# Učení, rozhodovací stromy, neuronové sítě

**Aleš Horák**

**E-mail: hales@fi.muni.cz**

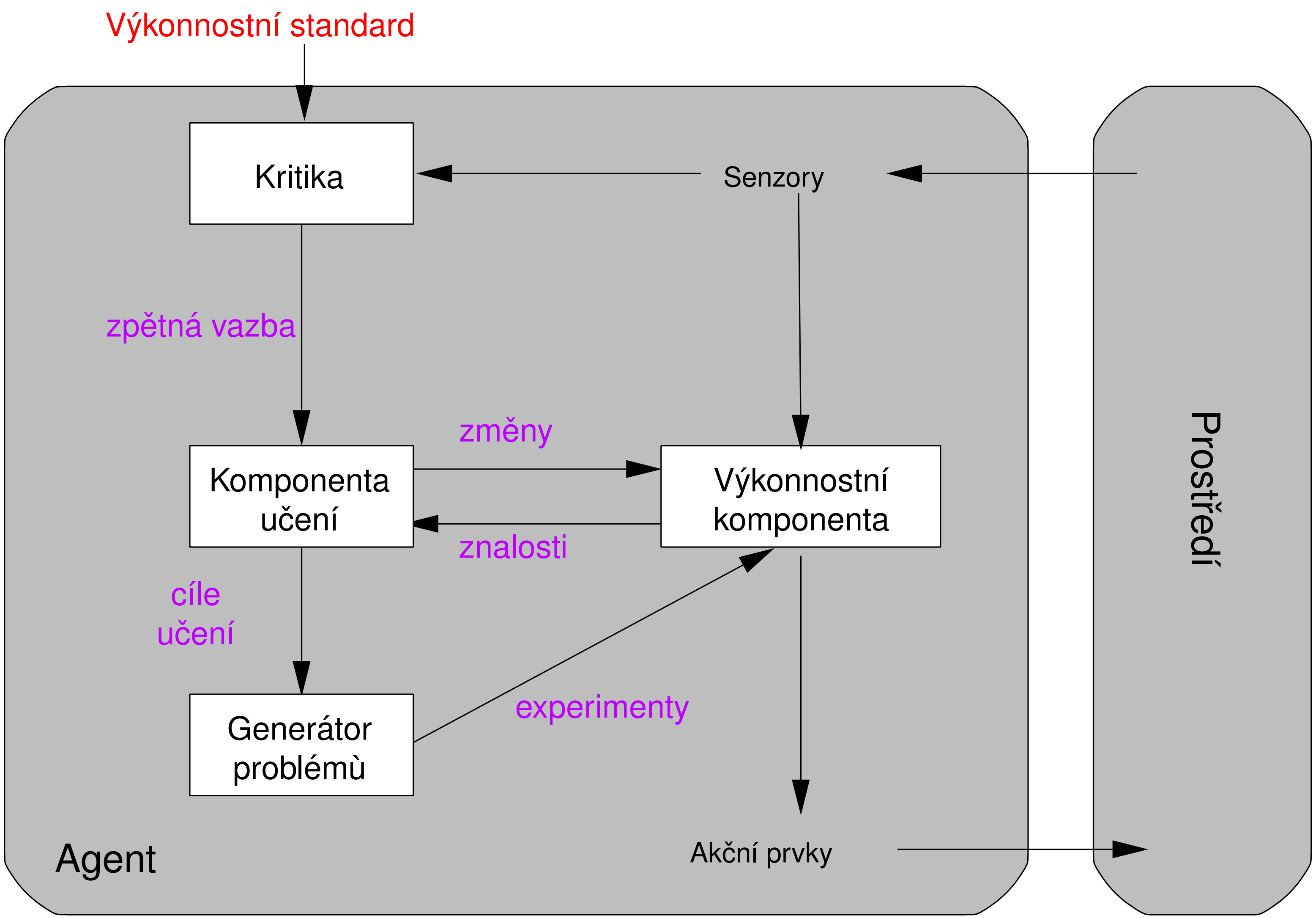
**http://nlp.fi.muni.cz/uui/**

Obsah:

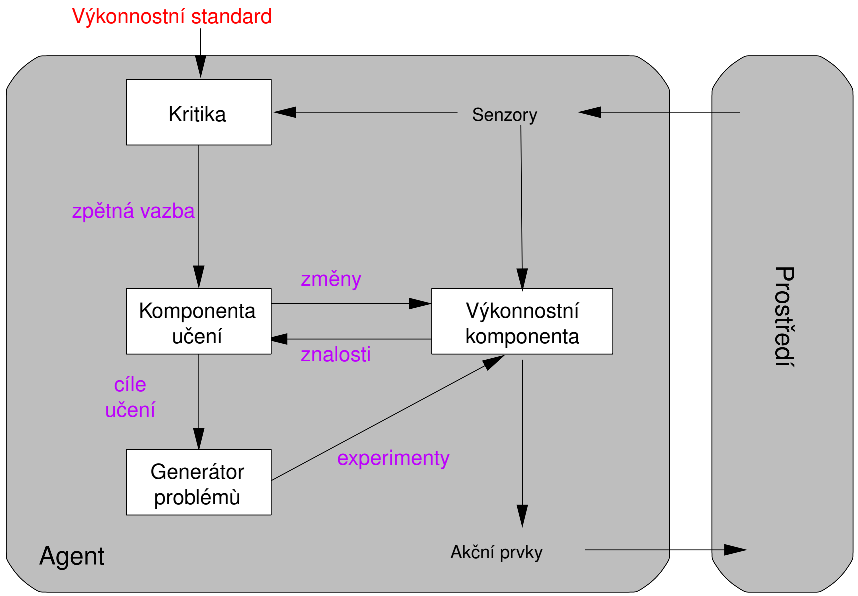
# UČENÍ

- 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



# 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 předjíždění je nevhodné, a modifikuje odpovídajícím způsobem Generátor problémů
- **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

příklady:

| 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 preceptronu | neuronová síť | správná/špatná akce |

# 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

příklady:

| 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 preceptronu | neuronová síť | správná/špatná akce |

učení s dohledem (*supervised learning*) $\times$ bez dohledu (*unsupervised learning*)

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

# INDUKTIVNÍ UČENÍ

známé taky jako věda ☺

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

$f$ je cílová funkce

příklad je dvojice $x, f(x)$ např.

