

Učení, rozhodovací stromy, neuronové sítě

Aleš Horák

E-mail: hales@fi.muni.cz
<http://nlp.fi.muni.cz/uui/>

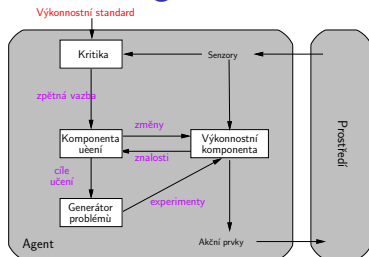
Obsah:

- ▶ Učení
- ▶ Rozhodovací stromy
- ▶ Hodnocení úspěšnosti učícího algoritmu
- ▶ Neuronové sítě
- ▶ PA026 – Projekt z umělé inteligence

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 zaznamená a předá pohoršující reakce dalších řidičů
- ▶ **Komponenta učení** – z hlášení Kritiky vyvodí nové pravidlo, že takové přejíždění 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 atributech:

- 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

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

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**

každý **příklad** je dvojice $x, f(x)$ např.

O	O	×
×	×	×
×	×	×

, +1

úkol **indukce**:

najdi **hypotézu h**

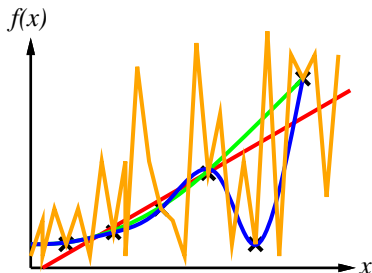
takovou, že **$h \approx f$**

pomocí sady **trénovacích příkladů**

Metoda induktivního učení

zkonstruuj/uprav h , aby souhlasila s f na trénovacích příkladech
 h je konzistentní \Leftrightarrow souhlasí s f na všech příkladech

např. hledání křivky:

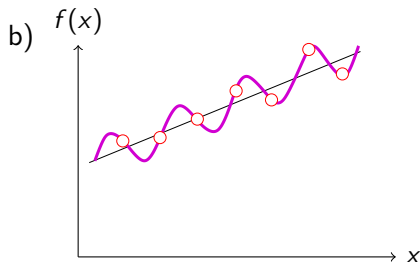
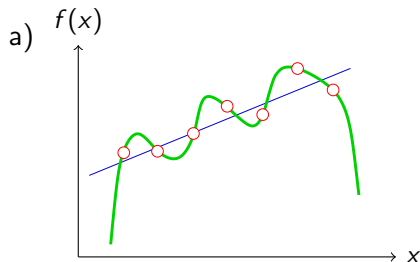


pravidlo **Ockhamovy břitvy** – maximalizovat kombinaci konzistence a jednoduchosti (*nejjednodušší ze správných je nejlepší*)

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



- 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)

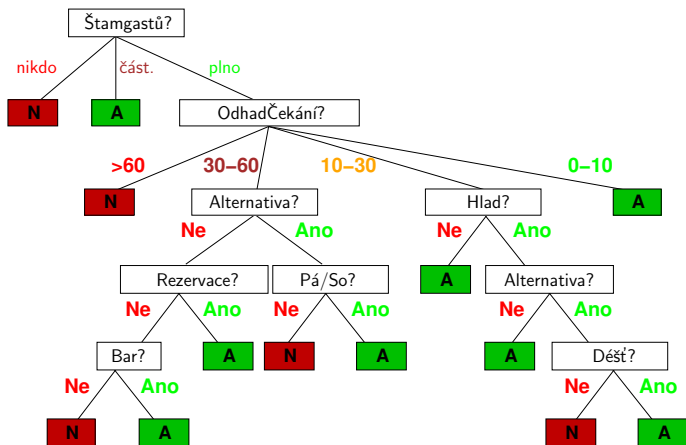
např. rozhodování, zda počkat na uvolnění stolu v restauraci:

