Učení, rozhodovací stromy, neuronové sítě

Aleš Horák

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

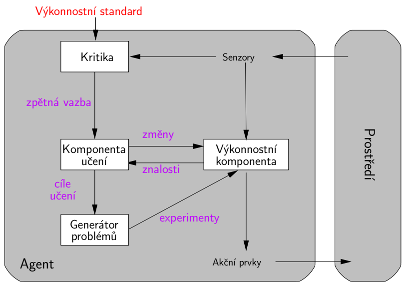
Obsah:

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

Učení

- učení 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	určení akce	axiomy *Result*	výsledné skóre
Reflexní agent	váhy perceptronu	neuronová síť	správná/špatná akce

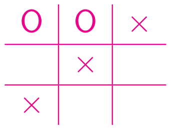
učení s dohledem (*supervised learning*) × bez dohledu (*unsupervised learning*)

- s dohledem – učení funkce z příkladů vstupů a výstupů
- bez dohledu – učení vzorů na vstupu vzhledem k reakcím prostředí
- 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

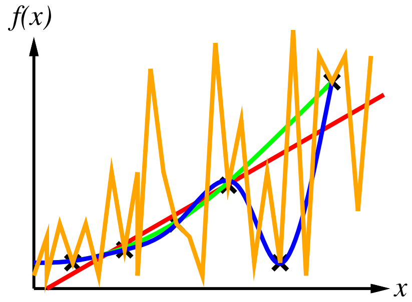
každý příklad je dvojice x, $f(x)$ např.  , $+1$

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:

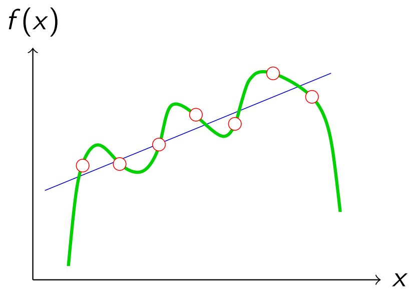


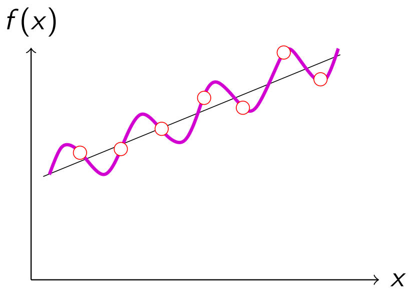
pravidlo Ockhamovy břitvy – maximalizovat kombinaci konzistence a jednoduchosti (*nejjednodušší ze správných je nejlepší*)

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 $ax + by + c \sin x$

Atributová reprezentace příkladů

příklady popsané výčtem hodnot atributů (libovolných hodnot)

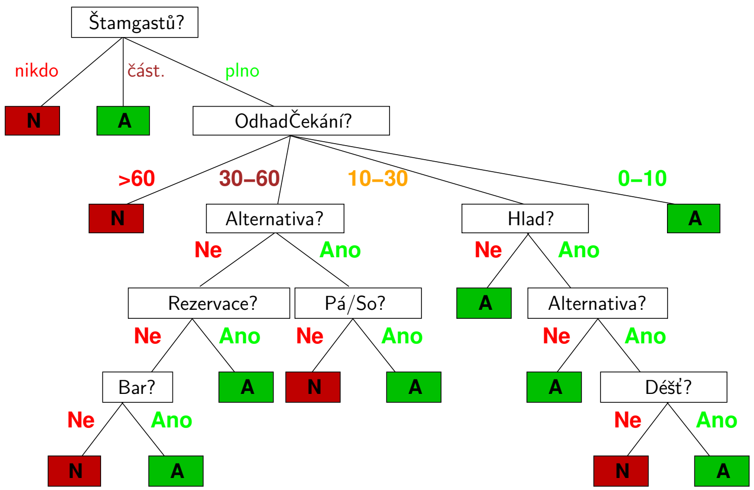
např. rozhodování, zda počkat na uvolnění stolu v restauraci:

