Učení, rozhodovací stromy, neuronové sítě

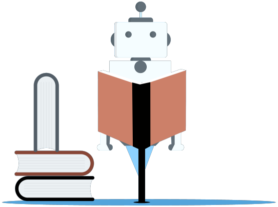
Aleš Horák

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

Obsah:

- Učení
- Rozhodovací stromy
- Neuronové sítě

Strojové učení



?

$\rightarrow$ sli.do/uui

Induktivní učení

známé taky jako věda ☺

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

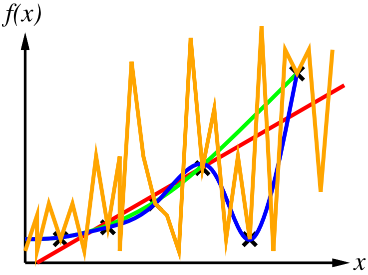
každý příklad je dvojice x, $f(x)$ např. $\begin{array}{c|c|c} \mathrm{O} & \mathrm{O} & \times \\ \hline & \times & \\ \hline \times & & \end{array} \; , \quad +1$

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:

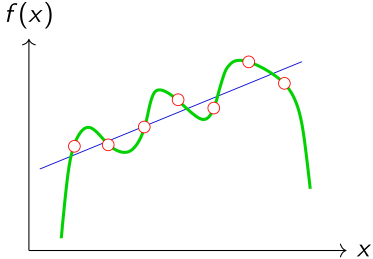


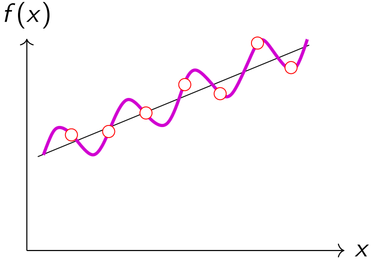
pravidlo Ockhamovy břitvy – maximalizovat kombinaci konzistence a jednoduchosti (*nejjednodušší ze správných je nejlepší*)

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

a)


b)


- 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 $ax + by + c \sin x$

Hodnocení úspěšnosti učícího algoritmu

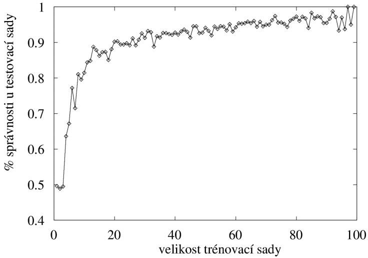
jak můžeme zjistit, zda $h \approx f$?

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

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 úspěšnosti na velikosti trénovací sady



Hodnocení úspěšnosti učícího algoritmu – pokrač.

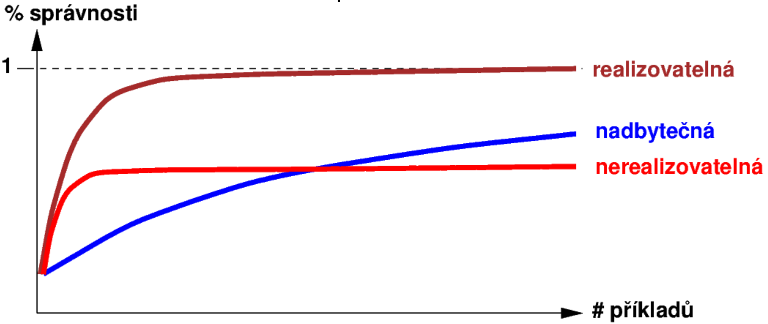
tvar křivky učení závisí na

- je hledaná funkce

  realizovatelná × nerealizovatelná

  funkce může být nerealizovatelná kvůli
  - chybějícím atributům
  - omezenému prostoru hypotéz
- naopak nadbytečné expresivitě

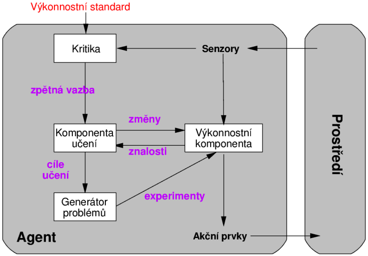
  např. množství nerelevantních atributů



Učící se agent

- **učení** agenta – využití jeho vjemů z prostředí nejen pro vyvození další akce
- učení modifikuje rozhodovací systém agenta pro zlepšení jeho výkonnosti
- 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

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	**ohodnocovací funkce**	vážená lineární funkce	výhra/prohra
Logický agent	**určení akce**	axiomy *Result*	výsledné skóre
Reflexní agent	**váhy perceptronu**	neuronová síť	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
- zpětnovazební (*reinforcement learning*) – agent se učí podle odměn/pokut

