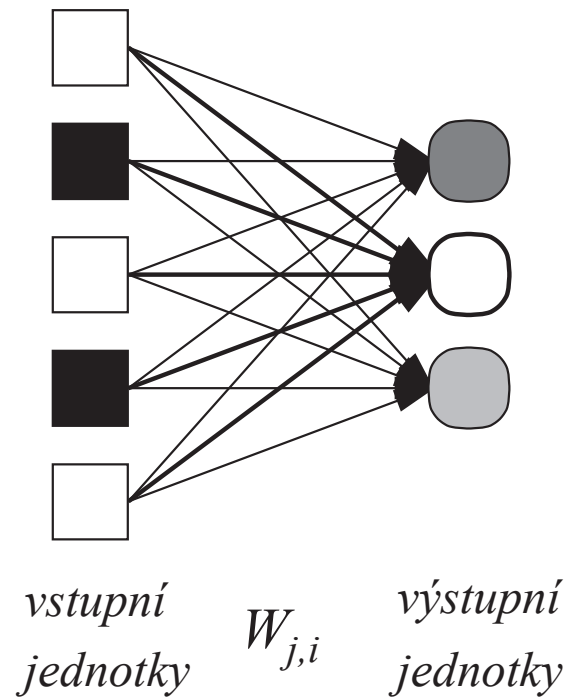
síť s předním vstupem = parametrizovaná nelineární funkce vstupu

$$\begin{aligned} a_5 &= g(W_{3,5} \cdot a_3 + W_{4,5} \cdot a_4) \\ &= g(W_{3,5} \cdot g(W_{1,3} \cdot a_1 + W_{2,3} \cdot a_2) + W_{4,5} \cdot g(W_{1,4} \cdot a_1 + W_{2,4} \cdot a_2)) \end{aligned}$$

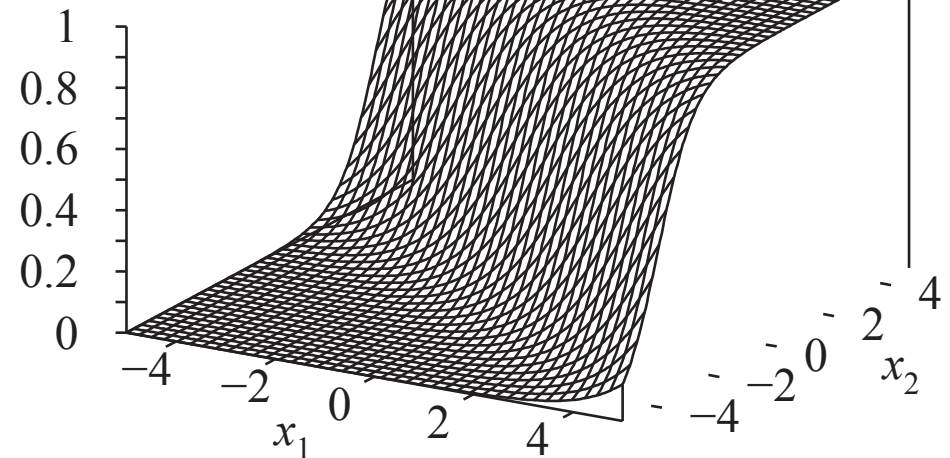
# Jednovrstvá síť – perceptron

## perceptron

- pro Booleovskou funkci 1 výstupní jednotka
- pro složitější klasifikaci – **více výstupních jednotek**



