

## Učení, rozhodovací stromy, neuronové sítě

Aleš Horák

E-mail: [hales@fi.muni.cz](mailto:hales@fi.muni.cz)

<http://nlp.fi.muni.cz/uui/>

Obsah:

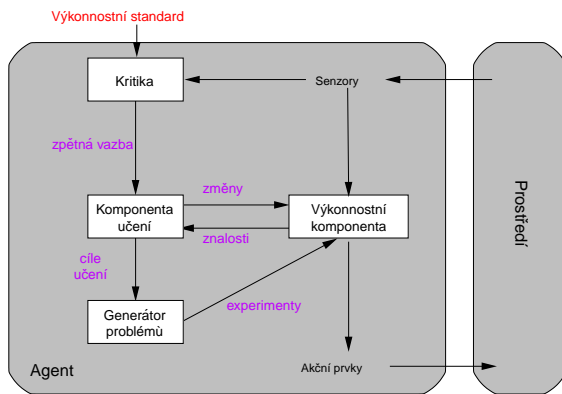
- Učení
- Rozhodovací stromy
- Neuronové sítě

## Učení

### UČENÍ

- **učení** je klíčové pro neznámé prostředí (kde návrhář není vševědoucí)
- učení je také někdy vhodné jako **metoda konstrukce** systému – vystavit agenta realitě místo přepisování reality do pevných pravidel
- učení agenta – využití jeho **vjemů** z prostředí nejen pro vyvození další akce
- učení **modifikuje rozhodovací systém** agenta pro zlepšení jeho výkonnosti

## UČÍCÍ SE AGENT



příklad automatického taxi:

- ❑ **Výkonnostní komponenta** – obsahuje znalosti a postupy pro výběr akcí pro vlastní řízení auta
- ❑ **Kritika** – sleduje reakce okolí na akce taxi. Např. při rychlém přejetí 3 podélných pruhů zaznamenaná a předá pohoršující reakce dalších řidičů
- ❑ **Komponenta učení** – z hlášení Kritiky vyvodí nové pravidlo, že takové přeježdění je nevhodné, a modifikuje odpovídajícím způsobem Výkonnostní komponentu
- ❑ **Generátor problémů** – zjišťuje, které oblasti by mohly potřebovat vylepšení a navrhuje experimenty, jako je třeba brždění na různých typech vozovky

## KOMPONENTA UČENÍ

návrh komponenty učení závisí na několika attributech:

- jaký typ výkonnostní komponenty je použit
- která funkční část výkonnostní komponenty má být učena
- jak je tato funkční část reprezentována
- jaká zpětná vazba je k dispozici

příklady:

výkonnostní komponenta	funkční část	reprezentace	zpětná vazba
Alfa-beta prohledávání	<b>vyhodnocovací funkce</b>	<b>vážená lineární funkce</b>	<b>výhra/prohra</b>
Logický agent	<b>určení akce</b>	<b>axiomy <i>Result</i></b>	<b>výsledné skóre</b>
Reflexní agent	<b>váhy preceptronu</b>	<b>neuronová síť</b>	<b>správná/špatná akce</b>

učení **s dohledem** (*supervised learning*) × **bez dohledu** (*unsupervised learning*)

- ❑ **s dohledem** – učení **funkce** z příkladů vstupů a výstupů
- ❑ **bez dohledu** – učení **vzorů** na vstupu vzhledem k reakcím prostředí
- ❑ **posílené** (*reinforcement learning*) – nejobecnější, agent se učí podle **odměn/pokut**

## INDUKTIVNÍ UČENÍ

známé taky jako **věda** 😊

nejjednodušší forma – učení funkce z příkladů (agent je tabula rasa)

$f$  je **cílová funkce**

příklad je dvojice  $x, f(x)$  např.

o	o	x
	x	
x		

, +1

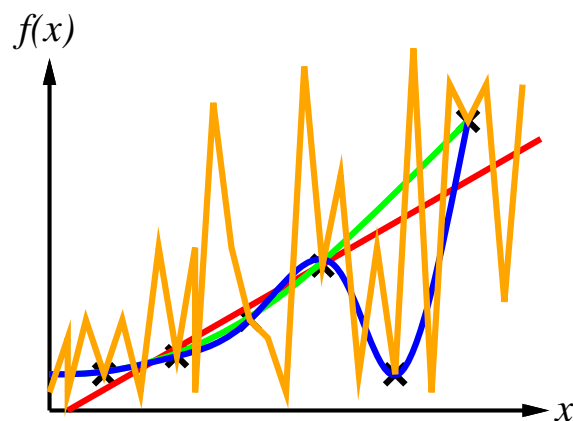
úkol **indukce**: najdi **hypotézu**  $h$   
 takovou, že  $h \approx f$   
 pomocí sady **trénovacích příkladů**

## METODA INDUKTIVNÍHO UČENÍ

zkonstruuuj/uprav  $h$ , aby souhlasila s  $f$  na trénovacích příkladech

$h$  je **konzistentní**  $\Leftrightarrow$  souhlasí s  $f$  na všech příkladech

např. hledání křivky:



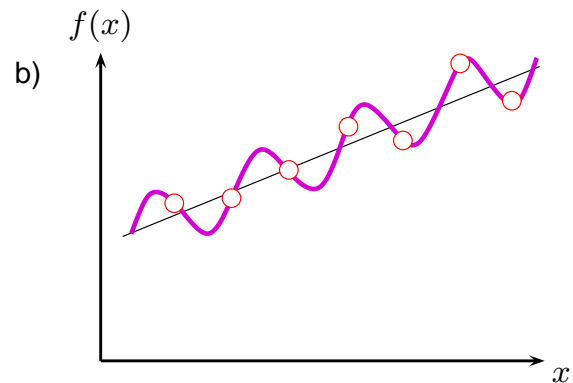
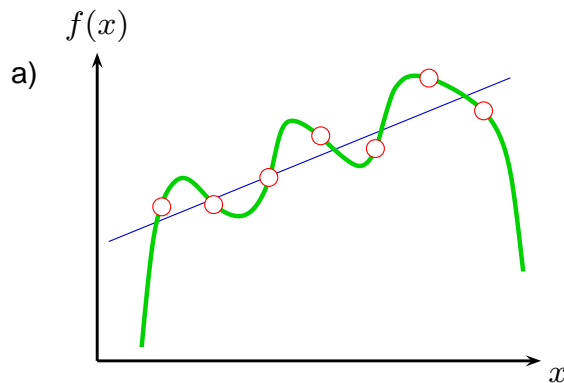
pravidlo **Ockhamovy břitvy** – maximalizovat kombinaci konzistence a jednoduchosti (*nejjednodušší ze správných je nejlepší*)

## METODA INDUKTIVNÍHO UČENÍ pokrač.

hodně záleží na **prostoru hypotéz**, jsou na něj protichůdné požadavky:

- pokrýt co **největší množství** hledaných funkcí
- udržet **nízkou výpočetní složitost** hypotézy

např.