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
- kvalita učení – přesnost odhadu změřená na testovací sadě

Atributová reprezentace příkladů

příklady popsané výčtem hodnot atributů (libovolných hodnot)

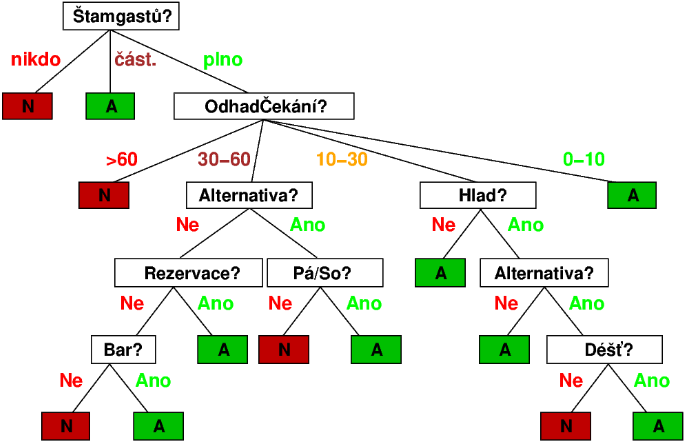
např. rozhodování, zda počkat na uvolnění stolu v restauraci:

Příklad	Atributy										počkat?
	Alt	*Bar*	*Pá/So*	*Hlad*	*Štam*	*Cen*	*Déšť*	*Rez*	*Typ*	*ČekD*	
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:



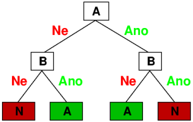
Vyjadřovací síla rozhodovacích stromů

rozhodovací stromy vyjádří libovolnou Booleovskou funkci vstupních atributů $\rightarrow$ odpovídá výrokové logice

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

pro libovolnou Booleovskou funkci $\rightarrow$ řádek v pravdivostní tabulce $=$ cesta ve stromu (od kořene k listu)

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

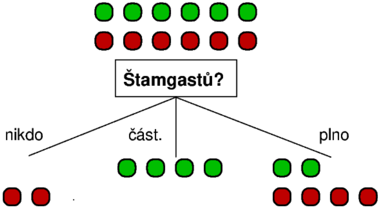
– zvyšuje šance, že najdeme přesné vyjádření cílové funkce
– ALE zvyšuje i počet možných hypotéz, které jsou konzistentní s trénovací množinou
  $\Rightarrow$ můžeme získat nižší kvalitu předpovědí (generalizace)

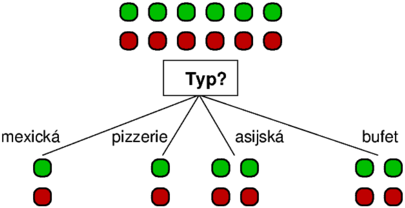
Učení formou rozhodovacích stromů

- **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
    $\rightarrow$ heuristikou najdeme alespoň dostatečně malý
  - hlavní myšlenka – vybíráme atributy pro test v co **nejlepším pořadí**
  - algoritmus IDT, Induction of Decision Trees

Výběr atributu

dobrý 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 $\rightarrow$ tím více informace je v ní obsaženo

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

n možných odpovědí $\langle P(v_1), \ldots, P(v_n)\rangle$ $\rightarrow$ míra informace v odpovědi obsažená

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

tato míra se také nazývá entropie

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

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

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

Použití míry informace pro výběr atributu

předpokládejme, že máme p pozitivních a n negativních příkladů

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

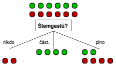
např. pro $X_1, \ldots, X_{12}$ z volby čekání na stůl je $p = n = 6$, takže potřebujeme **1 bit**

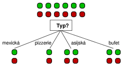
výběr atributu – *kolik informace nám dá test na hodnotu atributu A?*

$=$ 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)





nechť E_i má p_i pozitivních a n_i negativních příkladů

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

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

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

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

$Gain(\check{S}tamgast\mathring{u}?) \approx 0.541$ bitů $\qquad Gain(Typ?) = 0$ bitů

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

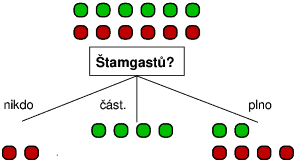
$\Rightarrow Remainder(A) = \sum_i P(v_i) \cdot I\left(\langle P(c_{i,1}), \ldots, P(c_{i,k}) \rangle\right)$

$\Rightarrow Gain(A) = I\left(\langle P(v_1), \ldots, P(v_n) \rangle\right) - Remainder(A)$

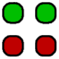
Algoritmus IDT – učení formou rozhodovacích stromů

rekurzivní **tvorba** rozhodovacího stromu (vstup = atributy+příklady):

1. prázný strom – pokud nejsou další příklady
2. listový uzel – pokud jsou příklady stejného typu 
3. uzel s testem na atribut – pokud existuje nejlepší atribut podle *Gain*()



každý podstrom pro podmnožinu příkladů od kroku 1.

