

Učení, rozhodovací stromy, neuronové sítě

Aleš Horák

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

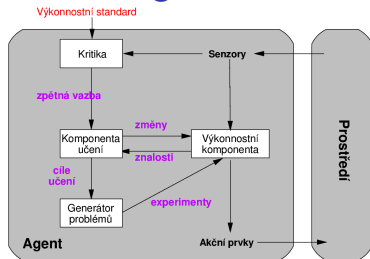
Obsah:

- ▶ Učení
- ▶ Rozhodovací stromy
- ▶ Hodnocení úspěšnosti učícího algoritmu
- ▶ Neuronové sítě

Učení

- ▶ **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 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

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
- ▶ **zpětnovazební** (*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

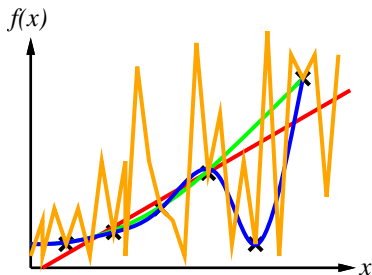
takovou, že $h \approx f$

pomocí sady **trénovacích příkladů**

Metoda induktivního učení

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

např. hledání křivky:

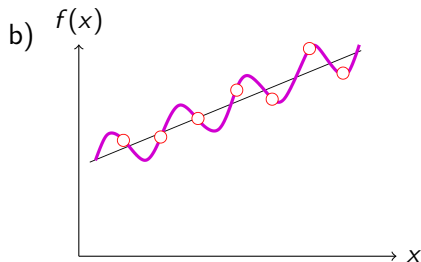
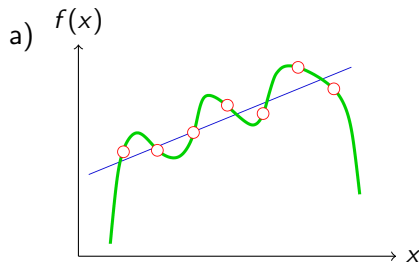


pravidlo **Ockhamovy břitvy** – maximalizovat kombinaci konzistence a jednoduchosti (*nejjednodušší ze správných je nejlepší*)

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)

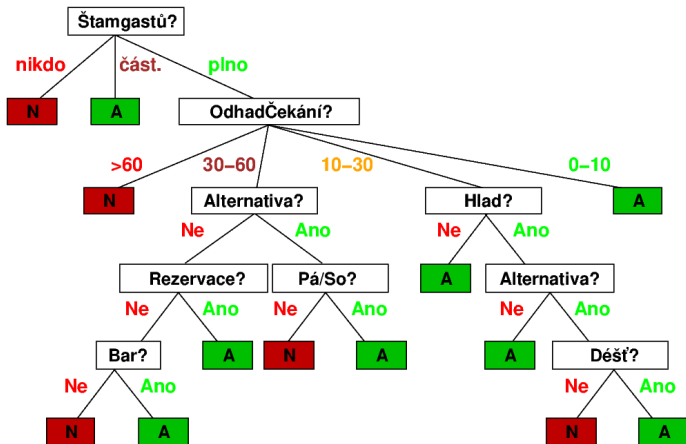
např. rozhodování, zda **počkat na uvolnění stolu v restauraci**:

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

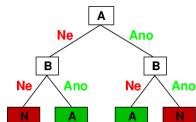
rozhodovací stromy vyjádří libovolnou **Booleovskou funkci vstupních atributů** → odpovídá **výrokové logice**

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

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
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 ($Hlad \wedge \neg D\acute{e}št'$)
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