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

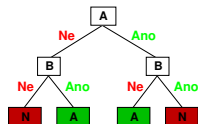
rozhodovací stromy vyjádří libovolnou **Booleovskou funkci vstupních atributů** → odpovídá **výrokové logice**

$\forall s$ počkat?(s) $\Leftrightarrow (P_1(s) \vee P_2(s) \vee \dots \vee P_n(s))$,

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

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 ($Hlad \wedge \neg D\acute{e}št'$)
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

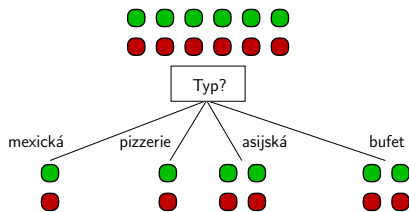
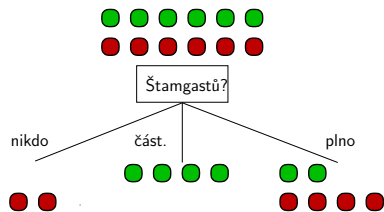
- pro každý příklad v trénovací sadě přidej jednu cestu od kořene k listu
- na stejných příkladech jako v trénovací sadě bude fungovat přesně
- na nových příkladech se bude chovat náhodně – **negeneralizuje** vzory z příkladů, pouze **kopíruje** pozorování

▶ heuristická konstrukce kompaktního stromu

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

Výběr atributu

dobry atribut \equiv rozdělí příklady na podmnožiny, které jsou (nejlépe) “všechny pozitivní” nebo “všechny negativní”



Štamgastů? je lepší volba atributu \leftarrow 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

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

n možných odpovědí $\langle P(v_1), \dots, P(v_n) \rangle$ → **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: $I(\langle \frac{1}{2}, \frac{1}{2} \rangle) = -\frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{2} \log_2 \frac{1}{2} = \frac{1}{2} + \frac{1}{2} = 1$ 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

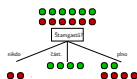
např. pro X_1, \dots, X_{12} z volby čekání na stůl je $p = n = 6$, takže potřebujeme 1 bit

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

= rozdíl odhadu odpovědi před a po testu atributu

Použití míry informace pro výběr atributu

atribut A rozdělí sadu příkladů E na podmnožiny E_i
(nejlépe tak, že každá potřebuje méně informace)



necht' E_i má p_i pozitivních a n_i negativních příkladů

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

⇒ očekávaný počet bitů celkem je $Remainder(A) = \sum_i \frac{p_i+n_i}{p+n} \cdot I(\langle \frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i} \rangle)$

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

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

$Gain(\text{Štamgastů?}) \approx 0.541$ bitů

$Gain(\text{Typ?}) = 0$ bitů

obecně: E_i (pro $A = v_i$) obsahuje $c_{i,k}$ klasifikací do tříd c_1, \dots, c_k

⇒ $Remainder(A) = \sum_i P(v_i) \cdot I(\langle P(c_{i,1}), \dots, P(c_{i,k}) \rangle)$

⇒ $Gain(A) = I(\langle P(v_1), \dots, P(v_n) \rangle) - Remainder(A)$

Algoritmus IDT – učení formou rozhodovacích stromů

```

% induce_tree( +Attributes, +Examples, -Tree)
induce_tree( _, [], null) :- !.
induce_tree( _, [example( Class, _ ) | Examples], leaf( Class)) :- %  $\forall$  příklady stejné klasifikace
    \+ (member( example( ClassX, _), Examples), ClassX \== Class), !.
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, distribuce 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( _, [], _, _, [] ). % žádné atributy, žádné podstromy
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 Coni) :- \+ (member( Att = Val Coni) member( Att = ValX Object) ValX \== Val)

```

Algoritmus IDT – učení formou rozhodovacích stromů

% choose_attribute(+Atts, +Examples, –BestAtt/BestGain) – výběr nejlepšího atributu