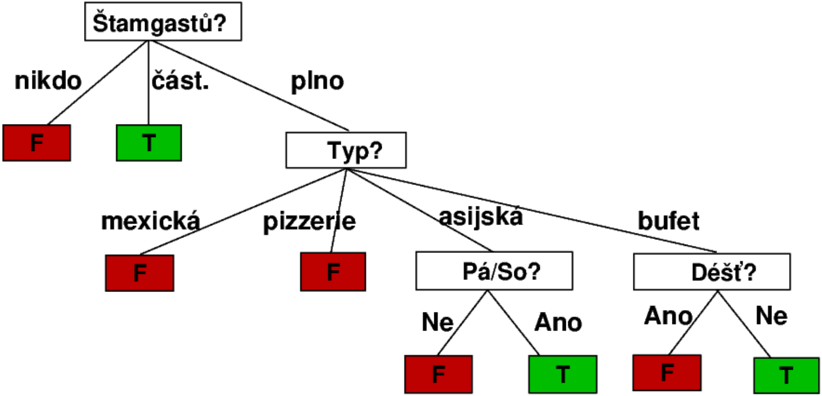
4. listový uzel s distribucí typů – jinak 

Algoritmus IDT – příklad

```
attributes = { "hlad": ["ano", "ne"],
               "štam": ["nikdo", "část", "plno"],
               "cen": ["$", "$$", "$$$"], ... }
examples = [
   ("počkat", [
      ("alt", "ano"), ("bar", "ne"), ("páso", "ne"), ("hlad", "ano"), ("štam", "část"),
      ("cen", "$$$"), ("déšť", "ne"), ("rez", "ano"), ("typ", "mexická") ]),
   ("nečekat", [
      ("alt", "ano"), ("bar", "ne"), ("páso", "ne"), ("hlad", "ano"), ("štam", "plno"),
      ("cen", "$"), ("déšť", "ne"), ("rez", "ne"), ("typ", "asijská") ]), ... ]
PrintTree(InduceTree( attributes , examples))
štam?
  = nikdo
    nečekat
  = část
    počkat
  = plno
    hlad?
     = ano
       cen?
        = $
          páso?
           = ano
             počkat
           = ne
             nečekat
        = $$$
          nečekat
     = ne
       nečekat
```

IDT – výsledný rozhodovací strom

rozhodovací strom naučený z 12-ti příkladů:

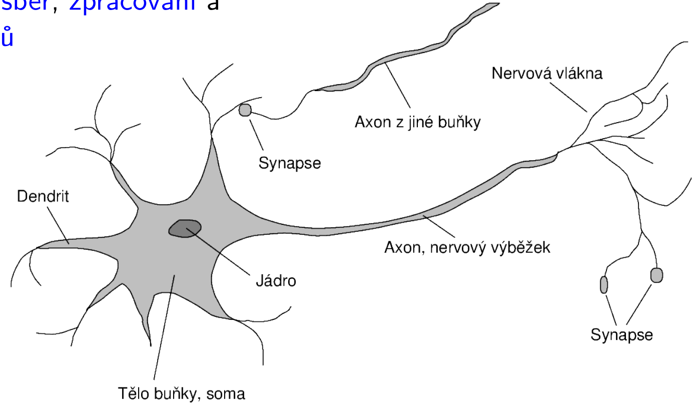


podstatně jednodušší než strom "z tabulky příkladů"

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 neuronu

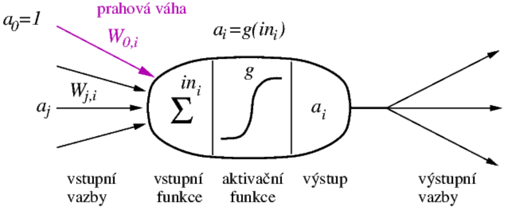
1943 – McCulloch & Pitts – matematický model neuronu
spojené do **neuronové sítě** – schopnost tolerovat šum ve vstupu a učit se

jednotky v neuronové síti (*units*)
- jsou propojeny vazbami (*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