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:



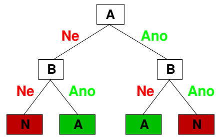
Vyjadřovací síla rozhodovacích stromů

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

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

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 ($Hlad \wedge \neg Déšť'$)
   Kolik existuje takových čistě konjunktivních hypotéz?
   každý atribut může být v pozitivní nebo negativní formě nebo nepoužit
   ⇒ 3^n různých konjunktivních hypotéz (pro 6 atributů = 729)

prostor hypotéz s větší expresivitou

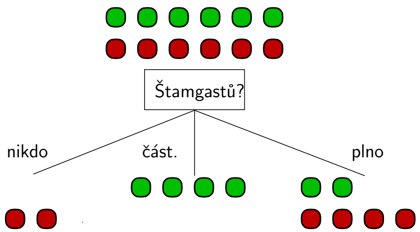
– zvyšuje šance, že najdeme přesné vyjádření cílové funkce
– ALE zvyšuje i počet možných hypotéz, které jsou konzistentní s trénovací množinou
  ⇒ můžeme získat nižší kvalitu předpovědí (generalizace)

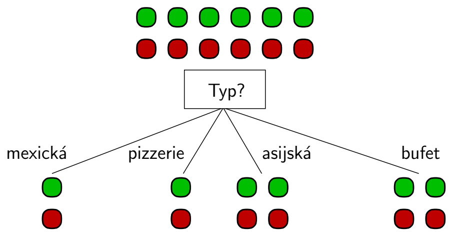
Učení ve formě rozhodovacích stromů

- 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

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

měřítko: 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

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

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:

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

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*

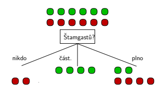
např. pro $X_1, \ldots, X_{12}$ z volby čekání na stůl je $p = n = 6$, takže potřebujeme 1 bit

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

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

$\Rightarrow$ 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 $Gain(A)$

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

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

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

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

Algoritmus IDT – příklad

```
attribute_dict  = dict( hlad=LinkedList(["ano", "ne"]),
                        stam=LinkedList(["nikdo", "cast", "plno"]),
                        cen=LinkedList(["$", "$$", "$$$"]), ...
example_list  = LinkedList([
    ("pockat", LinkedList([
        ("alt", "ano"), ("bar", "ne"), ("paso", "ne"), ("hlad", "ano"), ("stam", "cast"),
        ("cen", "$$$"), ("dest", "ne"), ("rez", "ano"), ("typ", "mexicka") ])),
    ("necekat", LinkedList([
        ("alt", "ano"), ("bar", "ne"), ("paso", "ne"), ("hlad", "ano"), ("stam", "plno"),
        ("cen", "$"), ("dest", "ne"), ("rez", "ne"), ("typ", "asijska") ])), ...
print_tree(induce_tree(attribute_dict.keys(),example_list))
stam?
  = nikdo
    necekat
  = cast
    pockat
  = plno
    hlad?
      = ano
        cen?
          = $
            paso?
              = ano
                pockat
              = ne
                necekat
          = $$$
            necekat
      = ne
```


Algoritmus IDT – učení formou rozhodovacích stromů

```
def induce_tree(attributes, examples):
    if examples == Nil: return None
    example = examples.head
    class_, _ = example
    other_class = False
    for example1 in member_anyX(examples):
        classX, _ = example1
        if classX != class_: other_class = True; break
    if other_class is False: return ("leaf", class_)  # ∀ příklady stejné třídy
    attribute, _ = choose_attribute(attributes, examples)
    if attribute is None: # žádný užitečný atribut, list s distribucí klasifikací
        exclasses = get_example_classes(examples)
        return ("leaf", exclasses)
    rest_atts = dele(attribute, attributes)
    values = get_attribute_values(attribute)
    subtrees = induce_trees(attribute, values, rest_atts, examples)
    return ("tree", attribute, subtrees)
```

je v příkladech více tříd?