`choose_attribute([], -, 0/0).`

`choose_attribute([Att], Examples, Att/Gain):- !, gain(Examples, Att, Gain).`

`choose_attribute([Att|Atts], Examples, BestAtt/BestGain):-`

`choose_attribute(Atts, Examples, BestAtt1/BestGain1),`

`gain(Examples, Att, Gain),`

`(Gain>BestGain1, !, BestAtt=Att, BestGain=Gain ;`

`BestAtt=BestAtt1, BestGain=BestGain1).`

% gain(+Examples, +Attribute, –Gain) – zisk atributu

`gain(Exs, Att ,Gain) :- attribute(Att ,AttVals), length(Exs, Total),`

`setof(Class, X^example(Class,X), Classes), % množina všech Class`

`findall(Nc, (member(C,Classes), cntclass(C,Exs,Nc)), CCnts),`

`info(CCnts,Total,I), rem(Att, AttVals,Exs,Classes,Total,Rem),`

`Gain is I-Rem.`

% info(+ValueCounts, +Total, –I)

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

`info([], -, 0).`

`info([VC|ValueCounts], Total, I) :- info(ValueCounts,Total,I1),`

`(VC = 0, !, I is I1 ;`

`Pvi is VC / Total, log2(Pvi, LogPvi), I is - Pvi * LogPvi + I1).`

Algoritmus IDT – učení formou rozhodovacích stromů

```

% rem( +Att, +AttVals, +Exs, +Classes, +Total, -Rem)
% "zbytková informace" po testu na Att:  $Remainder(A) = \sum_i P(v_i) \cdot I(\langle P(c_{i,1}), \dots, P(c_{i,k}) \rangle)$ 
rem( -, [], -, -, -, 0).
rem( Att, [V | Vs], Exs, Classes, Total, Rem) :-
    findall(1, (member(example(-, AVs), Exs), member(Att = V, AVs)), L1),
    length(L1, Nv), %  $Nv = p_i + n_i$ 
    findall(Ni, (member(C, Classes), cntclassattv(Att, V, C, Exs, Ni)), VCnts),
    Pv is Nv / Total, %  $P(v)$ 
    info(VCnts, Nv, I), rem(Att, Vs, Exs, Classes, Total, Rem1),
    Rem is Pv * I + Rem1.

% cntclass( +Class, +Exs, -Cnt) – počet příkladů třídy Class
cntclass( Class, Exs, Cnt) :-
    findall(1, member(example(Class, _), Exs), L), length(L, Cnt).

% cntclass( +Att, +Val, +Class, +Exs, -Cnt)
% počet příkladů třídy Class pro hodnotu Val atributu Att
cntclassattv( Att, Val, Class, Exs, Cnt) :-
    findall(1, (member(example(Class, AVs), Exs), member(Att = Val, AVs)), L),
    length(L, Cnt).

```

```
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
```

```
            = $$$
```

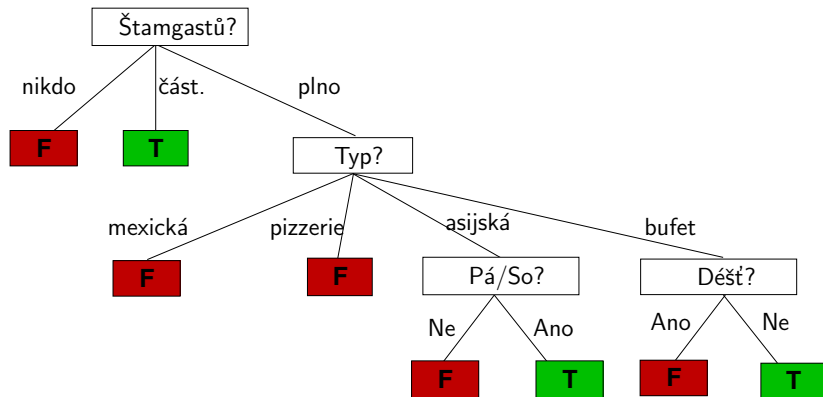
```
            necekat
```

```
  = ne
```

```
    necekat
```