výstup perceptronu

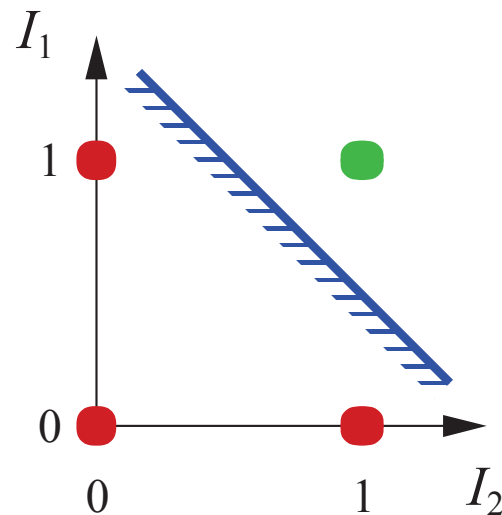


# Vyjadřovací síla perceptronu

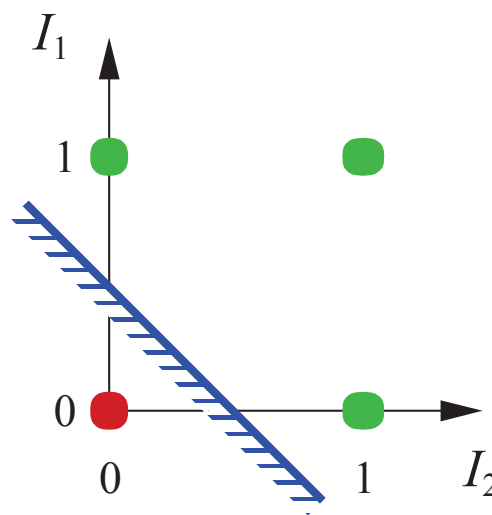
**perceptron** může reprezentovat hodně Booleovských funkcí – AND, OR, NOT, majoritní funkci, ...

$$\sum_j W_j x_j > 0 \quad \text{nebo} \quad \mathbf{W} \cdot \mathbf{x} > 0$$

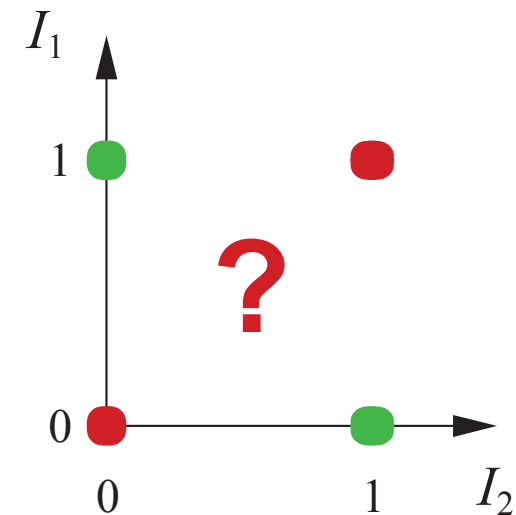
reprezentuje **lineární separátor** (nádřovina) v prostoru vstupu:



a)  $I_1$  **and**  $I_2$



b)  $I_1$  **or**  $I_2$



c)  $I_1$  **xor**  $I_2$



# Učení perceptronu

**výhoda** perceptronu – existuje jednoduchý **učící algoritmus** pro libovolnou lineárně separabilní funkci

**učení perceptronu** = upravování vah, aby se **snížila chyba** na trénovací sadě

# Učení perceptronu

**výhoda** perceptronu – existuje jednoduchý **učící algoritmus** pro libovolnou lineárně separabilní funkci

**učení perceptronu** = upravování vah, aby se **snížila chyba** na trénovací sadě

**kvadratická chyba**  $E$  pro příklad se vstupem  $\mathbf{x}$  a požadovaným (=správným) výstupem  $y$  je

$$E = \frac{1}{2} Err^2 \equiv \frac{1}{2} (y - h_{\mathbf{w}}(\mathbf{x}))^2, \quad \text{kde } h_{\mathbf{w}}(\mathbf{x}) \text{ je výstup perceptronu}$$