- stejná sada 7 bodů
- nejmenší konzistentní polynom – polynom 6-tého stupně (7 parametrů)
- může být výhodnější použít nekonzistentní **přibližnou** lineární funkci
- přitom existuje konzistentní funkce  $ax + by + c \sin x$

## ATRIBUTOVÁ REPREZENTACE PŘÍKLADŮ

příklady popsané výčtem **hodnot atributů** (libovolných hodnot)

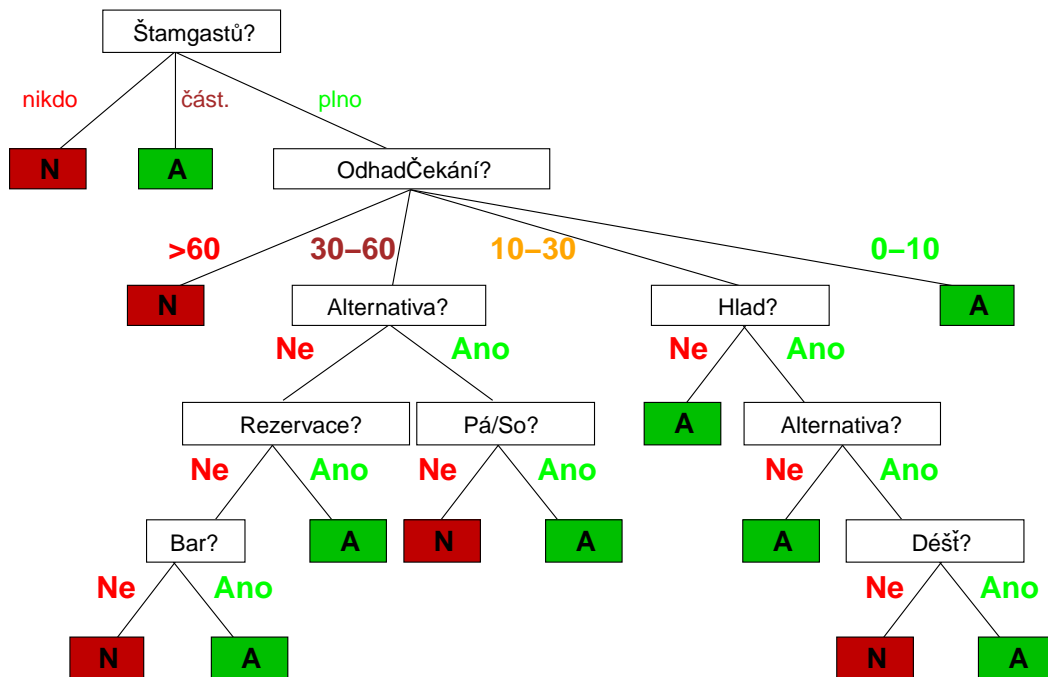
např. rozhodování, zda počkat na uvolnění stolu v restauraci:

Příklad	Atributy										počkat?
	<i>Alt</i>	<i>Bar</i>	<i>Pá/So</i>	<i>Hlad</i>	<i>Štam</i>	<i>Cen</i>	<i>Děšť'</i>	<i>Rez</i>	<i>Typ</i>	<i>ČekD</i>	
$X_1$	A	N	N	A	část.	\$\$\$	N	A	mexická	0–10	A
$X_2$	A	N	N	A	plno	\$	N	N	asijská	30–60	N
$X_3$	N	A	N	N	část.	\$	N	N	bufet	0–10	A
$X_4$	A	N	A	A	plno	\$	N	N	asijská	10–30	A
$X_5$	A	N	A	N	plno	\$\$\$	N	A	mexická	>60	N
$X_6$	N	A	N	A	část.	\$\$	A	A	pizzerie	0–10	A
$X_7$	N	A	N	N	nikdo	\$	A	N	bufet	0–10	N
$X_8$	N	N	N	A	část.	\$\$	A	A	asijská	0–10	A
$X_9$	N	A	A	N	plno	\$	A	N	bufet	>60	N
$X_{10}$	A	A	A	A	plno	\$\$\$	N	A	pizzerie	10–30	N
$X_{11}$	N	N	N	N	nikdo	\$	N	N	asijská	0–10	N
$X_{12}$	A	A	A	A	plno	\$	N	N	bufet	30–60	A

Ohodnocení tvoří **klasifikaci** příkladů – pozitivní (A) a negativní (N)

## ROZHODOVACÍ STROMY

jedna z možných reprezentací hypotéz – **rozhodovací strom** pro určení, jestli počkat na stůl:



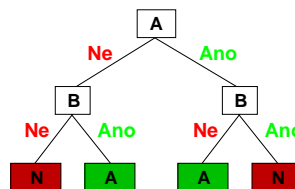
## VYJADŘOVACÍ SÍLA ROZHODOVACÍCH STROMŮ

rozhodovací stromy vyjádří libovolnou **Booleovskou funkci vstupních atributů** → odpovídá **výrokové logice**

$$\forall s \text{ počkat?}(s) \Leftrightarrow (P_1(s) \vee P_2(s) \vee \dots \vee P_n(s)), \quad \text{kde } P_i(s) = (A_1(s) = V_1 \wedge \dots \wedge A_m(s) = V_m)$$

pro libovolnou Booleovskou funkci → řádek v pravdivostní tabulce = **cesta ve stromu** (od kořene k listu)