IDT – výsledný rozhodovací strom

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



podstatně jednodušší než strom “z tabulky příkladů”

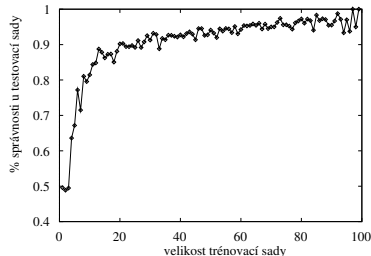
Hodnocení úspěšnosti učícího algoritmu

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

používaná **metodologie** (cross validation):

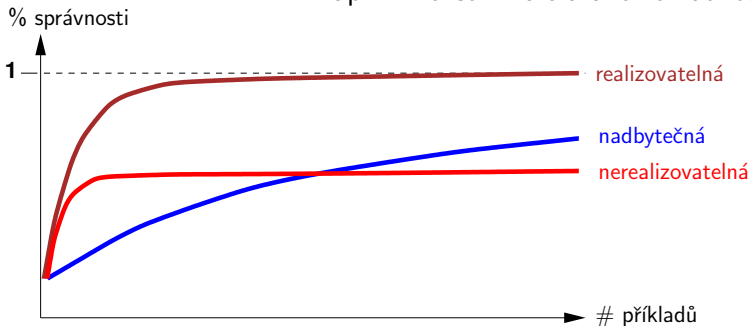
1. vezmeme velkou množinu příkladů
2. rozdělíme ji na 2 množiny – **trénovací** a **testovací**
3. aplikujeme učící algoritmus na **trénovací** sadu, získáme hypotézu h
4. **změříme** procento příkladů v **testovací** sadě, které jsou správně klasifikované hypotézou h
5. opakujeme kroky 2–4 pro různé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

křivka učení – závislost velikosti trénovací sady na úspěšnosti



Hodnocení úspěšnosti učícího algoritmu – pokrač.

- tvar 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



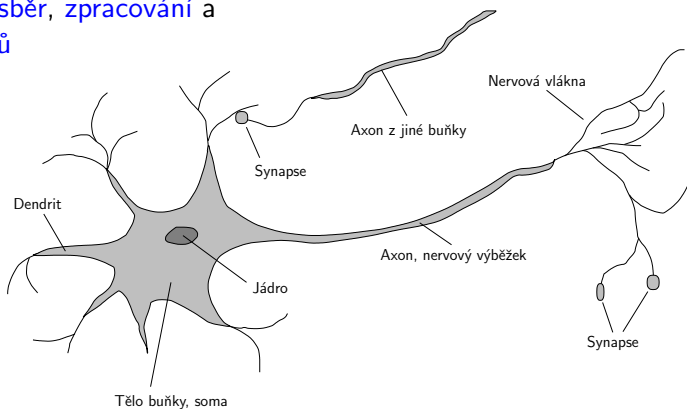
Induktivní učení – shrnutí

- ▶ **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

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*)

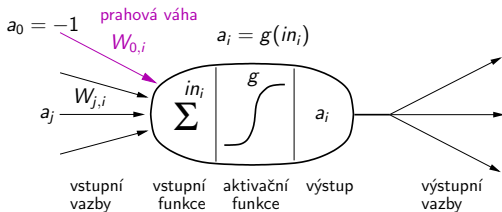
– vazba z jednotky j do i propaguje **aktivaci** a_j jednotky j

– každá vazba má číselnou **váhu** $W_{j,i}$ (síla+znaménko)

funkce jednotky i :