| | | |
|---|---|---|
| O | O | × |
| | × | |
| × | | |

$, \quad +1$

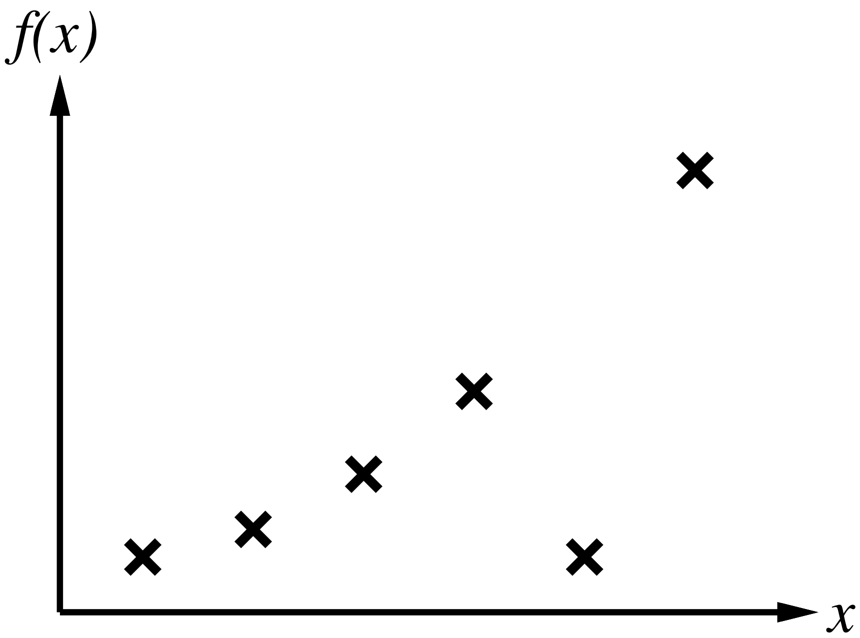
úkol indukce: najdi hypotézu $h$
takovou, že $h \approx f$
pomocí sady trénovacích příkladů

# Metoda induktivního učení

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

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

např. hledání křivky:

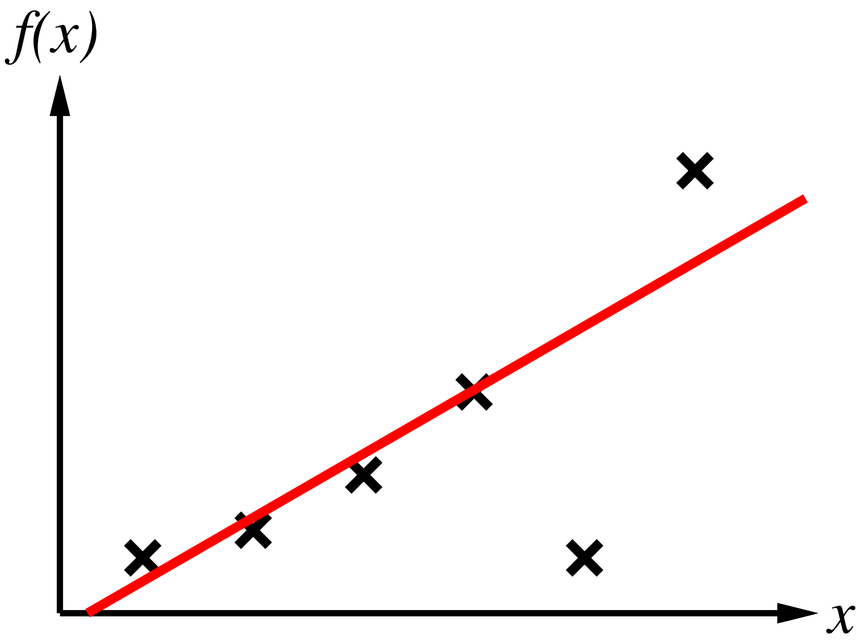


# Metoda induktivního učení

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

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

např. hledání křivky:

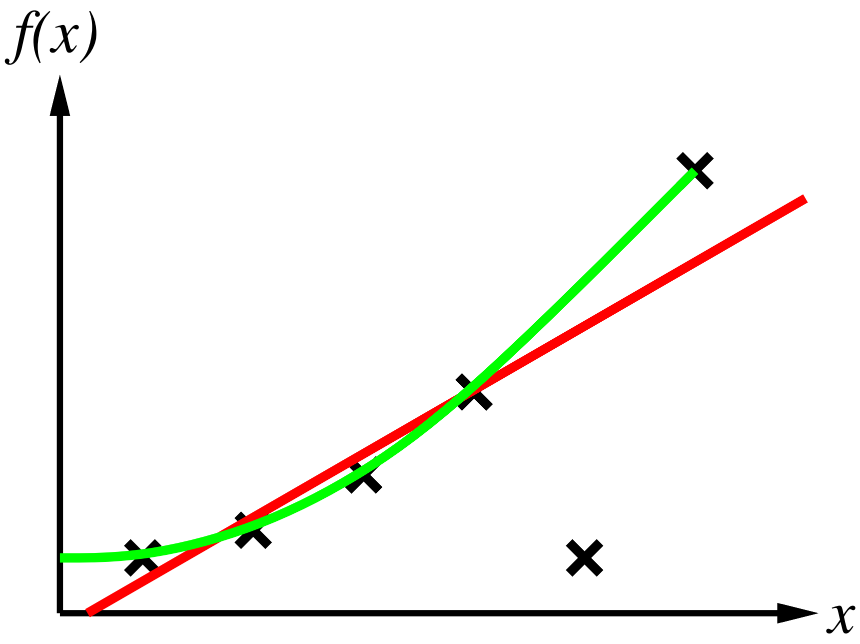


# Metoda induktivního učení

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

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

např. hledání křivky:

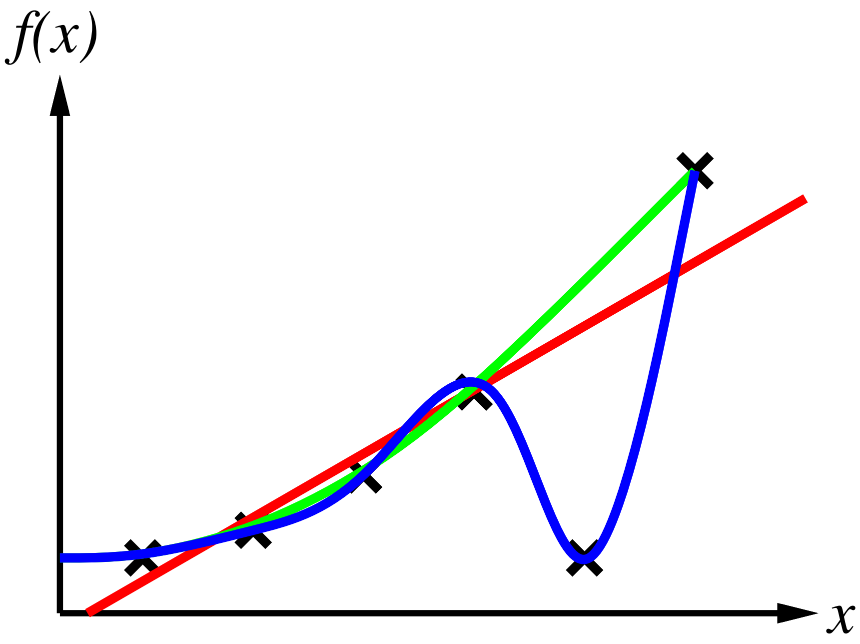


# Metoda induktivního učení

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

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

např. hledání křivky:

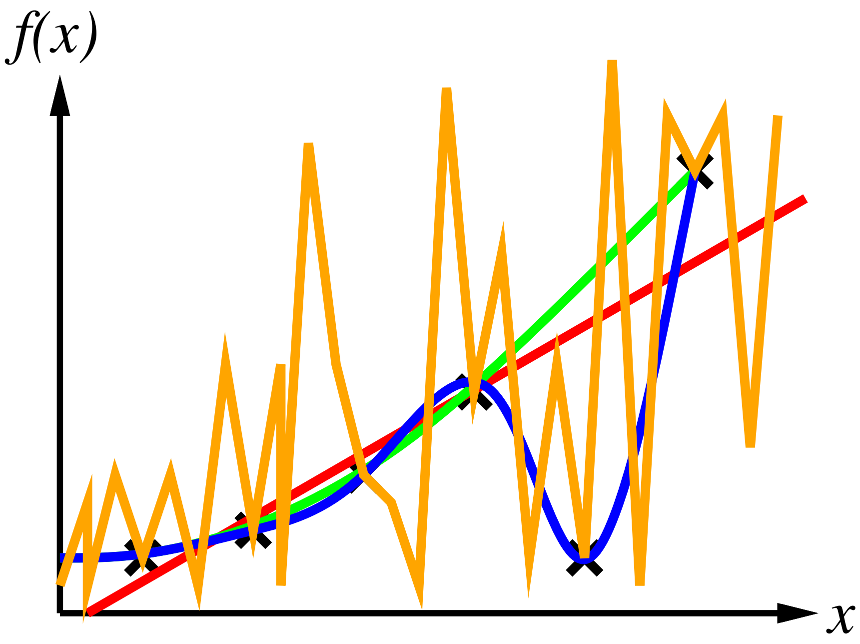


# METODA INDUKTIVNÍHO UČENÍ

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

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

např. hledání křivky:

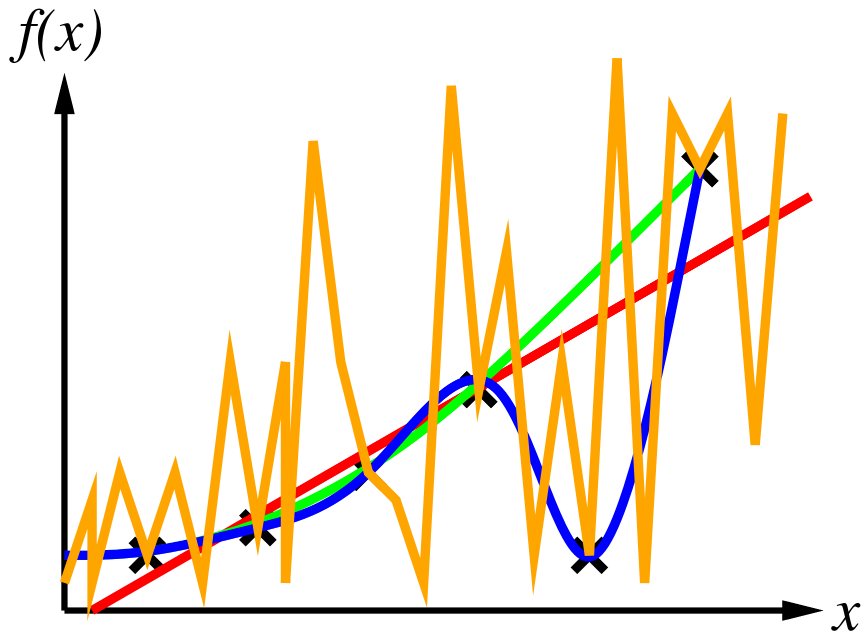


## METODA INDUKTIVNÍHO UČENÍ

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

$h$ je konzistentní $\Leftrightarrow$ souhlasí f $f$ 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:

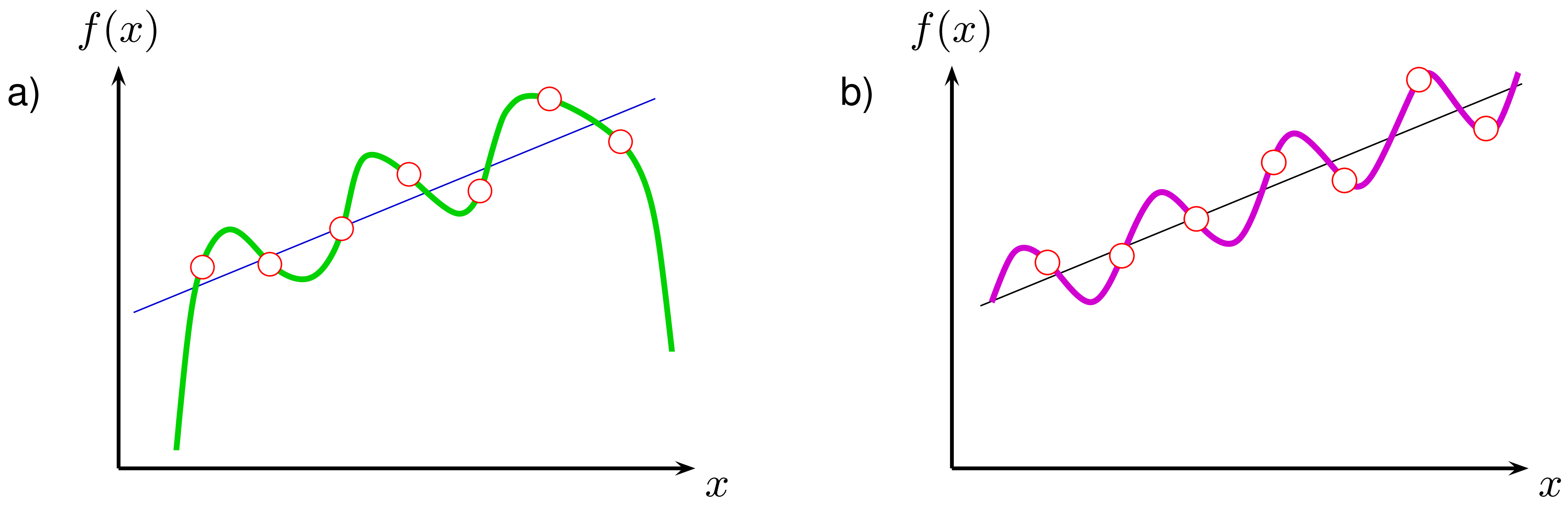
- pokrýt co největší množství hledaných funkcí
- udržet nízkou výpočetní složitost hypotézy

# METODA INDUKTIVNÍHO UČENÍ pokrač.

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

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

např.