podstromy podle nejlepšího atributu

Algoritmus IDT – učení formou rozhodovacích stromů

```
def induce_trees(att, vals, rest_atts, exs): # SubTrees podle hodnot atributu Att
    if vals is Nil: return Nil #   žádné atributy → žádné podstromy
    val1 = vals.head
    example_subset = attval_subset(att, val1, exs)
    tree1 = induce_tree(rest_atts, example_subset)
    trees = induce_trees(att, vals.tail, rest_atts, exs)
    return Cons((val1, tree1), trees)

def attval_subset(attribute, value, examples): # vybere příklady, kde Attribute = Value
    return filter_examples(examples, None, attribute, value)

def choose_attribute(atts, examples): #    výběr nejlepšího atributu
    if atts == Nil: return (None, 0)
    att = atts.head
    if atts.tail == Nil:
        gain_ = gain(examples, att)
        return (att, gain_)
    best_att1, best_gain1 = choose_attribute(atts.tail, examples)
    gain_ = gain(examples, att)
    if gain_ > best_gain1: return (att, gain_)
    return (best_att1, best_gain1)
```

příklady pro **att=val1**

atribut s nejvyšším ziskem na sadě příkladů

Algoritmus IDT – učení formou rozhodovacích stromů

```
def gain(exs, att): #   zisk atributu  Gain(A) = I(⟨P(v1), ..., P(vn)⟩) − Remainder(A)
    att_vals = get_attribute_values(att)
    total = length(exs) #          počet příkladů  p_i + n_i nebo p_{c_i,1} + ... + p_{c_i,k}
    classes = get_example_classes(exs)
    ccnts = cnt_classes(classes, exs) #    počty příkladů ve třídách  p + n nebo p_{c_1} + ... + p_{c_k}
    i = info(ccnts, total)
    rem_ = rem(att, att_vals, exs, classes, total)
    gain_ = i − rem_
    return gain_

def info(value_counts, total): # míra informace  I(⟨P(v1), ..., P(vn)⟩) = Σ_{i=1}^{n} −P(v_i) log2 P(v_i)
    if value_counts == Nil: return 0
    vc = value_counts.head
    i1 = info(value_counts.tail, total)
    if vc == 0: return i1
    pvi = vc / total
    return −pvi ∗ math.log(pvi, 2) + i1

def rem(att, vs, exs, classes, total): # "zbytková informace" po testu na Att
    if vs == Nil: return 0
    v = vs.head
    nv = length(filter_examples(exs, None, att, v))
    vcnts = cnt_classes_attv(att, v, classes, exs)
    pv = nv / total
    i = info(vcnts, nv)
    rem1 = rem(att, vs.tail, exs, classes, total)
    return pv ∗ i + rem1 #      Remainder(A) = Σ_i P(v_i) · I(⟨P(c_{i,1}), ..., P(c_{i,k})⟩)
```

Algoritmus IDT – učení formou rozhodovacích stromů

```
def cnt_classes(classes, exs): # kolik příkladů má každá třída ze seznamu?
    if classes == Nil:
        return Nil
    c = classes.head
    nc = cnt_class(c, exs)
    ncs = cnt_classes(classes.tail, exs)
    return Cons(nc, ncs)

def cnt_class(class_, exs): # kolik příkladů má danou třídu?
    count = 0
    for example in member_anyX(exs):
        class1, _ = example
        if class1 == class_:
            count = count + 1
    return count

def cnt_classes_attv(att, val, classes, exs): # počty příkladů každé třídy s Att = Val
    if classes == Nil: return Nil
    c = classes.head
    nc = cnt_class_attv(att, val, c, exs)
    ncs = cnt_classes_attv(att, val, classes.tail, exs)
    return Cons(nc, ncs)

def cnt_class_attv(att, val, class_, exs): # počet příkladů třídy Class s Att = Val
    return length(filter_examples(exs, class_, att, val))
```