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

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

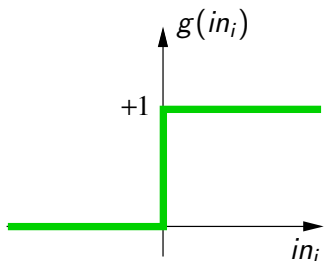


Aktivační funkce

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

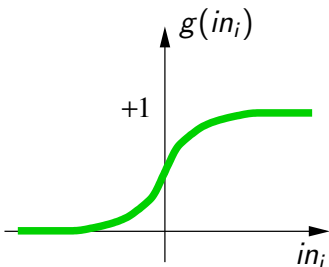
např.

a)



prahová funkce

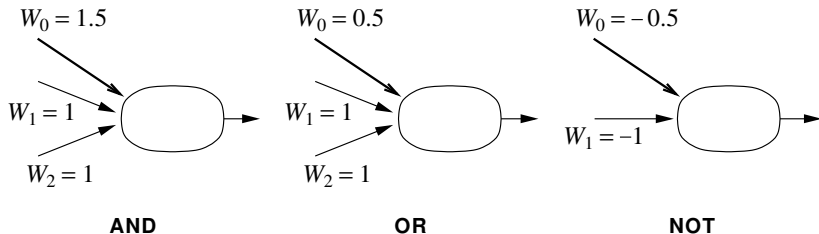
b)



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

▶ síť s předním vstupem (*feed-forward networks*)

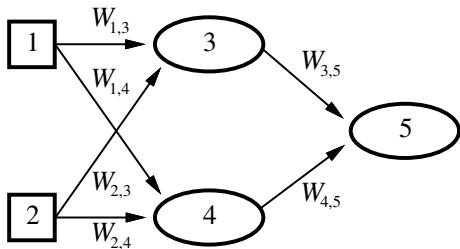
- necyklické
- implementují funkce
- nemají vnitřní paměť

▶ 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



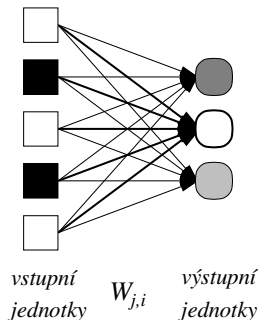
síť s předním vstupem = parametrizovaná nelineární funkce vstupu

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

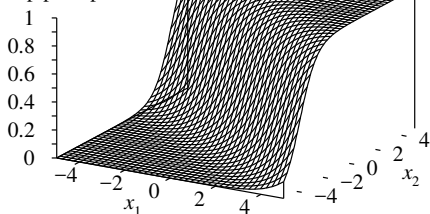
Jednovrstvá síť – perceptron

perceptron

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



výstup perceptronu

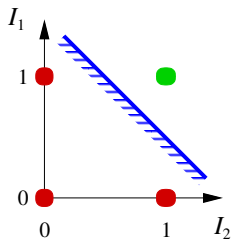


Vyjadřovací síla perceptronu

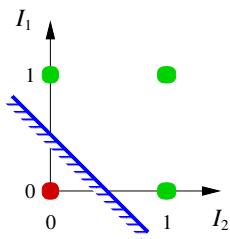
perceptron může reprezentovat hodně Booleovských funkcí – AND, OR, NOT, majoritní funkci, ...

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

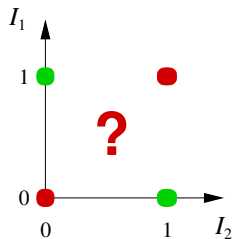
reprezentuje **lineární separátor** (nadrovina) v prostoru vstupu:



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

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

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

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

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

pravidlo pro úpravu váhy

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

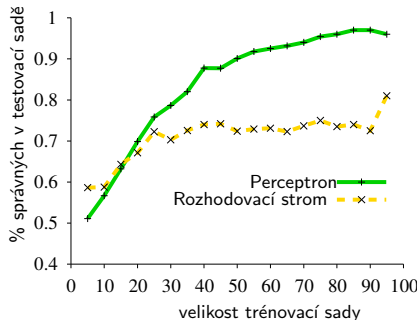
např. $\text{Err} = y - h_{\mathbf{W}}(\mathbf{x}) > 0 \Rightarrow$ výstup $h_{\mathbf{W}}(\mathbf{x})$ je moc malý
 \Rightarrow váhy se musí **zvýšit** pro pozitivní příklady a **snížit** pro negativní