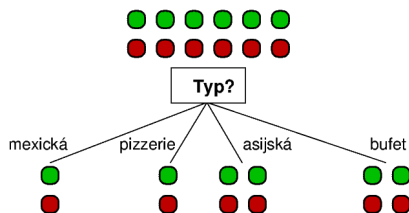
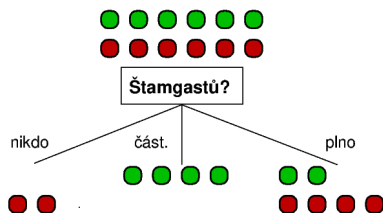
- pro každý příklad v trénovací sadě přidej jednu cestu od kořene k listu
- na stejných příkladech jako v trénovací sadě bude fungovat přesně
- na nových příkladech se bude chovat náhodně – **negeneralizuje** vzory z příkladů, pouze **kopíruje** pozorování

▶ 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 $\langle P(v_1), \dots, P(v_n) \rangle \rightarrow$ 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: $I(\langle \frac{1}{2}, \frac{1}{2} \rangle) = -\frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{2} \log_2 \frac{1}{2} = \frac{1}{2} + \frac{1}{2} = 1$ 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

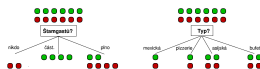
např. pro X_1, \dots, X_{12} z volby čekání na stůl je $p = n = 6$, takže potřebujeme 1 bit

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 – 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

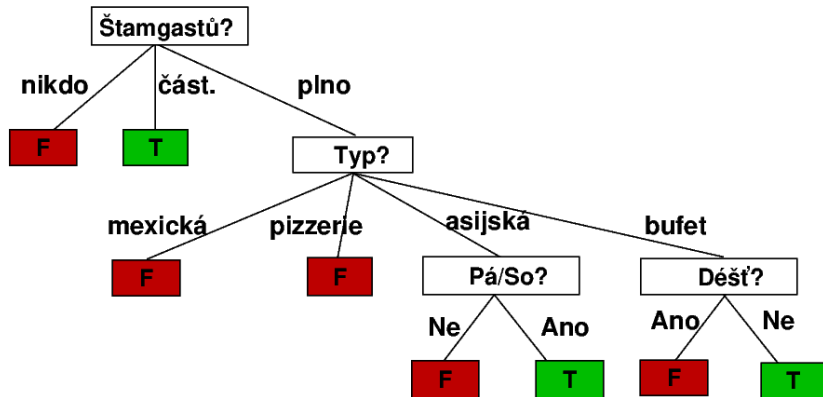
```

Algoritmus IDT – učení formou rozhodovacích stromů