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

# ATRIBUTOVÁ REPREZENTACE PŘÍKLADŮ

příklady popsané výčtem hodnot atributů (libovolných hodnot)

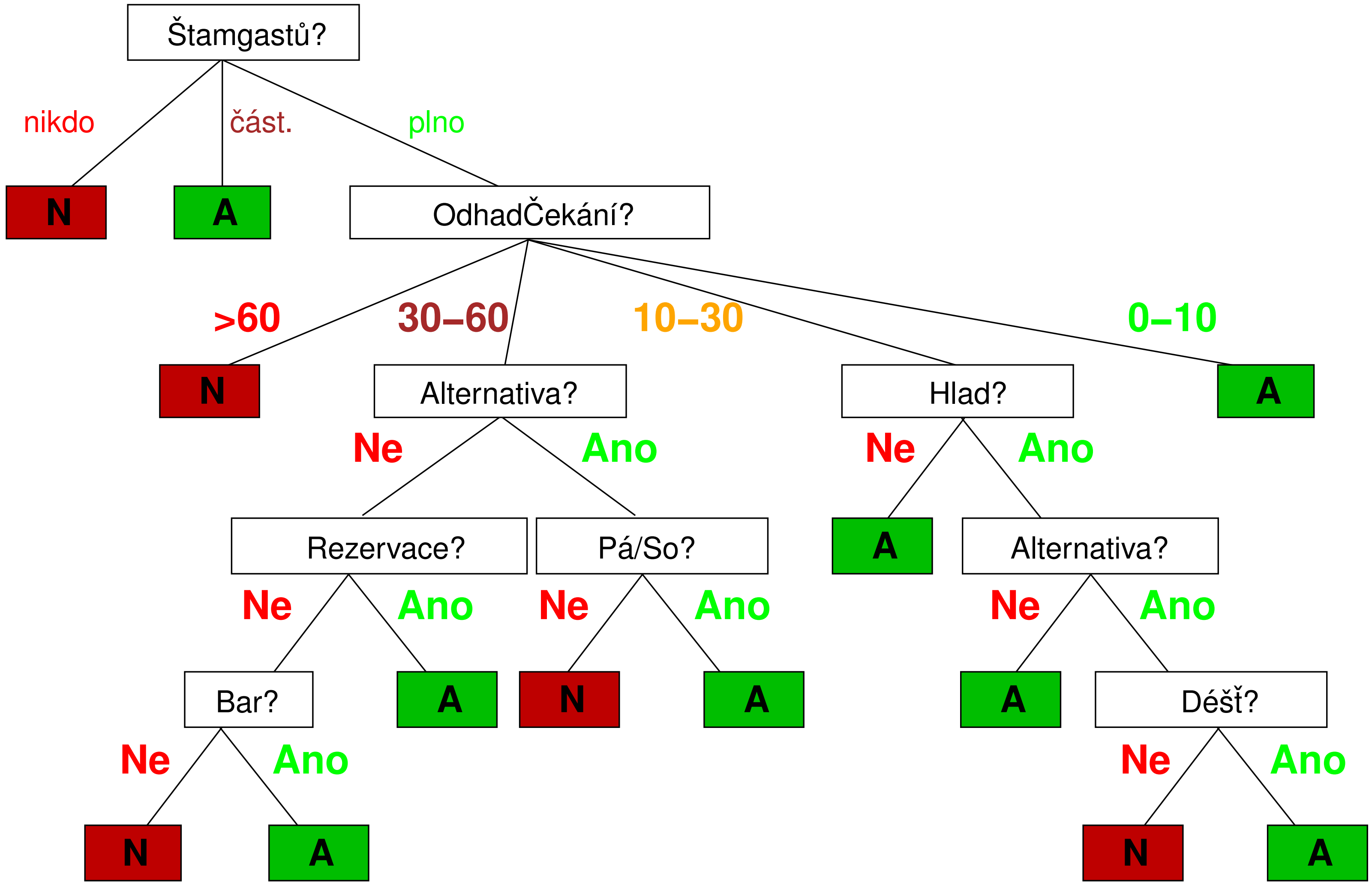
např. rozhodování, zda počkat na uvolnění stolu v restauraci:

| Příklad | Atributy | | | | | | | | | | počkat? |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | $Alt$ | $Bar$ | $Pá/So$ | $Hlad$ | $Štam$ | $Cen$ | $Déšť$ | $Rez$ | $Typ$ | $ČekD$ | |
| $X_1$ | 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:




slajd 18

## Vyjadřovací síla rozhodovacích stromů

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

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

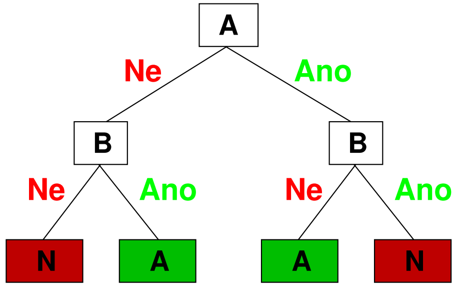
## Vyjadřovací síla rozhodovacích stromů

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

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

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 |



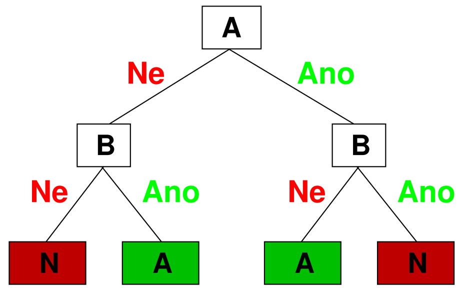
# Vyjadřovací síla rozhodovacích stromů

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

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

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

| A | B | A xor B |
|---|---|---|
| F | F | F |
| F | T | T |
| T | F | T |
| T | T | F |




triviálně

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

ale takový strom pravděpodobně nebude generalizovat na nové příklady

chceme najít co možná kompaktní rozhodovací strom

## PROSTOR HYPOTÉZ

1. vezměme pouze Booleovské atributy, bez dalšího omezení

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

## 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

## 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

# 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

# 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 ($Hlad \land \neg Déšt'$)

   Kolik existuje takových čistě konjunktivních hypotéz?