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



triviálně

*pro libovolnou trénovací sadu existuje konzistentní rozhodovací strom s jednou cestou k listům pro každý příklad*

ale takový strom pravděpodobně nebude generalizovat na nové příklady

chceme najít co možná **kompaktní** rozhodovací strom

## PROSTOR HYPOTÉZ

1. vezměme pouze Booleovské atributy, bez dalšího omezení

**Kolik existuje různých rozhodovacích stromů s  $n$  Booleovskými atributy?**

= počet všech Booleovských funkcí nad těmito atributy

= počet různých pravdivostních tabulek s  $2^n$  řádky =  $2^{2^n}$

např. pro 6 atributů existuje 18 446 744 073 709 551 616 různých rozhodovacích stromů

2. když se omezíme pouze na konjunktivní hypotézy ( $Hlad \wedge \neg Déšť'$ )

**Kolik existuje takových čistě konjunktivních hypotéz?**

každý atribut může být v pozitivní nebo negativní formě nebo nepoužit

$\Rightarrow 3^n$  různých konjunktivních hypotéz (pro 6 atributů = 729)

**prostor hypotéz s větší expresivitou**

- **zvyšuje šance**, že najdeme přesné vyjádření cílové funkce
- ALE **zvyšuje i počet** možných hypotéz, které jsou konzistentní s trénovací množinou  
 $\Rightarrow$  můžeme získat **nižší kvalitu** předpovědí (generalizace)

## UČENÍ VE FORMĚ ROZHODOVACÍCH STROMŮ

### triviální konstrukce rozhodovacího stromu

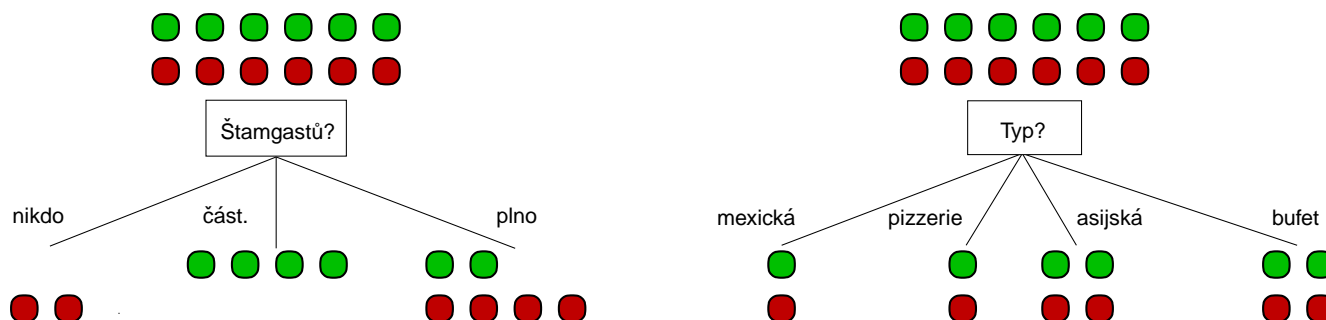
- pro každý příklad v trénovací sadě přidej jednu cestu od kořene k listu
- na stejných příkladech jako v trénovací sadě bude fungovat přesně
- na nových příkladech se bude chovat náhodně – **negeneralizuje** vzory z příkladů, pouze **kopíruje** pozorování

### heuristická konstrukce kompaktního stromu

- chceme najít **nejmenší** rozhodovací strom, který souhlasí s příklady
- vlastní nalezení nejmenšího stromu je ovšem příliš složité  
 $\rightarrow$  heuristikou najdeme alespoň **dostatečně malý** 😊
- hlavní myšlenka – vybíráme atributy pro test v co nejlepším **pořadí**

## VÝBĚR ATRIBUTU

myšlenka – **dobry atribut** rozdělí příklady na podmnožiny, které jsou (nejlépe) “všechny pozitivní” nebo “všechny negativní”



*Štamgastů?* je lepší volba atributu ← dává lepší **informaci** o vlastní **klasifikaci** příkladů

## VÝBĚR ATRIBUTU – MÍRA INFORMACE

**informace** – odpovídá na **otázku**

čím **méně** dopředu vím o výsledku obsaženém v odpovědi → tím **více** informace je v ní obsaženo

měřítka:

**1 bit** = odpověď na Booleovskou otázku s pravděpodobnostmi odpovědi  $\langle P(T) = \frac{1}{2}, P(F) = \frac{1}{2} \rangle$

pro pravděpodobnosti všech odpovědí  $\langle P(v_1), \dots, P(v_n) \rangle \rightarrow$  **míra informace** v odpovědi obsažená

$$I(\langle P(v_1), \dots, P(v_n) \rangle) = \sum_{i=1}^n -P(v_i) \log_2 P(v_i)$$

tato míra se také nazývá **entropie**

např. pro házení mincí:  $I(\langle \frac{1}{2}, \frac{1}{2} \rangle) = -\frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{2} \log_2 \frac{1}{2} = 1$  bit

pro házení *falešnou* mincí, která dává na 99% vždy jednu stranu mince:

$$I(\langle \frac{1}{100}, \frac{99}{100} \rangle) = -\frac{1}{100} \log_2 \frac{1}{100} - \frac{99}{100} \log_2 \frac{99}{100} = 0.08 \text{ bitů}$$

## POUŽITÍ MÍRY INFORMACE PRO VÝBĚR ATRIBUTU

předpokládejme, že máme  $p$  pozitivních a  $n$  negativních příkladů

$\Rightarrow I\left(\left\langle \frac{p}{p+n}, \frac{n}{p+n} \right\rangle\right)$  bitů je potřeba pro klasifikaci nového příkladu

např. pro  $X_1, \dots, X_{12}$  z volby čekání na stůl je  $p = n = 6$ , takže potřebujeme 1 bit