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



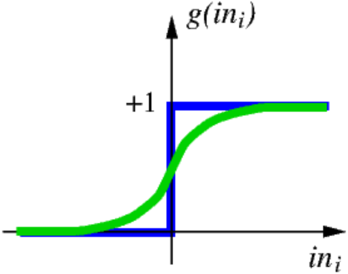
Aktivační funkce

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

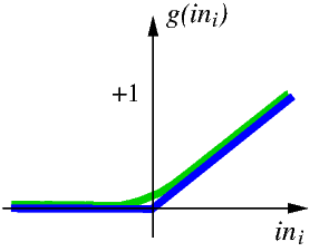
např.

a)



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

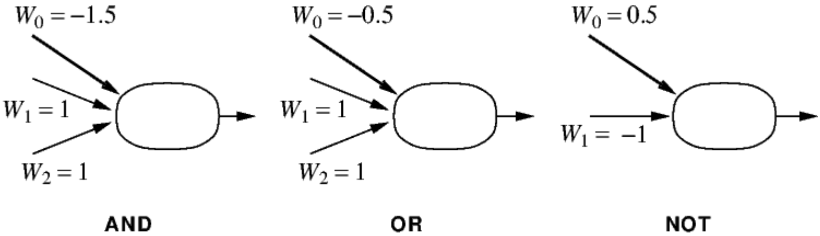
b)



ReLU (*rectified linear unit*)
softplus $log(1+e^{x})$

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

Logické funkce pomocí neuronové jednotky



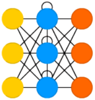
jednotka McCulloch & Pitts sama umí implementovat základní Booleovské funkce

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

Struktury neuronových sítí

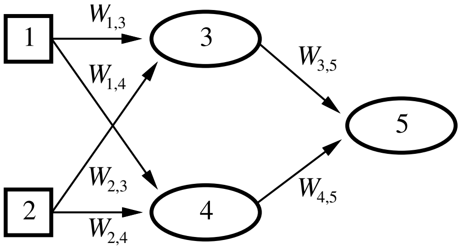
- sítě s předním vstupem (*feed-forward networks*)
  - necyklické
  - implementují funkce
  - nemají vnitřní paměť
- rekurentní sítě (*recurrent networks*)
  - cyklické, vlastní výstup si berou opět na vstup
  - složitější a schopnější
  - výstup má (zpožděný) vliv na aktivaci = paměť
  - Hopfieldovy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

www.asimovinstitute.org/neural-network-zoo



Příklad sítě s předním vstupem

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

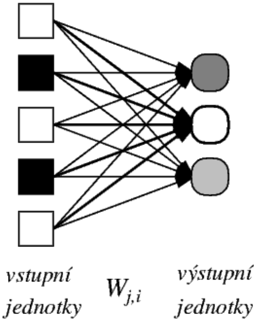


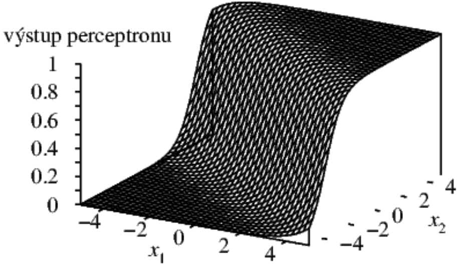
síť s předním vstupem = parametrizovaná nelineární funkce vstupu

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

Jednovrstvá síť – perceptron

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

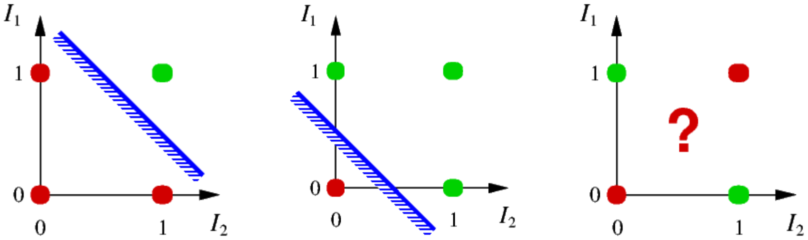




Vyjadřovací síla perceptronu

perceptron může reprezentovat hodně Booleovských funkcí – AND, OR, NOT, majoritní funkci ($\sum_j W_j x_j > n/2, W_j = 1$), ...