# 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 ($Hlad \wedge \neg Déšt'$)

   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

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 ($Hlad \wedge \neg Déšt'$)

   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

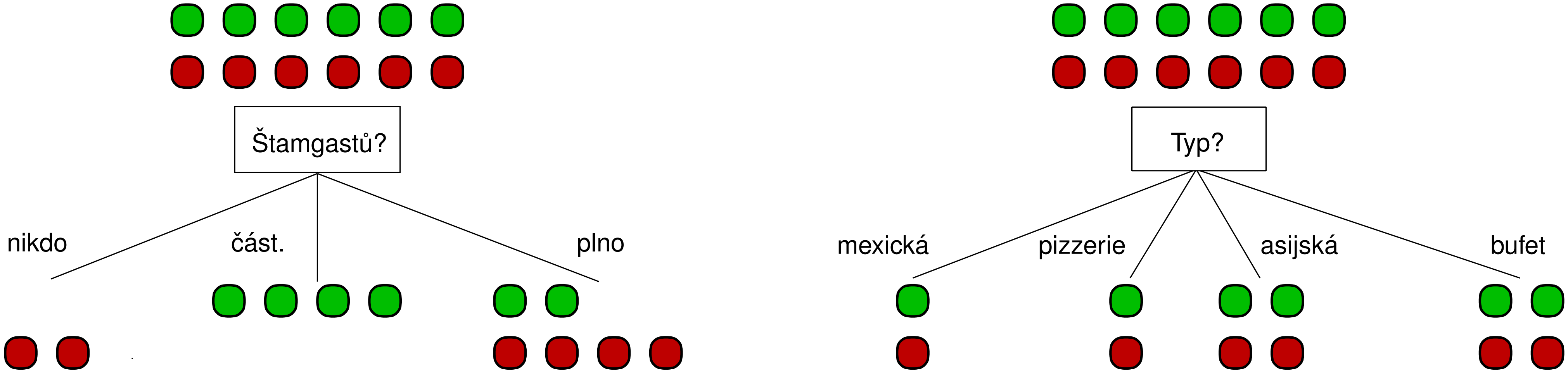
- **triviální konstrukce rozhodovacího stromu**
  - pro každý příklad v trénovací sadě přidej jednu cestu od kořene k listu
  - na stejných příkladech jako v trénovací sadě bude fungovat přesně
  - na nových příkladech se bude chovat náhodně – negeneralizuje vzory z příkladů, pouze kopíruje pozorování

## UČENÍ VE FORMĚ ROZHODOVACÍCH STROMŮ

- **triviální konstrukce rozhodovacího stromu**
  - pro každý příklad v trénovací sadě přidej jednu cestu od kořene k listu
  - na stejných příkladech jako v trénovací sadě bude fungovat přesně
  - na nových příkladech se bude chovat náhodně – negeneralizuje vzory z příkladů, pouze kopíruje pozorování
- **heuristická konstrukce kompaktního stromu**
  - chceme najít nejmenší rozhodovací strom, který souhlasí s příklady
  - vlastní nalezení nejmenšího stromu je ovšem příliš složité
    $\longrightarrow$ heuristikou najdeme alespoň dostatečně malý ☺
  - hlavní myšlenka – vybíráme atributy pro test v co nejlepším pořadí

# Výběr atributu

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




$\check{S}tamgast\mathring{u}?$ 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 $\longrightarrow$ tím více informace je v ní obsaženo

měřítko:

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

# Výběr atributu – míra informace

informace – odpovídá na otázku

čím méně dopředu vím o výsledku obsaženém v odpovědi $\longrightarrow$ tím více informace je v ní obsaženo

měřítko:

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

pro pravděpodobnosti všech odpovědí $\langle P(v_1), \ldots, P(v_n)\rangle$ $\longrightarrow$ míra informace v odpovědi obsažená

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

tato míra se také nazývá entropie

# Výběr atributu – míra informace

informace – odpovídá na otázku

čím méně dopředu vím o výsledku obsaženém v odpovědi $\longrightarrow$ tím více informace je v ní obsaženo

měřítko:

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

pro pravděpodobnosti všech odpovědí $\langle P(v_1), \ldots, P(v_n)\rangle \longrightarrow$ míra informace v odpovědi obsažená

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

tato míra se také nazývá entropie

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

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

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

## Použití míry informace pro výběr atributu

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

$\Rightarrow I\left(\langle \frac{p}{p+n}, \frac{n}{p+n} \rangle\right)$ 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

## Použití míry informace pro výběr atributu

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

$\Rightarrow I\left(\langle\frac{p}{p+n}, \frac{n}{p+n}\rangle\right)$ 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 $A$?*

# Použití míry informace pro výběr atributu

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

$\Rightarrow I\left(\langle \frac{p}{p+n}, \frac{n}{p+n} \rangle\right)$ 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

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

$\Rightarrow$ $I\left(\langle\frac{p}{p+n},\frac{n}{p+n}\rangle\right)$ 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

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

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

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

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

$\Rightarrow$ 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 $Gain(A)$

$Gain(Štamgastů?)\approx 0.541$ bitů $\qquad$ $Gain(Typ?)=0$ bitů

# Algoritmus IDT – učení formou rozhodovacích stromů

```
% induce_tree( Attributes , Examples, Tree)
induce_tree( _ , [],  null )  :- !.
induce_tree( _ , [ example( Class,_ ) | Examples], leaf( Class))  :-
    not (member( example( ClassX, _), Examples), ClassX \== Class), !. % ∀ příklady stejné klasifikace
induce_tree( Attributes , Examples, tree( Attribute , SubTrees)) :-
    choose_attribute( Attributes , Examples, Attribute ), !,
    del( Attribute , Attributes , RestAtts),
    attribute( Attribute , Values),
    induce_trees( Attribute , Values, RestAtts, Examples, SubTrees).
induce_tree( _, Examples, leaf( ExClasses)) :-      % žádný užitečný atribut, list s stribucí klasifikací
     findall ( Class, member( example( Class, _), Examples), ExClasses).

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

% attval_subset( Attribute = Value, Examples, Subset):
%   Subset je podmnožina příkladů z Examples, které splňují podmínku Attribute = Value
attval_subset ( AttributeValue , Examples, ExampleSubset) :-
    findall ( example( Class, Obj),
              (member( example( Class, Obj), Examples), satisfy( Obj, [ AttributeValue ])),
              ExampleSubset).
```