**výběr atributu** – kolik informace nám dá test na hodnotu atributu  $A$ ?

= **rozdíl** odhadu odpovědi **před** a **po** testu atributu

atribut  $A$  rozdělí sadu příkladů  $E$  na podmnožiny  $E_i$  (nejlépe, že  $\forall$  potřebuje méně informace)

nechť  $E_i$  má  $p_i$  pozitivních a  $n_i$  negativních příkladů

$\Rightarrow$  je potřeba  $I\left(\left\langle \frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i} \right\rangle\right)$  bitů pro klasifikaci nového příkladu

$\Rightarrow$  **očekávaný** počet bitů přes  $\forall$  větve je  $Remainder(A) = \sum_i \frac{p_i+n_i}{p+n} I\left(\left\langle \frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i} \right\rangle\right)$

$\Rightarrow$  výsledný **zisk atributu**  $A$  je  $Gain(A) = I\left(\left\langle \frac{p}{p+n}, \frac{n}{p+n} \right\rangle\right) - Remainder(A)$

výběr atributu = nalezení atributu s nejvyšší hodnotou  $Gain(A)$

$Gain(\check{S}tamgastů?) \approx 0.541$  bitů       $Gain(Typ?) = 0$  bitů

## ALGORITMUS IDT – UČENÍ FORMOU ROZHODOVACÍCH STROMŮ

```
% induce_tree( +Attributes, +Examples, - Tree)
induce_tree( _, [], null) :- !.
induce_tree( _, [example( Class,_) | Examples], leaf( Class)) :-
    \+ (member( example( ClassX, _), Examples), ClassX \== Class), !. % \forall příklady stejné klasifikace
induce_tree( Attributes, Examples, tree( Attribute, SubTrees)) :-
    choose_attribute( Attributes, Examples, Attribute), !,
    del( Attribute, Attributes, RestAtts), attribute( Attribute, Values),
    induce_trees( Attribute, Values, RestAtts, Examples, SubTrees).
induce_tree( _, Examples, leaf( ExClasses)) :- % žádný užitečný atribut, list s distribucí klasifikací
    findall( Class, member( example( Class, _), Examples), ExClasses).

% induce_trees( +Att, +Values, +RestAtts, +Examples, - SubTrees):
% najdi podstromy SubTrees pro podmnožiny příkladů Examples podle hodnot (Values) atributu Att
induce_trees( _, [], -, -, [] ). % No attributes, no subtrees
induce_trees( Att, [Val1 | Vals], RestAtts, Exs, [Val1 : Tree1 | Trees]) :-
    attval_subset( Att = Val1, Exs, ExampleSubset),
    induce_tree( RestAtts, ExampleSubset, Tree1),
    induce_trees( Att, Vals, RestAtts, Exs, Trees).

% attval_subset( +Attribute = +Value, +Examples, - Subset):
% Subset je podmnožina příkladů z Examples, které splňují podmínku Attribute = Value
attval_subset( AttributeValue, Examples, ExampleSubset) :-
    findall( example( Class, Obj),
        (member( example( Class, Obj), Examples), satisfy( Obj, [ AttributeValue])),
        ExampleSubset).
% satisfy( Object, Description)
satisfy( Object, Conj) :- \+ (member( Att = Val, Conj), member( Att = ValX, Object), ValX \== Val).
```



## ALGORITMUS IDT – UČENÍ FORMOU ROZHODOVACÍCH STROMŮ pokrač.

```

% vybíráme atribut podle "čistoty" množin, na které rozdělí příklady, setof je setřídí podle Impurity
choose_attribute( Atts, Examples, BestAtt) :-
    setof( Impurity/Att, (member( Att, Atts), impurity(Examples, Att, Impurity)), [MinImpurity/BestAtt|-]).
impurity( Exs, Att, Imp) :- attribute( Att, AttVals), sumv(Att, AttVals, Exs, Rem), Imp is 0 - Rem.

% sumv(+Att, +AttVals, +Exs, -Rem) - "zbytková informace" po testu na  $\forall$  hodnoty atributu Att
sumv(-, [], -, 0).
sumv(Att, [V | Vs], Es, Rem) :-
    setof( Class, X^example(Class,X), Classes), % množina  $\forall$  Class, viz help( setof )
    sumc(Att, V, Classes, S), sumv(Att, Vs, Es, Rem1),
    findall( 1, example(Class,-), L), length(L, Nall), % spočítáme Nall a Nv
    findall( 1, (example(-, AVs), member(Att = V, AVs)), L1), length(L1, Nv),
    Pv is Nv / Nall, % P(v)
    Rem is Pv * S + Rem1.
% sumc(+Att, +V, +Classes, -S) - "vnitřní" suma
sumc(-, -, [], 0).
sumc(Att, V, [C | Classes], S) :-
    findall( 1, (example(-, AVs), member(Att = V, AVs)), L1), length(L1, Nv),
    findall( 1, (example(C, AVs), member(Att = V, AVs)), L2), length(L2, Ncv),
    sumc(Att, V, Classes, S1),
    ((Nv = 0, ! ; Ncv = 0), S is S1, ! ; % P(c|v)
    Pcv is Ncv / Nv, log2(Pcv, LogPcv), S is Pcv * LogPcv + S1).
% log2(+X, -Y)
log2(X, Y) :- Y is log(X) / log(2).