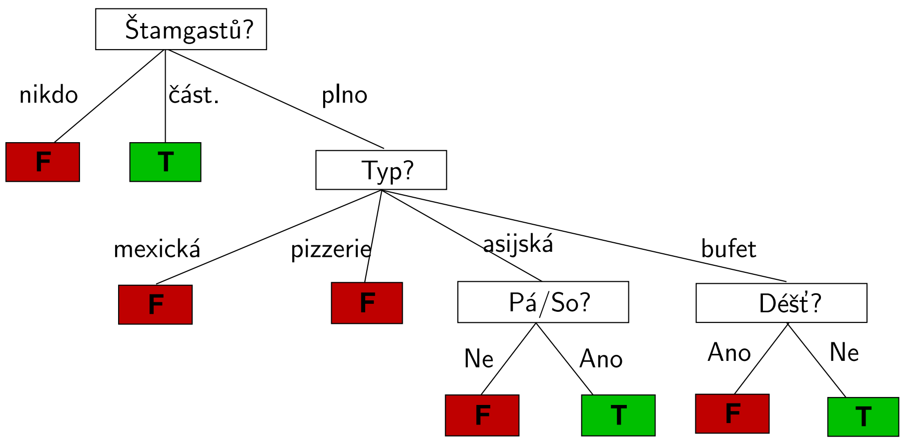

Algoritmus IDT – učení formou rozhodovacích stromů

```
def get_example_classes(examples): # vrátí třídy příkladů
    if examples == Nil: return Nil
    example = examples.head
    class_, _ = example
    other_classes = get_example_classes(examples.tail)
    if not member(class_, other_classes):
        return Cons(class_, other_classes)
    return other_classes

# filtruj příklady podle hodnoty atributu a volitelně i podle třídy výstupu
def filter_examples(examples, class_, attribute, value):
    if examples == Nil: return Nil
    example = examples.head
    class1, obj = example
    other_examples = filter_examples(examples.tail, class_, attribute, value)
    if class_ is None or class_ == class1:
        if member((attribute, value), obj): return Cons(example, other_examples)
    return other_examples
```


IDT – výsledný rozhodovací strom

rozhodovací strom naučený z 12-ti příkladů:



podstatně jednodušší než strom "z tabulky příkladů"

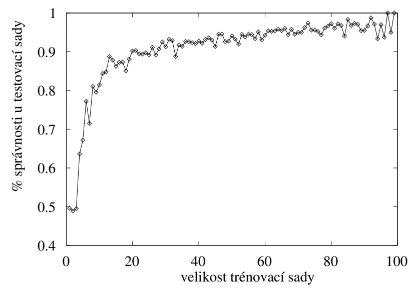
Hodnocení úspěšnosti učícího algoritmu

jak můžeme zjistit, zda $h \approx f$?
- dopředu – použít věty Teorie komputačního učení
- po naučení – kontrolou na jiné trénovací sadě

používaná metodologie (cross validation):

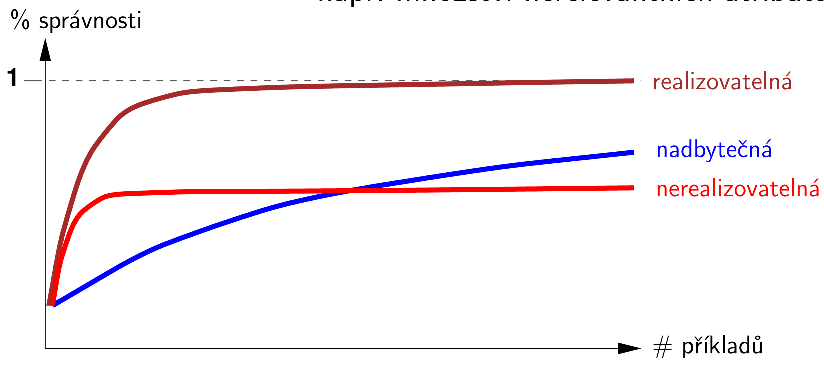
1. vezmeme velkou množinu příkladů
2. rozdělíme ji na 2 množiny – trénovací a testovací
3. aplikujeme učící algoritmus na trénovací sadu, získáme hypotézu h
4. změříme procento příkladů v testovací sadě, které jsou správně klasifikované hypotézou h
5. opakujeme kroky 2–4 pro různé velikosti trénovacích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