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

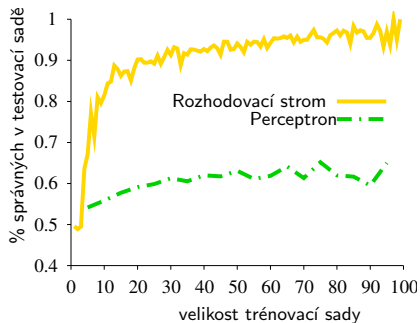
Učení perceptronu pokrač.

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

$W_{j,i}$

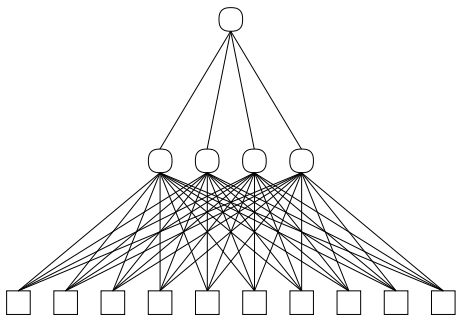
skryté jednotky

a_j

$W_{k,j}$

vstupní jednotky

a_k



Vyjadřovací síla vícevrstevných sítí

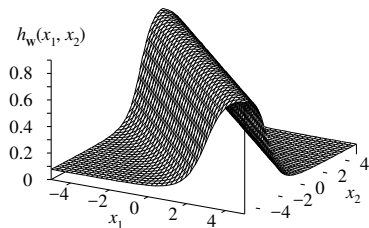
s jednou skrytou vrstvou – všechny **spojité funkce**

se dvěma skrytými vrstvami – **všechny funkce**

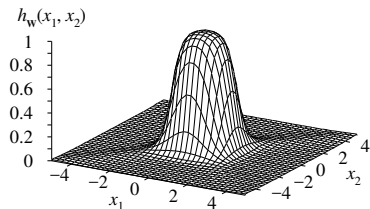
těžko se ovšem pro **konkrétní síť** zjišťuje její prostor **reprezentovatelných funkcí**

např.

dvě “opačné” skryté jednotky
vytvoří *hřbet*



dva hřbety vytvoří *homoli*



Učení vícevrstvých sítí

pravidla pro úpravu vah:

- ▶ výstupní vrstva – stejně jako u perceptronu

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

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

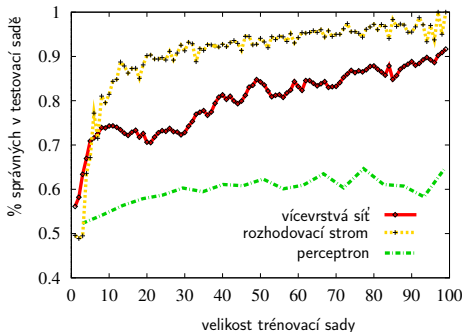
$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j \quad \text{kde} \quad \Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i$$

problémy učení:

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

Učení vícevrstevných 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
 - ...

PA026 – Projekt z umělé inteligence

- ▶ navazuje na předmět *PB016 Úvod do umělé inteligence*
- ▶ volba programovacího jazyka ovšem není nijak omezena
- ▶ samostatná volba tématu v rozsahu ≥ 1 semestru
- ▶ předmět probíhá jako konzultace
- ▶ zajímavé výsledky (<http://nlp.fi.muni.cz/uiprojekt/>)
 - projekt [elnet](#) – > 5 let spolupráce na grantových projektech simulace elektrorozvodných sítí
 - projekt [plagiaty_z_webu](#) – reálné a funkční vyhledávání shod s dokumenty na celém webu
 - projekt [robot_johnny_5](#) – sestavení a “oživení” robota – mobilního počítače