```

## ALGORITMUS IDT – PŘÍKLAD

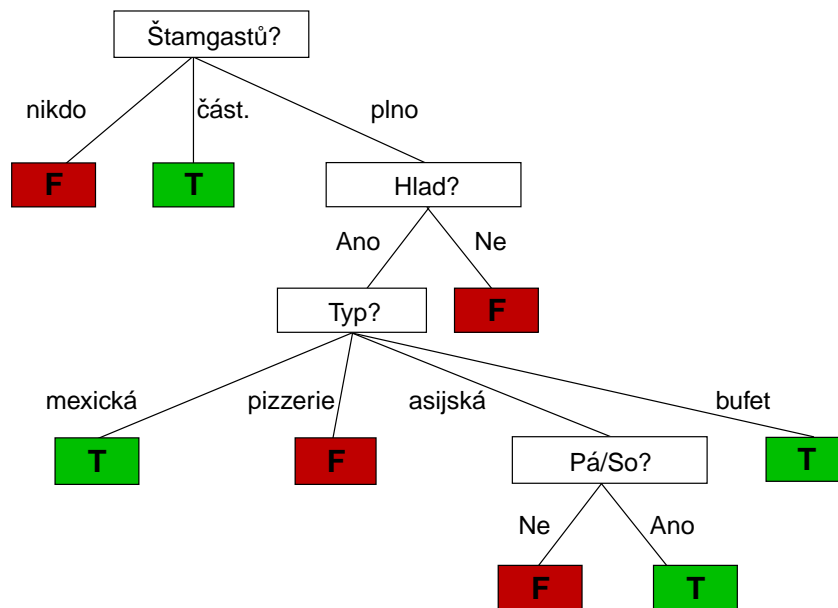
```

attribute( hlad, [ano, ne]).
attribute( stam, [nikdo, cast, plno]).
attribute( cen, ['$', '$$', '$$$']).
...
example(pockat, [alt=ano, bar=ne, paso=ne, hlad=ano, stam=cast,
    cen='$$$', dest=ne, rez=ano, typ=mexicka ]).
example(necekat, [alt=ano, bar=ne, paso=ne, hlad=ano, stam=plno,
    cen='$', dest=ne, rez=ne, typ=asijska ]).
...
:-induce_tree(T),show(T).
stam?
= nikdo
  necekat
= cast
  pockat
= plno
  hlad?
  = ano
    cen?
    = $
      paso?
      = ano
        pockat
        = ne
          necekat
      = $$$
        necekat
  = ne
    necekat

```

## IDT – VÝSLEDNÝ ROZHODOVACÍ STROM

rozhodovací strom **naučený** z 12-ti příkladů:



podstatně jednodušší než strom “z tabulky příkladů”

## HODNOCENÍ ÚSPĚŠNOSTI UČÍČÍHO ALGORITMU

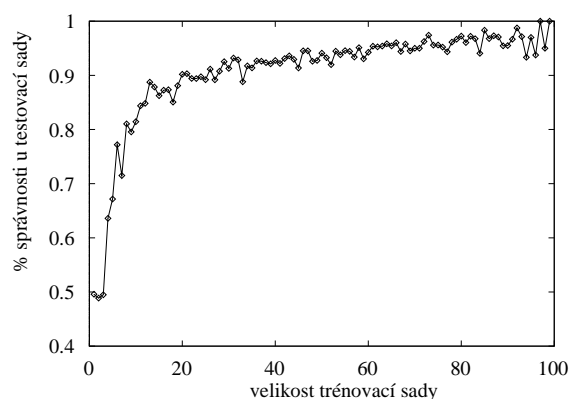
jak můžeme zjistit, zda  $h \approx f$  ?

- ↳ dopředu — použít věty Teorie počítačného učení
- ↳ po naučení — kontrolou na **jiné trénovací sadě**

používaná **metodologie**:

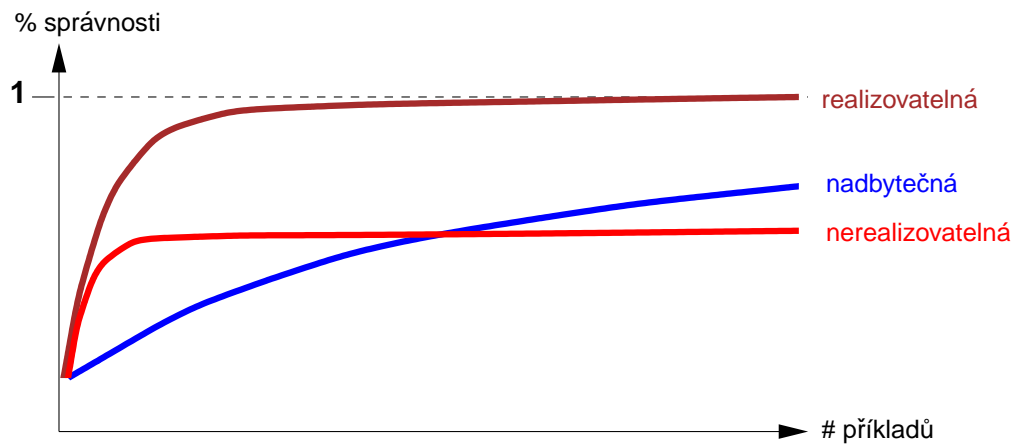
1. vezmeme velkou množinu příkladů
2. rozdělíme ji na 2 množiny – **trénovací a testovací**
3. aplikujeme učící algoritmus na *trénovací* sadu, získáme hypotézu  $h$
4. změříme procento příkladů v *testovací* sadě, které jsou správně klasifikované hypotézou  $h$
5. opakujeme kroky 2–4 pro různé velikosti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

**křivka učení** – závislost velikosti trénovací sady na úspěšnosti



## HODNOCENÍ ÚSPĚŠNOSTI UČÍCÍHO ALGORITMU pokrač.

- tvár křivky učení závisí na → je hledaná funkce **realizovatelná** × **nerealizovatelná**  
 funkce může být nerealizovatelná kvůli
- chybějícím atributům
  - omezenému prostoru hypotéz
- naopak **nadbytečné expresivité**  
 např. množství nerelevantních atributů