# Učení perceptronu

**výhoda** perceptronu – existuje jednoduchý **učící algoritmus** pro libovolnou lineárně separabilní funkci

**učení perceptronu** = upravování vah, aby se **snížila chyba** na trénovací sadě

**kvadratická chyba**  $E$  pro příklad se vstupem  $\mathbf{x}$  a požadovaným (=správným) výstupem  $y$  je

$$E = \frac{1}{2} Err^2 \equiv \frac{1}{2} (y - h_{\mathbf{W}}(\mathbf{x}))^2, \quad \text{kde } h_{\mathbf{W}}(\mathbf{x}) \text{ je výstup perceptronu}$$

**váhy pro minimální chybu** pak hledáme **optimalizačním prohledáváním** spojitého prostoru vah

$$\frac{\partial E}{\partial W_j} = Err \times \frac{\partial Err}{\partial W_j} = Err \times \frac{\partial}{\partial W_j} (y - g(\sum_{j=0}^n W_j x_j)) = -Err \times g'(in) \times x_j$$

**pravidlo pro úpravu váhy**

$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j \quad \alpha \dots \text{učící konstanta (learning rate)}$$

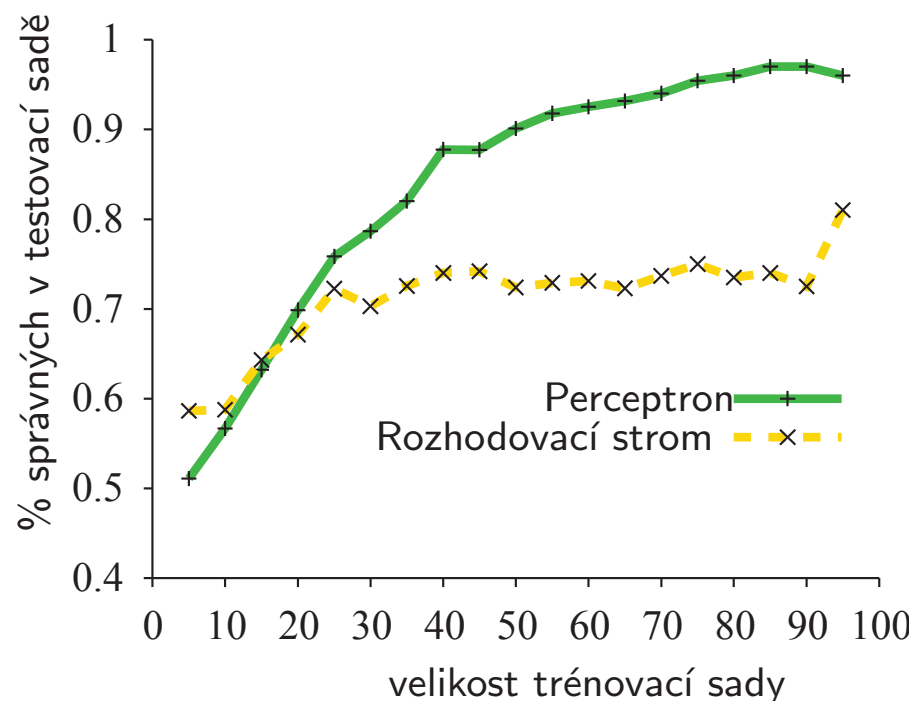
např.  $Err = y - h_{\mathbf{W}}(\mathbf{x}) > 0 \Rightarrow$  výstup  $h_{\mathbf{W}}(\mathbf{x})$  je moc malý  
 $\Rightarrow$  váhy se musí **zvýšit** pro pozitivní příklady a **snížit** pro negativní

úpravu vah provádíme po každém příkladu  $\rightarrow$  opakovaně až do dosažení **ukončovacího kritéria**