křivka učení – závislost velikosti trénovací sady na úspěšnosti



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



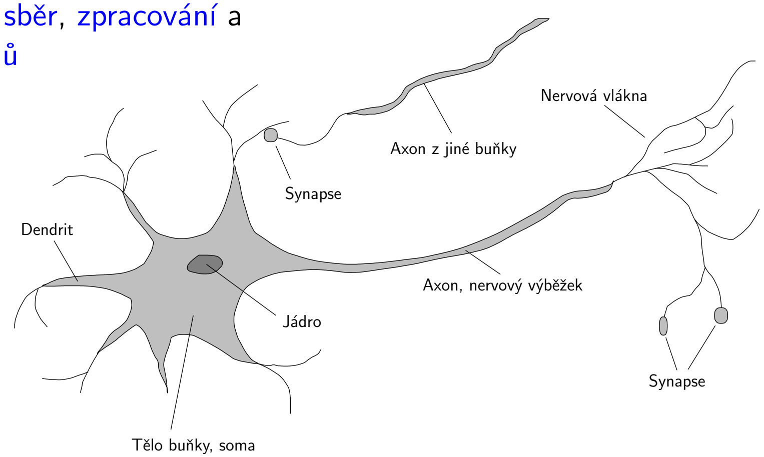
Induktivní učení – shrnutí

- 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



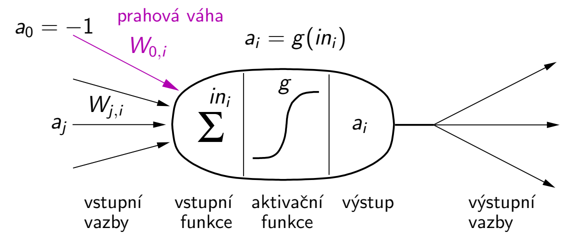
Počítačový model – neuronové sítě

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

jednotky v neuronové síti (*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

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

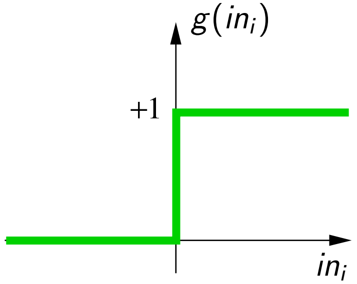


Aktivační funkce

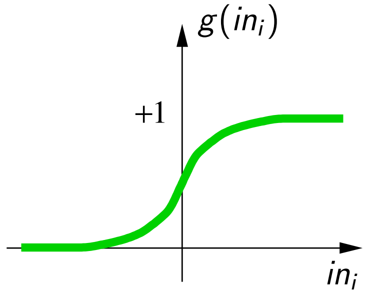
účel aktivační funkce:
- jednotka má být aktivní ($\approx +1$) pro pozitivní příklady, jinak neaktivní ≈ 0
- aktivace musí být nelineární, jinak by celá síť byla lineární

např.

a)



b)