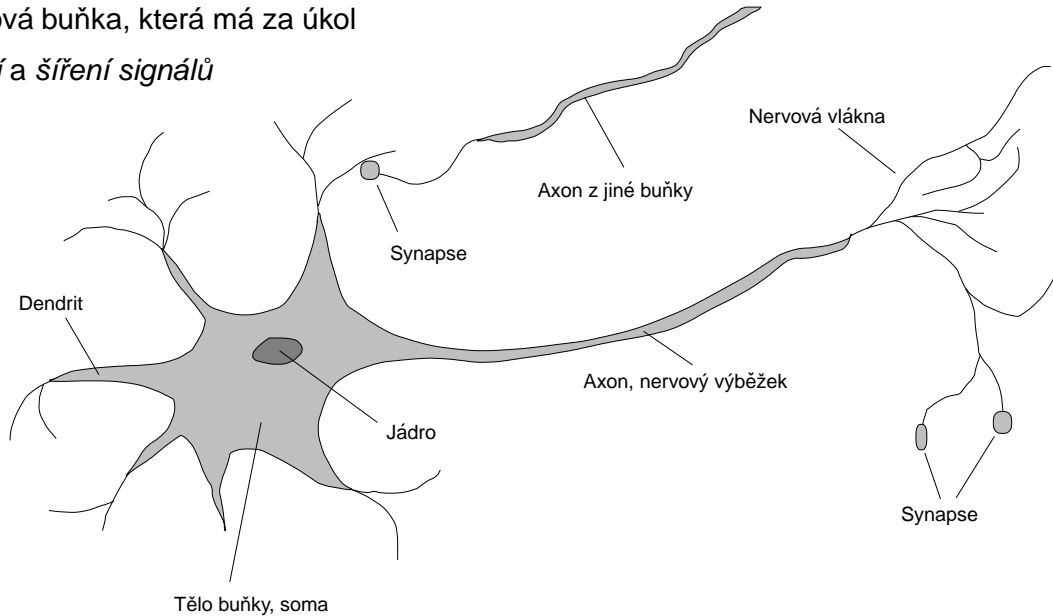
## INDUKTIVNÍ UČENÍ – SHRNUÍ

- 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

## 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

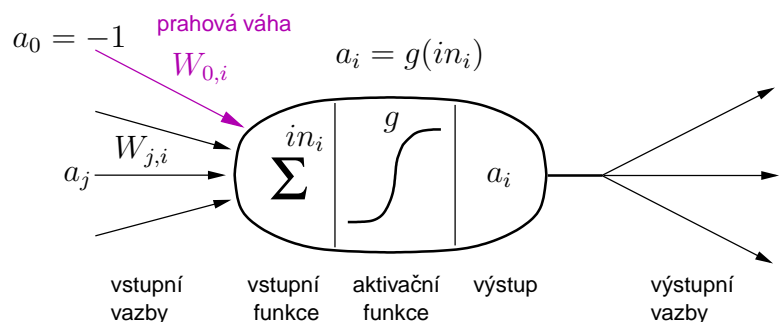
spojené do **neuronové sítě** – mají schopnost **tolerovat šum** ve vstupu a **učit se**

- jednotky (units)** v neuronové síti – jsou propojeny **vzbami (links)**
- 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**  $a_i$

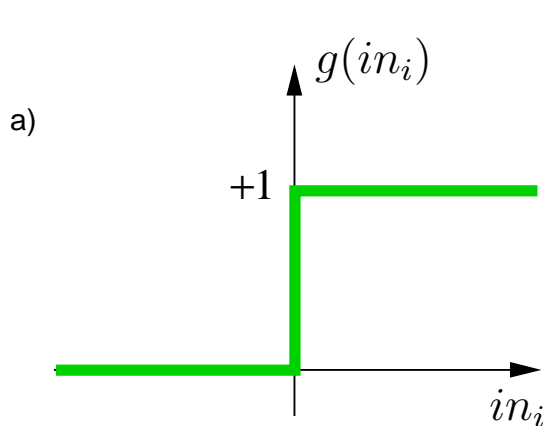
$$a_i = g(in_i) = g\left(\sum_j W_{j,i} a_j\right)$$



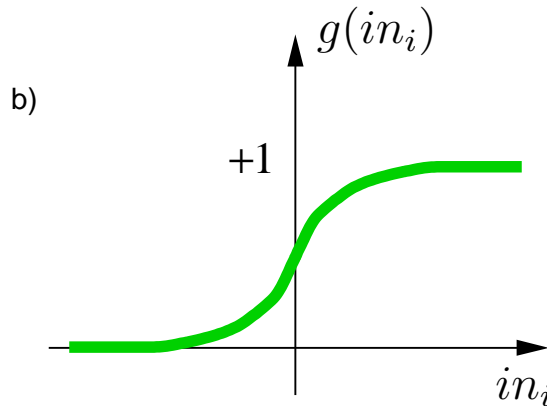
## AKTIVAČNÍ FUNKCE

účel aktivační funkce =  $\left\{ \begin{array}{l} \text{jednotka má být aktivní } (\approx +1) \text{ pro pozitivní příklady, jinak neaktivní } \approx 0 \\ \text{aktivace musí být nelineární, jinak by celá síť byla lineární} \end{array} \right.$

např.



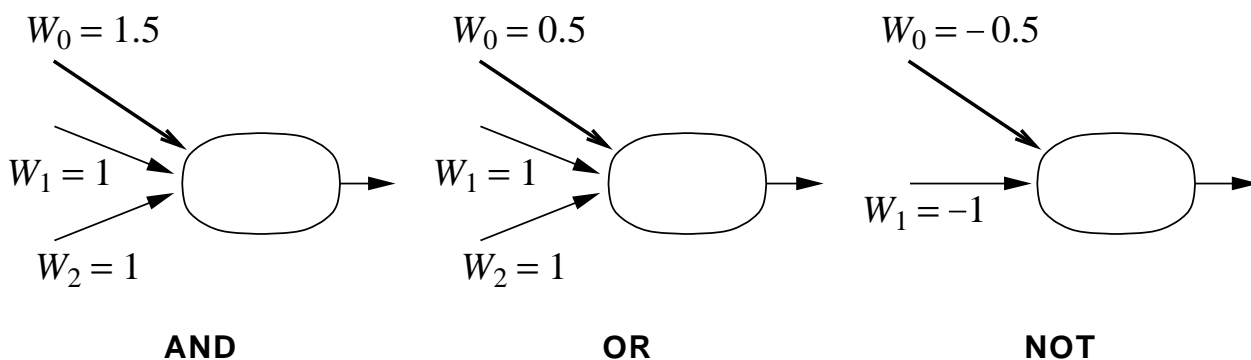
prahová funkce



sigmoida  $1/(1 + e^{-x})$   
je derivovatelná – důležité pro učení

změny prahové váhy  $W_{0,i}$  nastavují nulovou pozicí – 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