## Algoritmus IDT – učení formou rozhodovacích stromů pokrač.

```
% satisfy( Object, Description)
satisfy( Object, Conj) :- not (member( Att = Val, Conj), member( Att = ValX, Object), ValX \== Val).

% vybíráme atribut podle “ čistoty ” množin, na které rozdělí příklady, setof je setřídí podle Impurity
choose_attribute( Atts, Examples, BestAtt) :-
    setof( Impurity/Att, ( member( Att, Atts), impurity1(Examples, Att, Impurity )), [MinImpurity/BestAtt|_]).

impurity1( Exs, Att, Imp) :- attribute( Att, AttVals ), term_sum( Exs, Att, AttVals , 0, Imp).

% term_sum( Exs, Att, AttVals, PartialSum, Sum) – vážená suma “čistoty” přes ∀ hodnoty atributu Att
term_sum( _, _ , [], Sum, Sum).
term_sum( Exs, Att, [Val | Vals ], PartSum, Sum) :- length( Exs, N),
    findall ( C, (member( example(C,Desc), Exs), satisfy( Desc, [Att=Val])), ExClasses),
    % ExClasses = seznam klasifikací (s opakováním) všech příkladů s Att=Val
    length( ExClasses, NV), NV > 0, !,
    findall ( P, (bagof( 1, member( Class, ExClasses), L), length( L, NVC), P is NVC/NV), ClassDistribution),
    gini ( ClassDistribution , GiniI ),
    NewPartSum is PartSum + GiniI*NV/N,
    term_sum( Exs, Att, Vals, NewPartSum, Sum)
    ; term_sum( Exs, Att, Vals, PartSum, Sum).      % žádné příklady nesplňují Att = Val
```

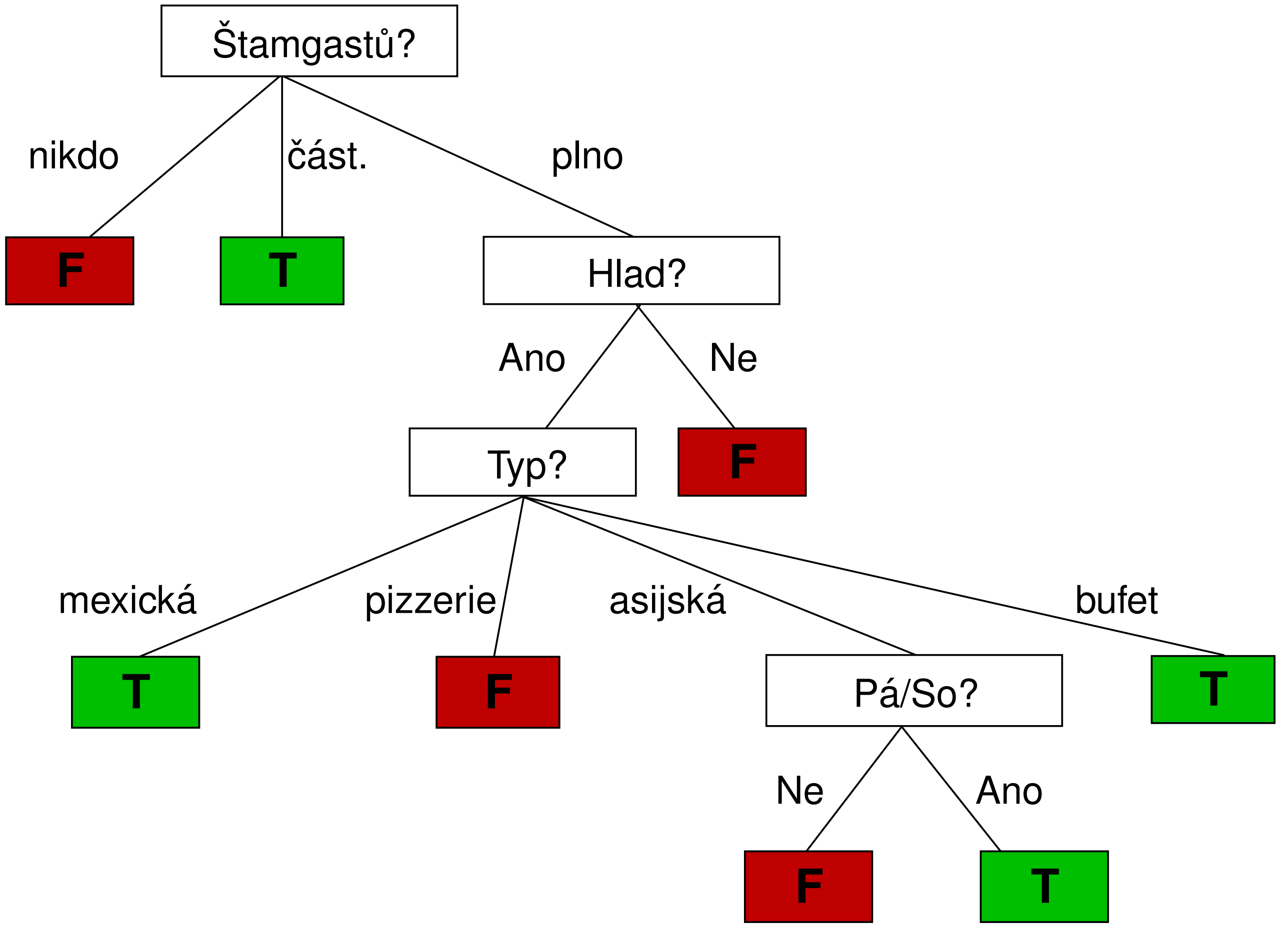
% gini( ProbabilityList , GiniIndex) – míra “ čistoty ”, $GiniIndex = \sum_{\forall i,j: i \neq j} P_i \cdot P_j \equiv 1 - \sum_{\forall i} P_i \cdot P_i$

```
gini ( Probs, Index) :- square_sum( Probs, 0, SquareSum), Index is 1 - SquareSum.

square_sum( [], S, S).
square_sum( [P | Ps], PartS, S) :- NewPartS is PartS + P*P, square_sum( Ps, NewPartS, S).
```

# IDT – VÝSLEDNÝ ROZHODOVACÍ STROM

rozhodovací strom naučený z 12-ti příkladů:




podstatně jednodušší než strom “z tabulky příkladů” (slajd 9)