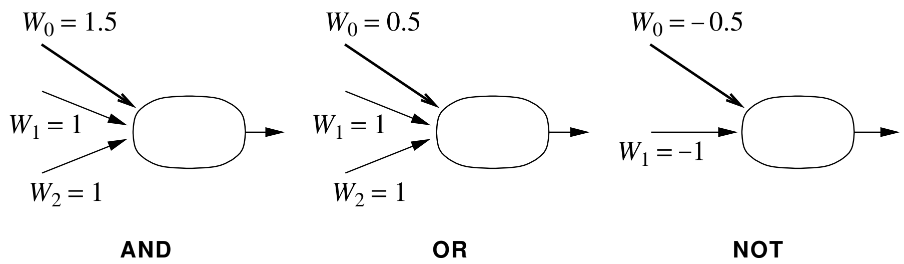


prahová funkce

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

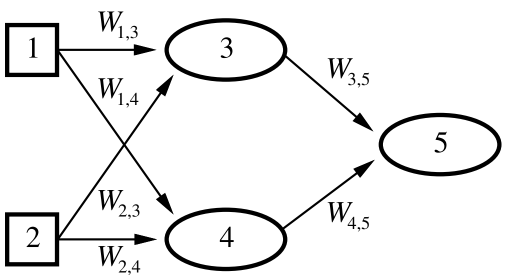
$\Rightarrow$ kombinacemi jednotek do sítě můžeme implementovat libovolnou Booleovskou funkci

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

Příklad sítě s předním vstupem

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

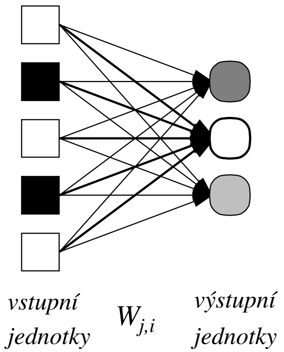


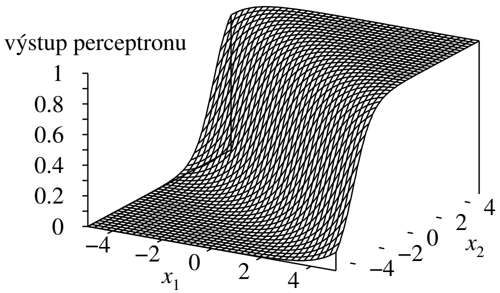
síť s předním vstupem = parametrizovaná nelineární funkce vstupu

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

Jednovrstvá síť – perceptron

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



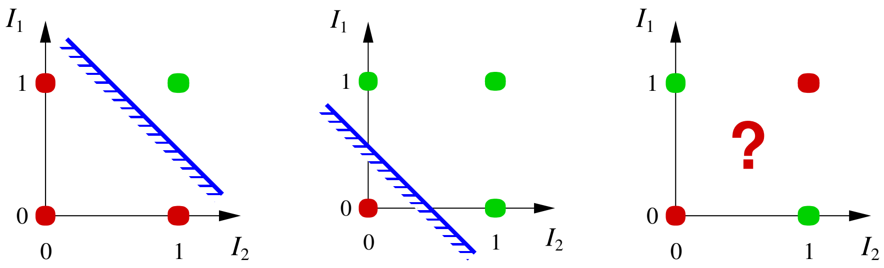


Vyjadřovací síla perceptronu

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

$\sum_j W_j x_j > 0$ *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, aby se snížila chyba na trénovací sadě

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

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

váhy pro minimální chybu pak hledáme optimalizačním prohledáváním spojitého prostoru vah

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

pravidlo pro úpravu váhy

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

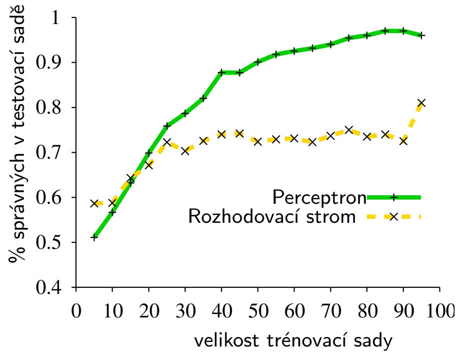
např. $Err = y - h_{\mathbf{W}}(\mathbf{x}) > 0$ $\Rightarrow$ výstup $h_{\mathbf{W}}(\mathbf{x})$ je moc malý
$\Rightarrow$ váhy se musí *zvýšit* pro pozitivní příklady a *snížit* pro negativní