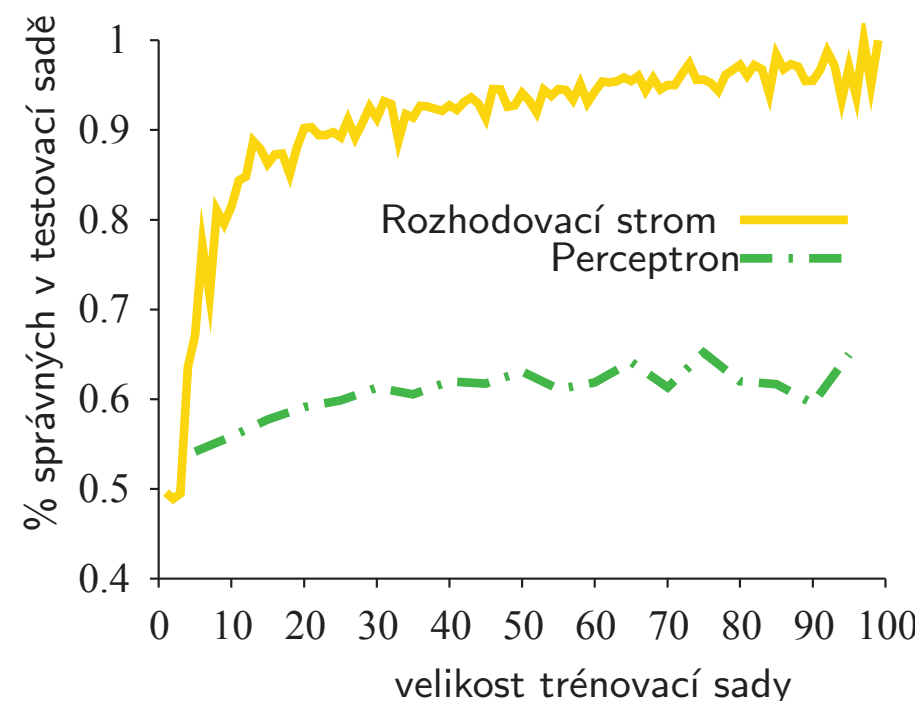
# Učení perceptronu pokrač.

učicí pravidlo pro perceptron **konverguje ke správné funkci** pro libovolnou **lineárně separabilní** množinu dat