### ☐ síť s předním vstupem (*feed-forward networks*)

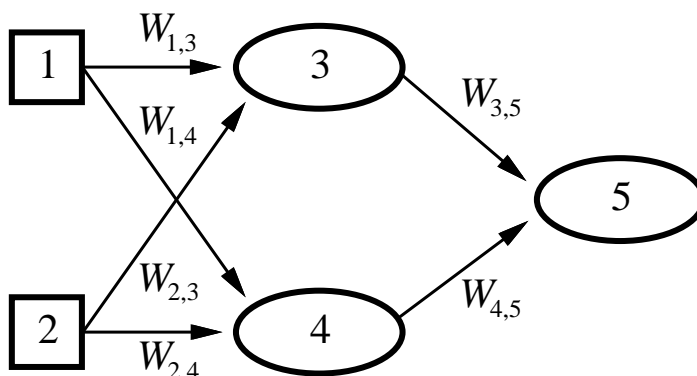
- necyclecké
- implementují funkce
- nemají vnitřní paměť

### ☐ rekurentní síť (*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

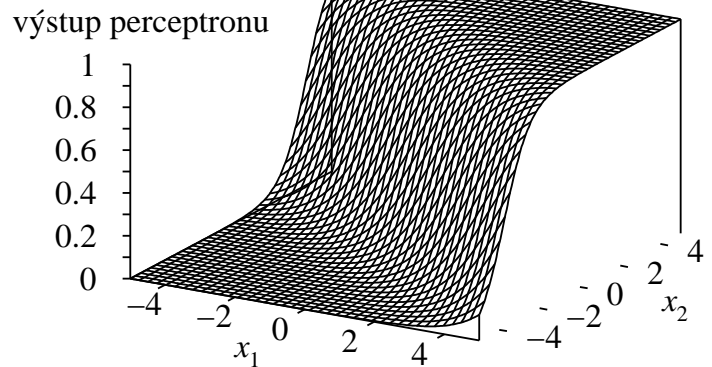
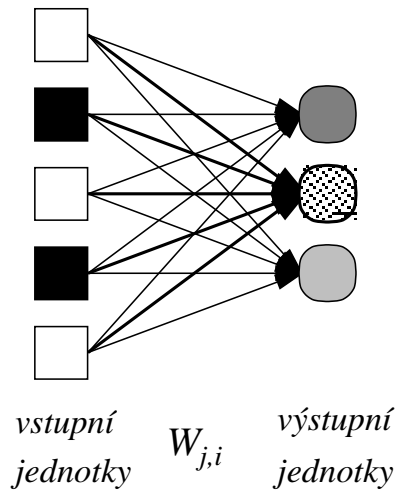


síť s předním vstupem = **parametrizovaná** nelineární funkce vstupu

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

## JEDNOVRSTVÁ SÍŤ – PERCEPTRON

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



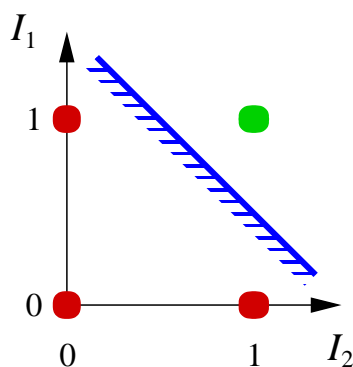
## VYJADŘOVACÍ SÍLA PERCEPTRONU

předpokládejme perceptron s  $g$  zvolenou jako prahová funkce ( $\lfloor \square \rfloor$ )

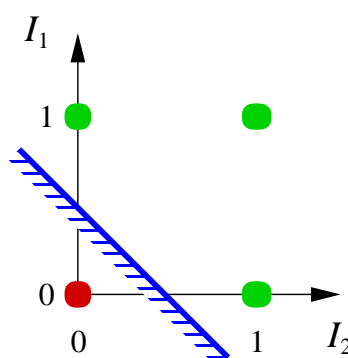
může reprezentovat hodně Booleovských funkcí – AND, OR, NOT, majoritní funkci, ...

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

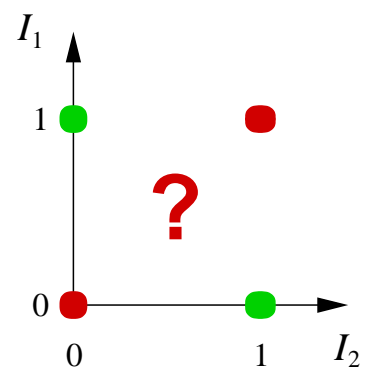
reprezentuje **lineární separátor** (nadrovina) v prostoru vstupu:



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 tak, aby se **snížila chyba** na trénovací sadě

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

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

**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**  $W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j$      $\alpha \dots$  učící konstanta (*learning rate*)

např.  $Err = y - h_{\mathbf{W}}(\mathbf{x}) > 0 \Rightarrow$  výstup  $h_{\mathbf{W}}(\mathbf{x})$  je moc malý

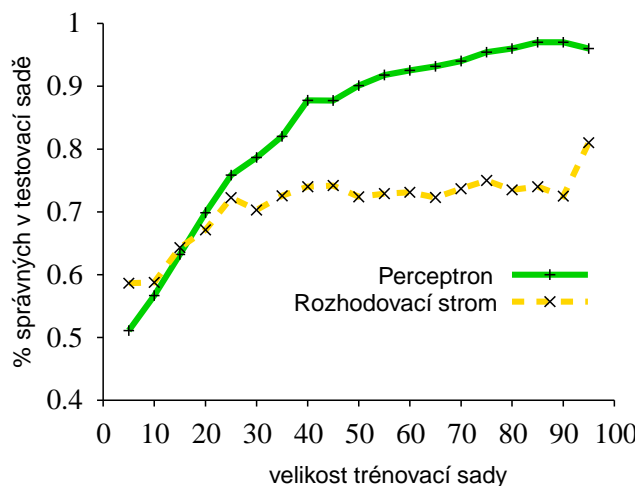
$\Rightarrow$  váhy se musí **zvýšit** pro pozitivní příklady a **snížit** pro negativní