## Hodnocení úspěšnosti učícího algoritmu

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

## HODNOCENÍ ÚSPĚŠNOSTI UČÍCÍHO ALGORITMU

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

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

# Hodnocení úspěšnosti učícího algoritmu

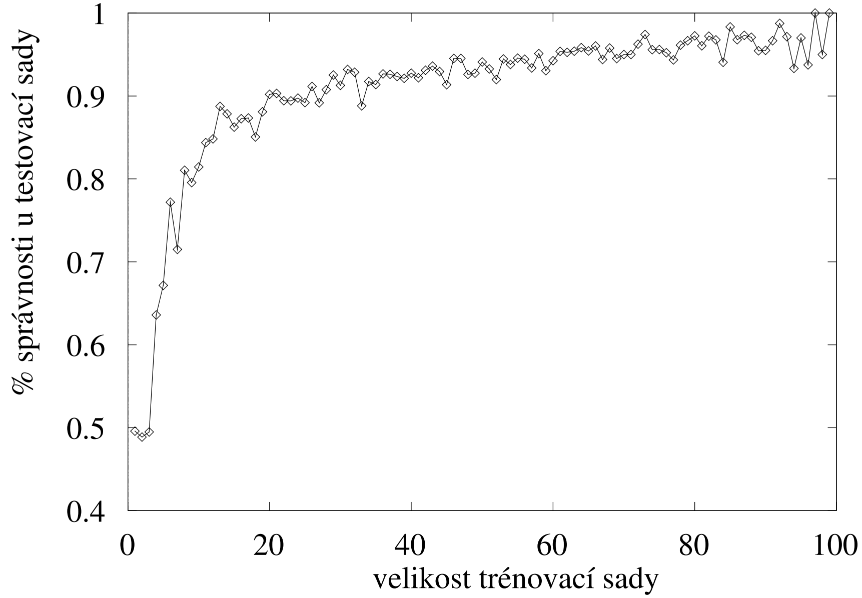
jak můžeme zjistit, zda $h \approx f$?

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

používaná metodologie:

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.

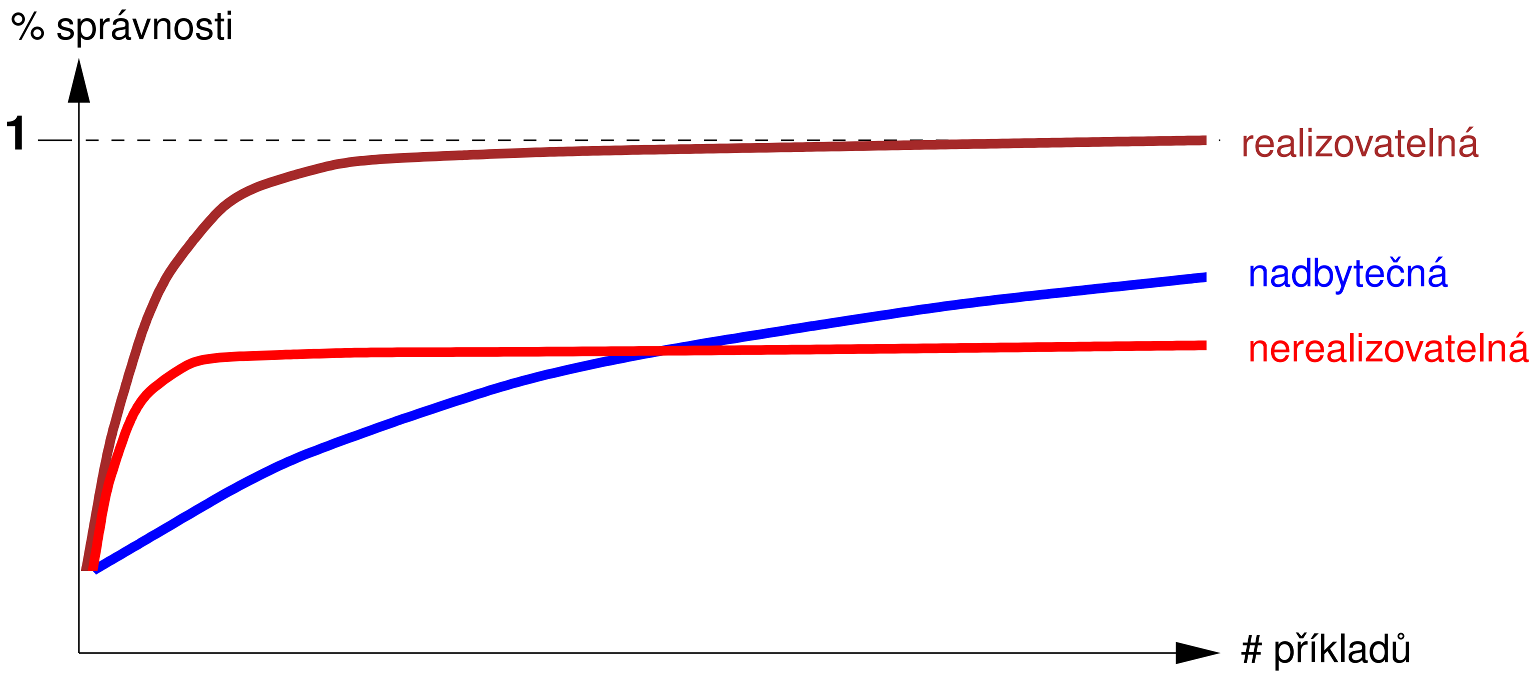
tvar křivky učení závisí na ➜ je hledaná funkce realizovatelná × nerealizovatelná