```
function INDUCETREE( attributes , examples )
  if length(examples) = 0 then return None
  single_class  $\leftarrow$  all_same_class(examples) #  $\forall$  příklady stejné třídy
  if single_class is not None then return leaf_tree(single_class)
  attribute  $\leftarrow$  choose_attribute(attributes, examples) # podle míry informace
  if attribute is None then # žádný užitečný atribut, list s distribucí klasifikací
    return leaf_tree(get_example_classes(examples))
  tree  $\leftarrow$  new_decision_tree(attribute) # nový (pod)strom s testem na atribut
  foreach value  $\in$  get_attribute_values(attribute) do
    val_examples  $\leftarrow$  [ e for e  $\in$  examples if attr_val(e, attribute) = value ]
    subtree  $\leftarrow$  InduceTree(attributes - attribute , val_examples)
    if subtree is None then
      subtree  $\leftarrow$  leaf_tree(get_example_classes(val_examples))
    add_tree_branch(value, subtree)
  return tree
```

IDT – výsledný rozhodovací strom

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



podstatně jednodušší než strom “z tabulky příkladů”

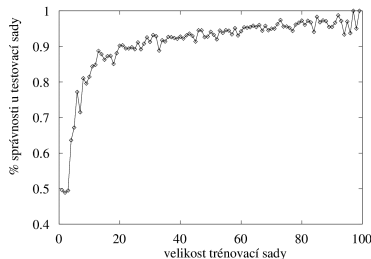
Hodnocení úspěšnosti učícího algoritmu

jak můžeme zjistit, zda $h \approx f$? $\left\{ \begin{array}{l} \text{dopředu} - \text{použit v\u011bt\u016f Teorie kompu-} \\ \text{ta\u010dn\u00edho u\u010den\u00ed} \\ \text{po nau\u010den\u00ed} - \text{kontrolou na jin\u00e9 tr\u00e9novac\u00ed} \\ \text{sad\u011b} \end{array} \right.$

pou\u017e\u00edvan\u00e1 **metodologie** (cross validation):

1. vezmeme velkou množinu p\u0159\u00edklad\u016f
2. rozd\u011bl\u00edme ji na 2 množiny – **tr\u00e9novac\u00ed** a **testovac\u00ed**
3. aplikujeme u\u010d\u00edc\u00ed algoritmus na **tr\u00e9novac\u00ed** sadu, z\u00edsk\u00e1me hypot\u011bz\u016f h
4. **zm\u011br\u00edme** procento p\u0159\u00edklad\u016f v **testovac\u00ed** sad\u011b, kter\u00e9 jsou spr\u00e1vn\u011b klasifikovan\u00e9 hypot\u011bzou h