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

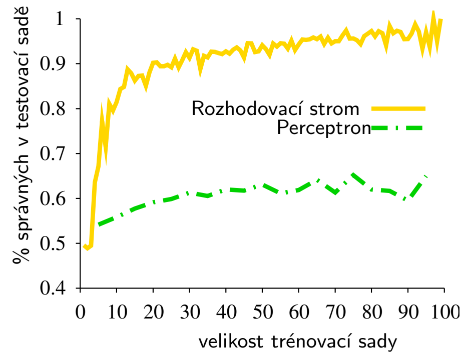
Učení perceptronu pokrač.

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

a) učení majoritní funkce

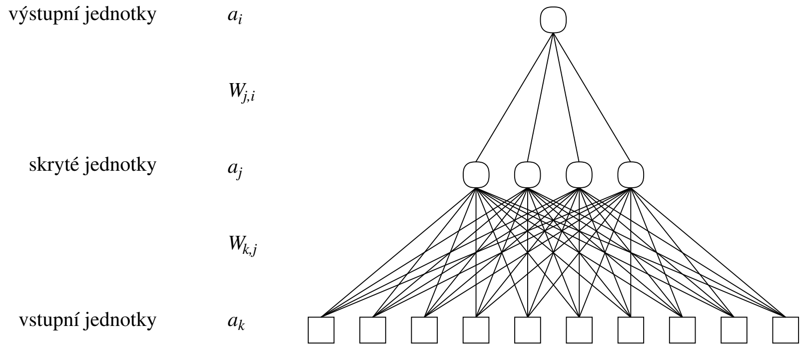


b) učení čekání na volný stůl v restauraci



Vícevrstvé neuronové sítě

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

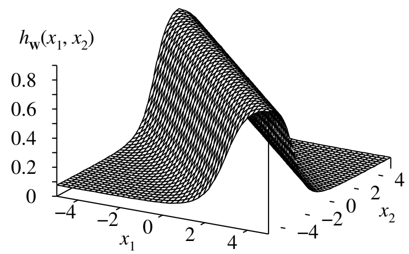


Vyjadřovací síla vícevrstvých sítí

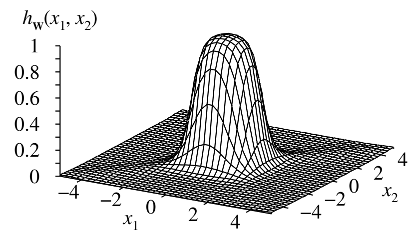
s jednou skrytou vrstvou – všechny spojité funkce
se dvěma skrytými vrstvami – všechny funkce
těžko se ovšem pro konkrétní síť zjišťuje její prostor reprezentovatelných funkcí

např.

dvě "opačné" skryté jednotky vytvoří *hřbet*



dva hřbety vytvoří *homoli*



Učení vícevrstvých sítí

pravidla pro úpravu vah:

- výstupní vrstva – stejně jako u perceptronu

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

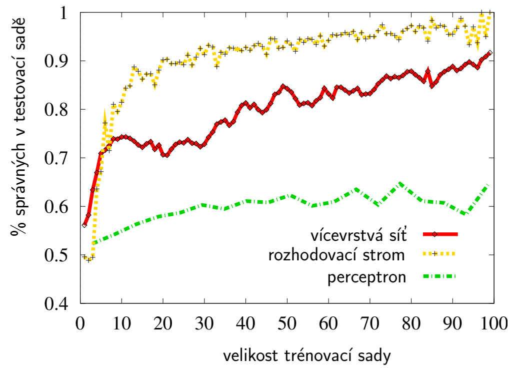
- skryté vrstvy – zpětné šíření (*back-propagation*) chyby z výstupní vrstvy

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

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 ≈ 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
  - ...
- v posledních letech hluboké neuronové sítě – lépe generalizují