## a) učení majoritní funkce



## b) učení čekání na volný stůl v restauraci



# Vícevrstvé neuronové sítě

vrstvy jsou obvykle úplně propojené

počet **skrytých jednotek** je obvykle volen experimentálně

výstupní jednotky

$a_i$

$W_{j,i}$

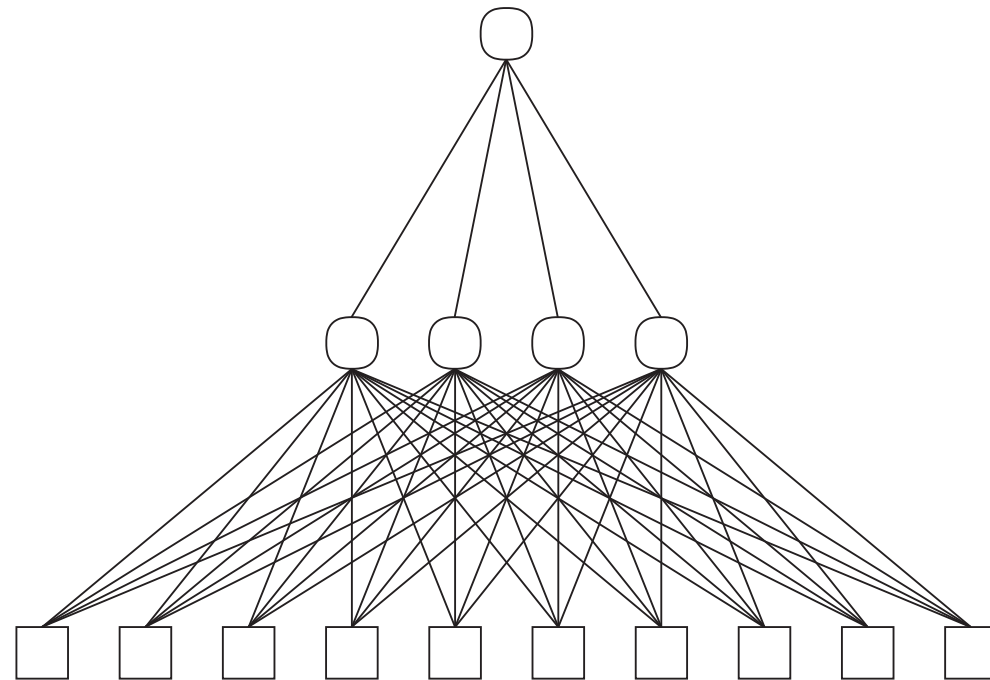
skryté jednotky

$a_j$

$W_{k,j}$

vstupní jednotky

$a_k$



# Vyjadřovací síla vícevrstevných sítí

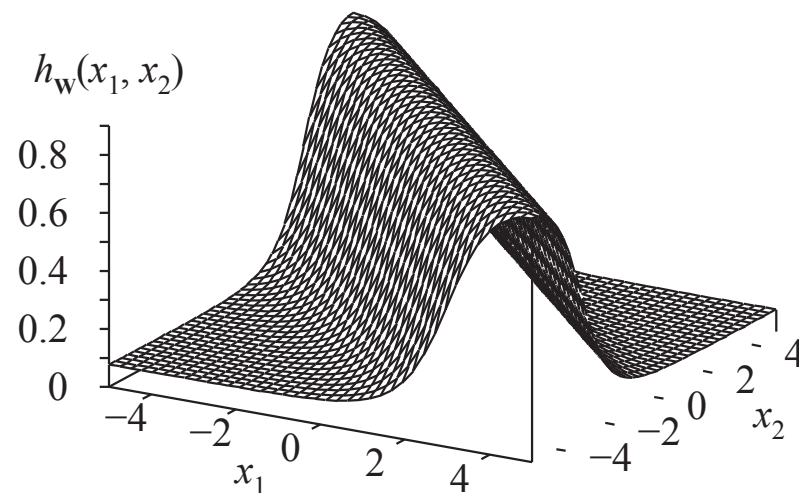
s jednou skrytou vrstvou – všechny **spojité funkce**

se dvěma skrytými vrstvami – **všechny funkce**

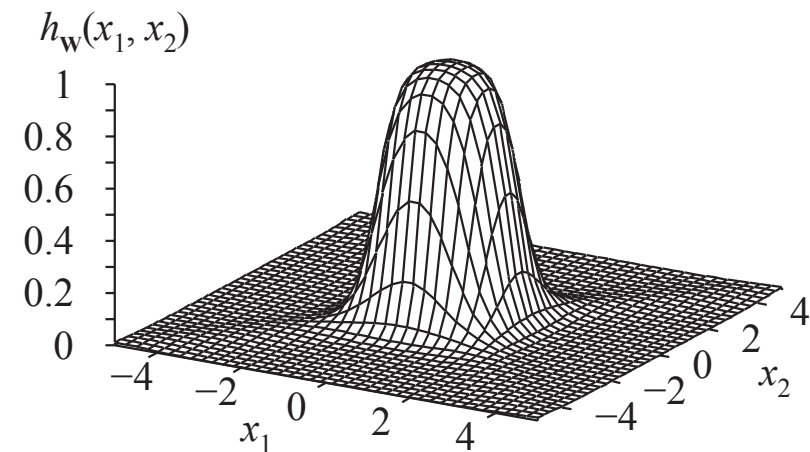
těžko se ovšem pro **konkrétní síť** zjišťuje její prostor **reprezentovatelných funkcí**

např.

dvě “opačné” skryté jednotky  
vytvoří *hřbet*



dva hřebety vytvoří *homoli*



# Učení vícevrstevných sítí

pravidla pro úpravu vah:

- **výstupní vrstva** – stejně jako u perceptronu

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i \quad \text{kde} \quad \Delta_i = Err_i \times g'(in_i)$$

# Učení vícevrstevných sítí

pravidla pro úpravu vah:

- výstupní vrstva – stejně jako u perceptronu

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i \quad \text{kde} \quad \Delta_i = Err_i \times g'(in_i)$$

- skryté vrstvy – zpětné šíření (*back-propagation*) chyby z výstupní vrstvy

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j \quad \text{kde} \quad \Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i$$



# Učení vícevrstevných sítí

pravidla pro úpravu vah:

- **výstupní vrstva** – stejně jako u perceptronu

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i \quad \text{kde} \quad \Delta_i = Err_i \times g'(in_i)$$

- **skryté vrstvy** – **zpětné šíření** (*back-propagation*) chyby z výstupní vrstvy

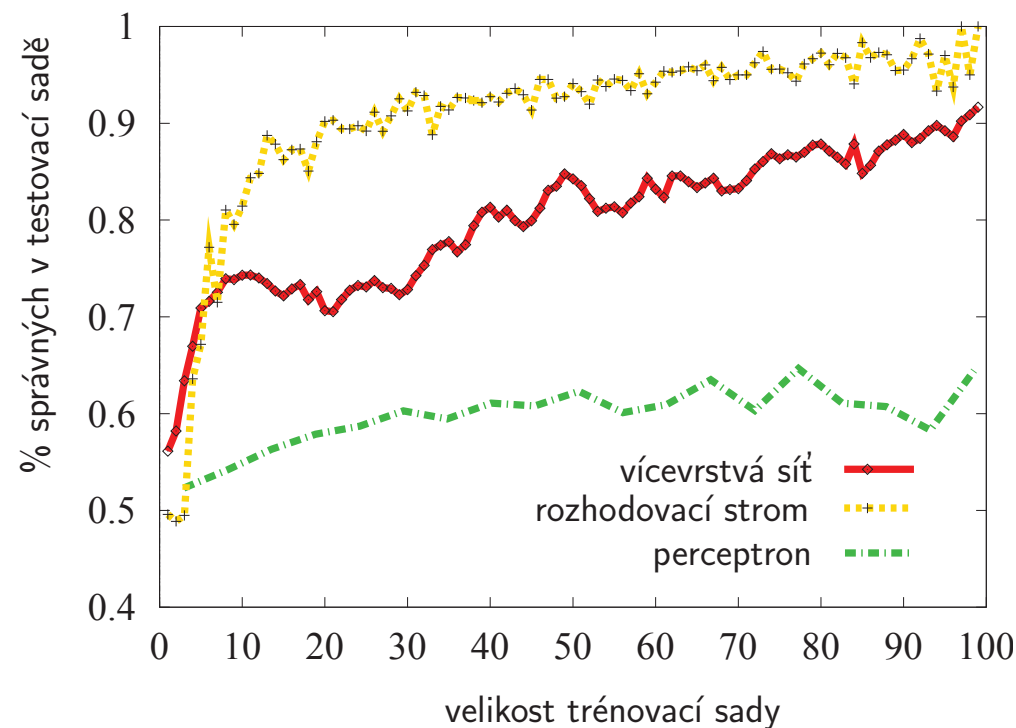
$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j \quad \text{kde} \quad \Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i$$

problémy učení:

- dosažení **lokálního minima** chyby
- příliš **pomalá konvergence**
- přílišné **upnutí** na příklady → neschopnost generalizovat

# Učení vícevrstvých sítí pokrač.

vícevrstvá síť se problémem čekání na volný stůl v restauraci **učí znatelně líp** než perceptron



# Neuronové sítě – shrnutí

- většina mozků má **velké množství** neuronů; každý **neuron**  $\approx$  lineární prahová jednotka (?)
- **perceptrony** (jednovrstvé sítě) mají **nízkou** vyjadřovací sílu
- **vícevrstvé sítě** jsou **dostatečně silné**; mohou být trénovány pomocí zpětného šíření chyby
- velké množství reálných aplikací
  - rozpoznávání řeči
  - řízení auta
  - rozpoznávání ručně psaného písma
  - ...