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

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

váhy pro minimální chybu pak hledáme optimalizačním prohledáváním spojitého prostoru vah

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

pravidlo pro úpravu váhy

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

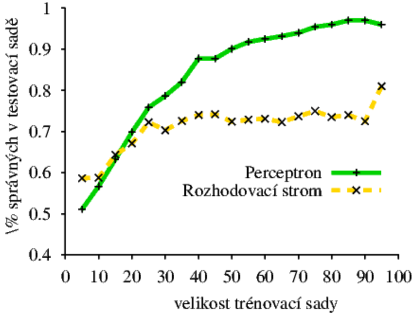
např. $Err = y - h_{\mathbf{W}}(\mathbf{x}) > 0$ $\Rightarrow$ výstup $h_{\mathbf{W}}(\mathbf{x})$ je moc malý
$\Rightarrow$ váhy se musí *zvýšit* pro pozitivní příklady a *snížit* pro negativní

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

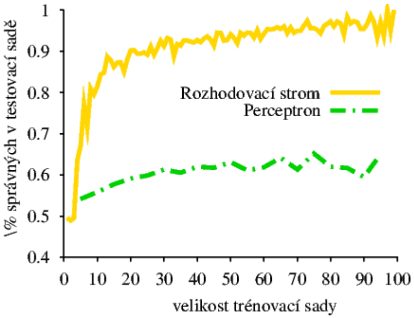
Učení perceptronu pokrač.

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

a) učení majoritní funkce



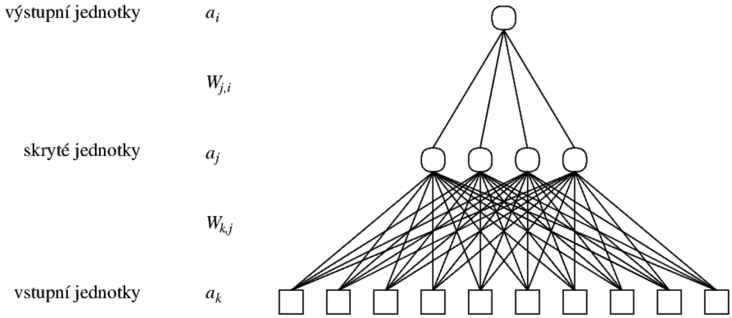
b) učení čekání na volný stůl v restauraci



Vícevrstvé neuronové sítě

označení MLP, multi-layer perceptron

vrstvy jsou obvykle úplně propojené
počet **skrytých jednotek** je obvykle volen experimentálně

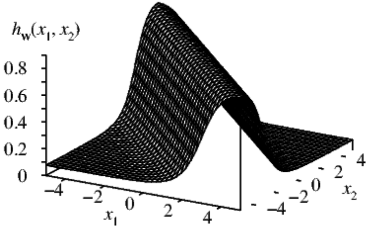


Vyjadřovací síla vícevrstvých sítí

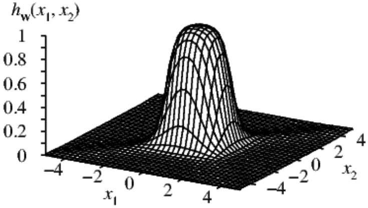
s jednou skrytou vrstvou – všechny spojité funkce
se dvěma skrytými vrstvami – všechny funkce
těžko se ovšem pro konkrétní síť zjišťuje její prostor reprezentovatelných funkcí

např.

dvě "opačné" skryté jednotky vytvoří *hřbet*



dva hřbety vytvoří *homoli*



playground.tensorflow.org

Učení vícevrstvých sítí

pravidla pro úpravu vah:

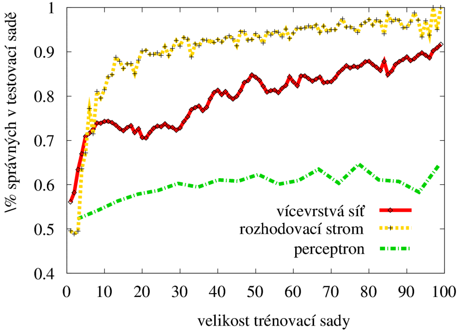
- výstupní vrstva – stejně jako u perceptronu
  $$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i \qquad kde \quad \Delta_i = Err_i \times \mathbf{g}'(in_i)$$
- skryté vrstvy – **zpětné šíření** (*back-propagation*) chyby z výstupní vrstvy
  $$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j \qquad kde \quad \Delta_j = \mathbf{g}'(in_j) \sum_i W_{j,i} \Delta_i$$

problémy učení:

- dosažení lokálního minima chyby
- příliš pomalá konvergence
- přílišné upnutí na příklady $\rightarrow$ 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
  - rozpoznávání ručně psaného písma
  - řízení auta, ...
- v posledních letech **hluboké neuronové sítě** – lépe generalizují