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

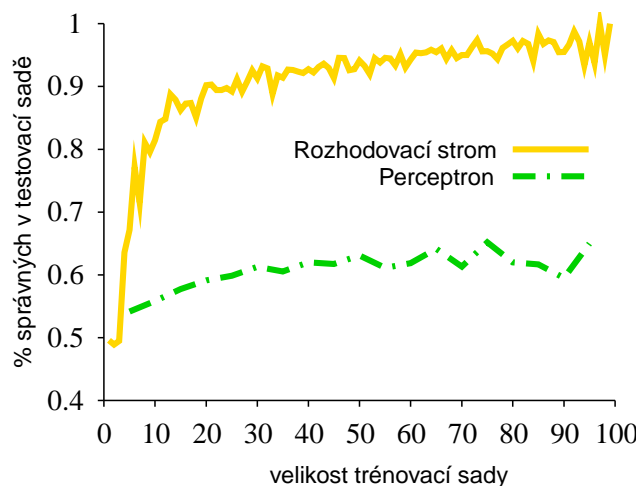
## UČENÍ PERCEPTRONU pokrač.

učící 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

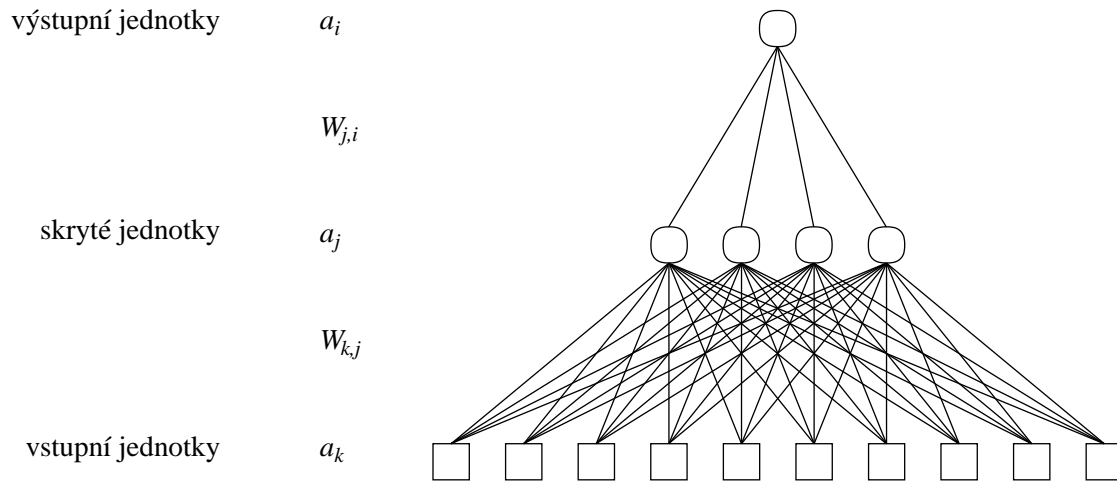




## VÍCEVRSTVÉ NEURONOVÉ SÍTĚ

vrstvy jsou obvykle **úplně propojené**

počet **skrytých jednotek** je obvykle volen experimentálně



## VYJADŘOVACÍ SÍLA VÍCEVRSTVÝCH SÍTÍ

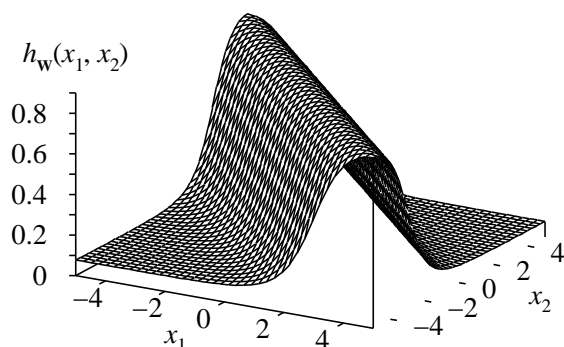
**s jednou skrytou vrstvou** – všechny spojité funkce

**se dvěma skrytými vrstvami** – všechny funkce

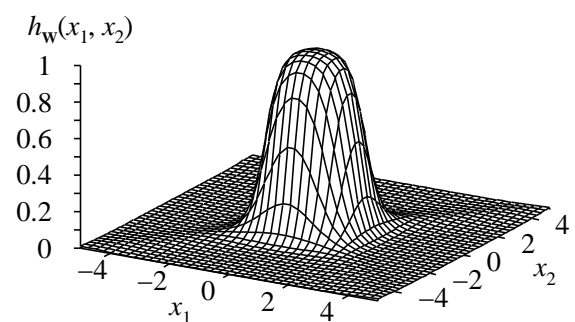
těžko se ovšem pro konkrétní síť zjišťuje její prostor reprezentovatelných funkcí

např.

dvě "opačné" skryté jednotky vytvoří *hřbet*



dva hřbety vytvoří *homoli*



## UČENÍ VÍCEVRSTVÝCH 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

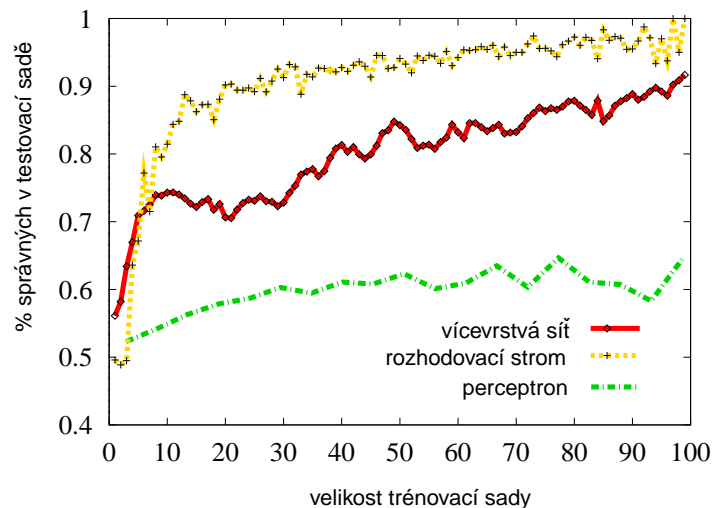
$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j \quad \text{kde} \quad \Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i$$

problémy učení:

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

## UČENÍ VÍCEVRSTVÝCH SÍTÍ pokrač.

vícevrstvá síť se problém 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
  - ...