5. opakujeme kroky 2–4 pro r\u016fzn\u00e9 velikosti tr\u00e9novac\u00edch sad a pro n\u00e1hodn\u011b vybran\u00e9 tr\u00e9novac\u00ed sady

k\u0159ivka u\u010den\u00ed – z\u00e1vislost velikosti tr\u00e9novac\u00ed sady na \u00fasp\u011b\u0161nosti



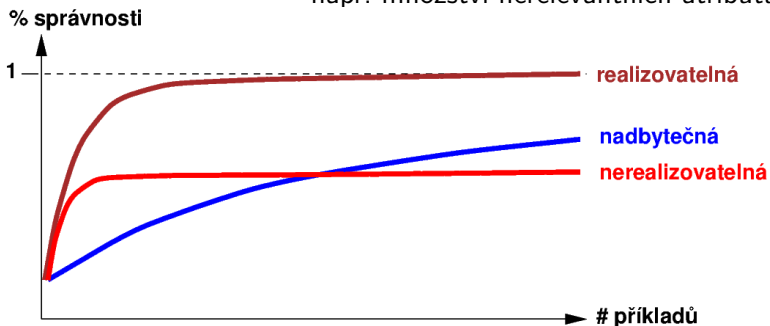
Hodnocení úspěšnosti učícího algoritmu – pokrač.

tvar křivky učení závisí na ► je hledaná funkce

realizovatelná \times nerealizovatelná
funkce může být nerealizovatelná kvůli

- chybějícím atributům
- omezenému prostoru hypotéz

► naopak **nadbytečné expresivité**
např. množství nerelevantních atributů



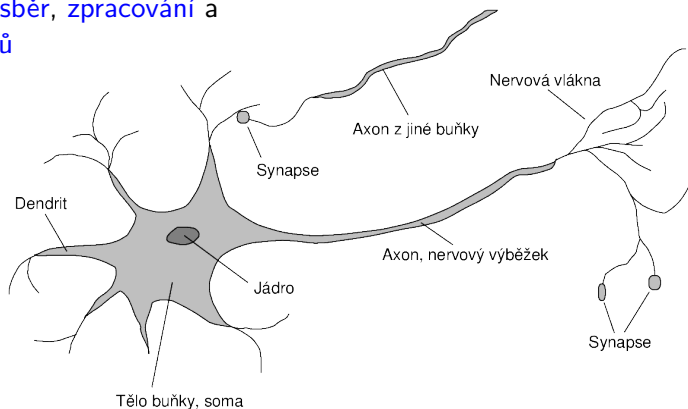
Induktivní učení – shrnutí

- ▶ **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 neuronu

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*)

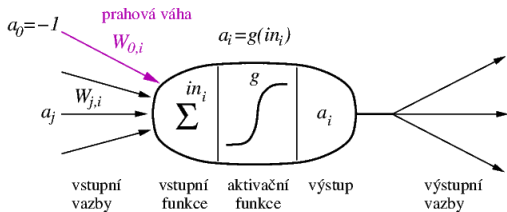
– vazba z jednotky j do i propaguje **aktivaci** a_j jednotky j

– 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

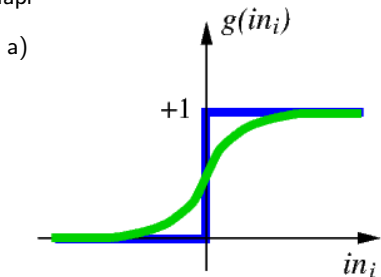
$$a_i = g(in_i) = g\left(\sum_j W_{j,i} a_j\right)$$



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

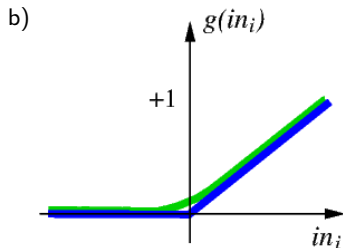


prahová funkce

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

je derivovatelná – důležité pro

učení

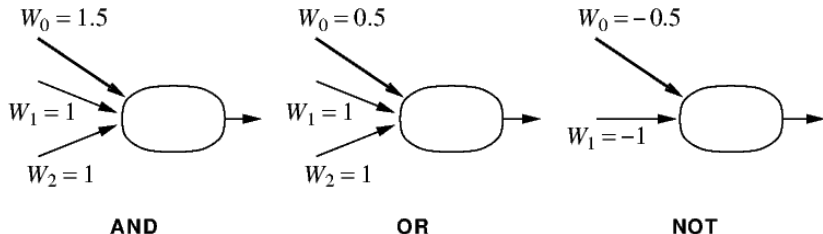


ReLU (*rectified linear unit*)

softplus $\log(1 + e^x)$

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*)

- 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
- **Long Short Term Memory (LSTM)** – spojují vzdálené závislosti v sekvenci vstupu

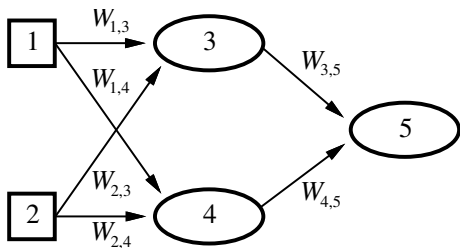


www.asimovinstitute.org/neural-network-zoo



Příklad sítě s předním vstupem

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



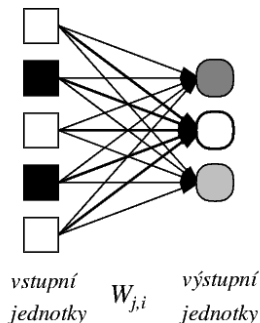
sít s předním vstupem = parametrizovaná nelineární funkce vstupu

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

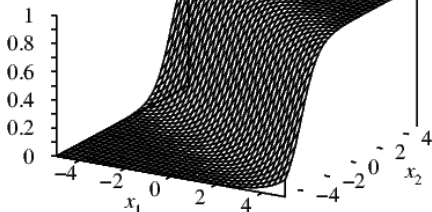
Jednovrstvá síť – perceptron

perceptron

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



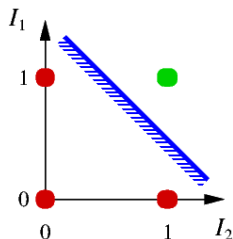
výstup perceptronu



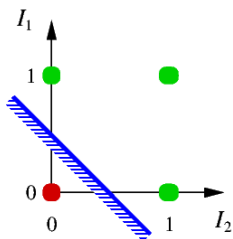
Vyjadřovací síla perceptronu

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

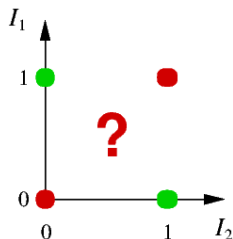
reprezentuje **lineární separátor** (nadrovina) v prostoru vstupu:



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

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

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

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

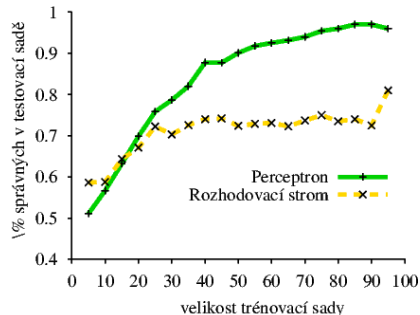
např. $\text{Err} = y - h_{\mathbf{W}}(\mathbf{x}) > 0 \Rightarrow$ výstup $h_{\mathbf{W}}(\mathbf{x})$ je moc malý
 \Rightarrow váhy se musí **zvýšit** pro pozitivní příklady a **snížit** pro negativní

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

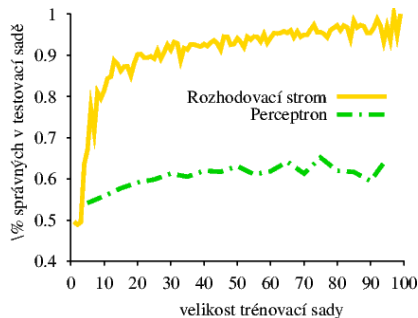
Učení perceptronu pokrač.

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

$W_{j,i}$

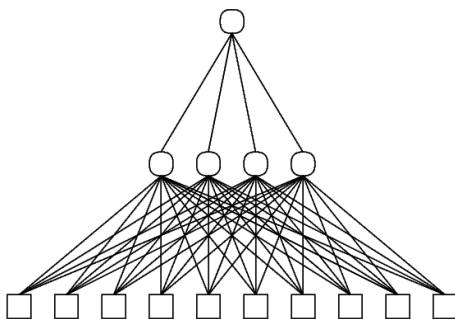
skryté jednotky

a_j

$W_{k,j}$

vstupní jednotky

a_k



Vyjadřovací síla vícevrstevných sítí

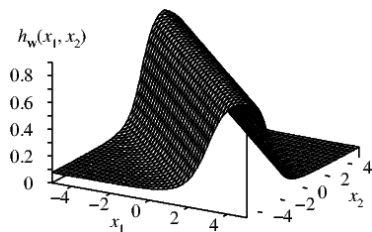
s jednou skrytou vrstvou – všechny **spojité funkce**

se dvěma skrytými vrstvami – **všechny funkce**

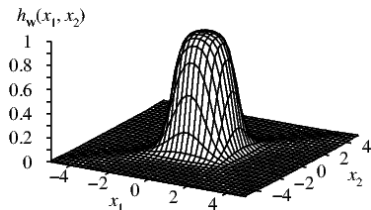
těžko se ovšem pro **konkrétní síť** zjišťuje její prostor **reprezentovatelných funkcí**

např.

dvě “opačné” skryté jednotky
vytvoří *hřbet*



dva hřbety vytvoří *homoli*



playground.tensorflow.org

Učení vícevrstvých sítí

pravidla pro úpravu vah:

- ▶ výstupní vrstva – stejně jako u perceptronu

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

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

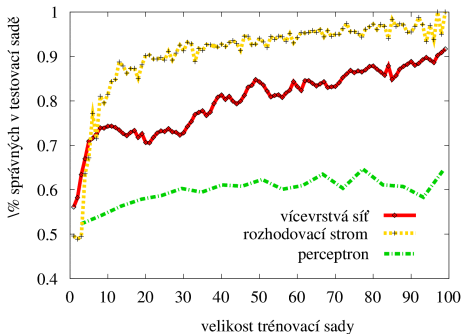
$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j \quad \text{kde} \quad \Delta_j = \mathbf{g}'(in_j) \sum_i W_{j,i} \Delta_i$$

problémy učení:

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