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

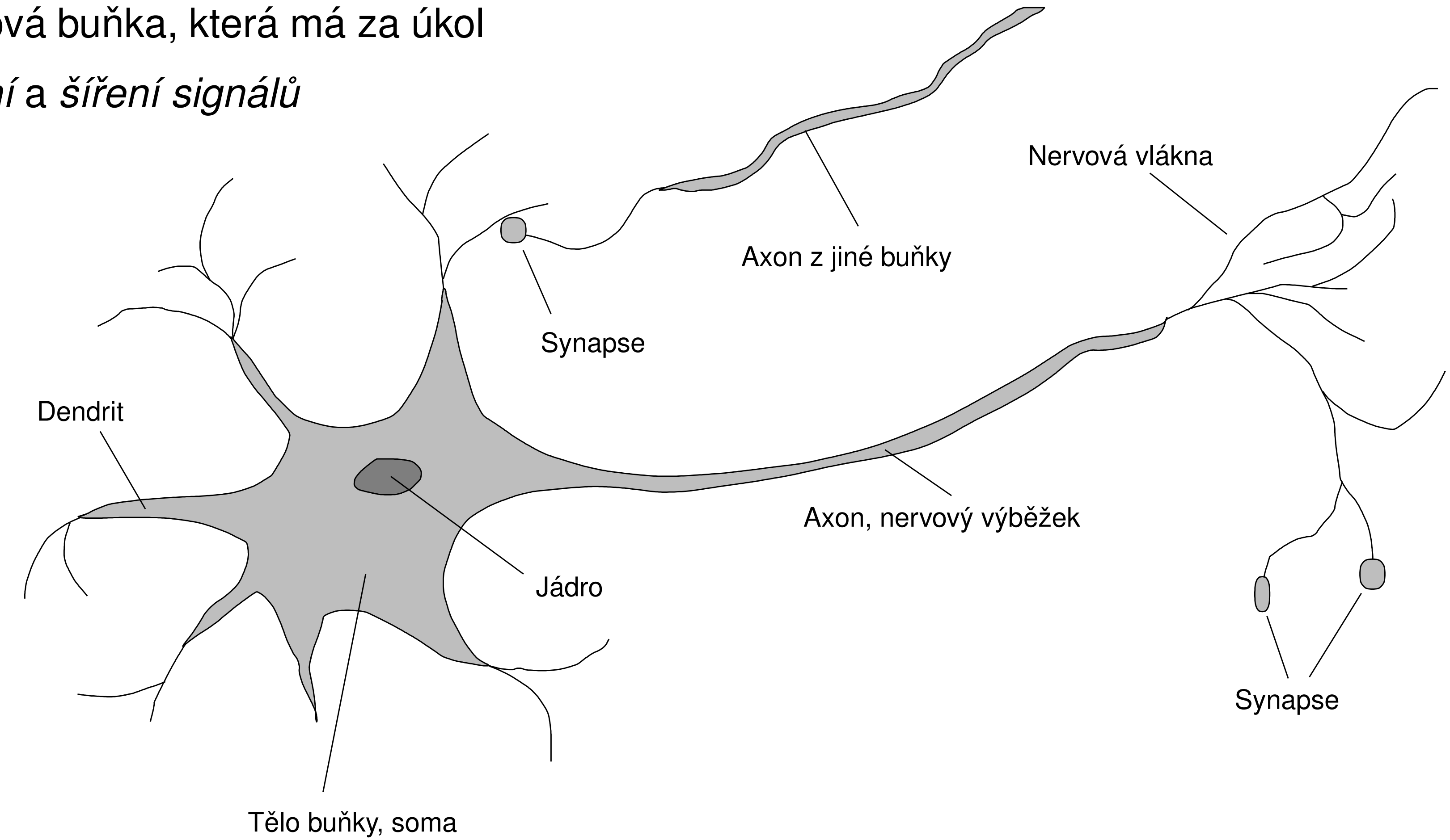
# Neuron

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*

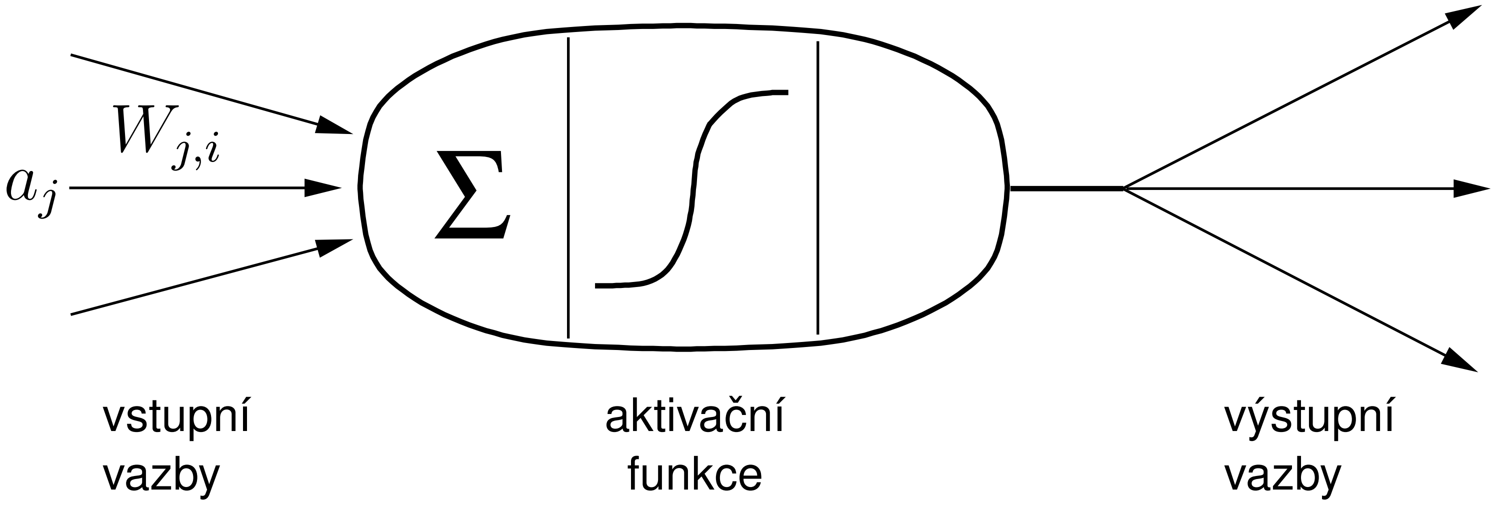


# POČÍTAČOVÝ MODEL – NEURONOVÉ SÍTĚ

1943 – McCulloch & Pitts – matematický model neuronu

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

jednotky (*units*) v neuronové síti
- 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)



# Počítačový model – neuronové sítě

1943 – McCulloch & Pitts – matematický model neuronu

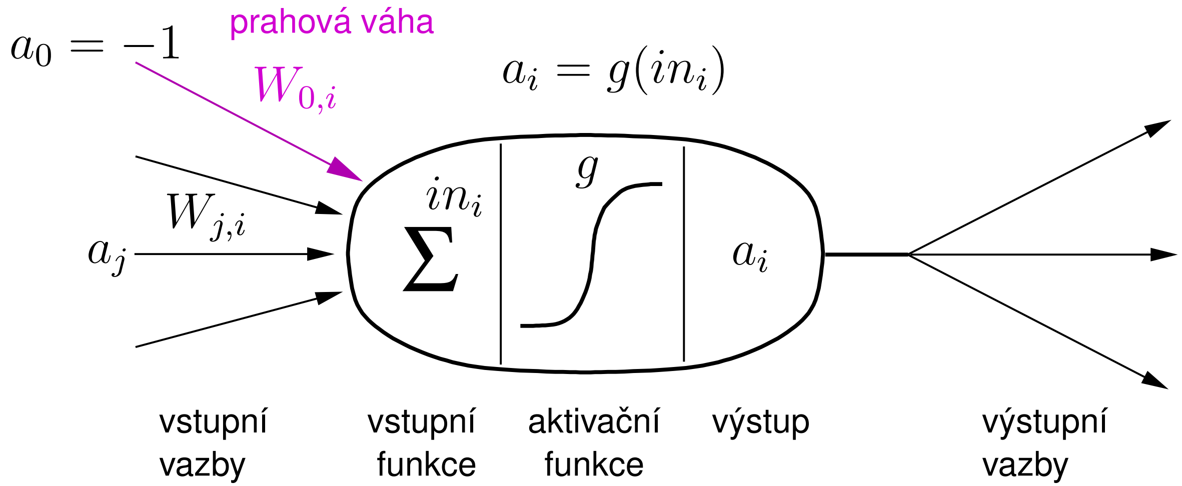
spojené do neuronové sítě – mají schopnost tolerovat šum ve vstupu a učit se

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

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



# AKTIVAČNÍ FUNKCE

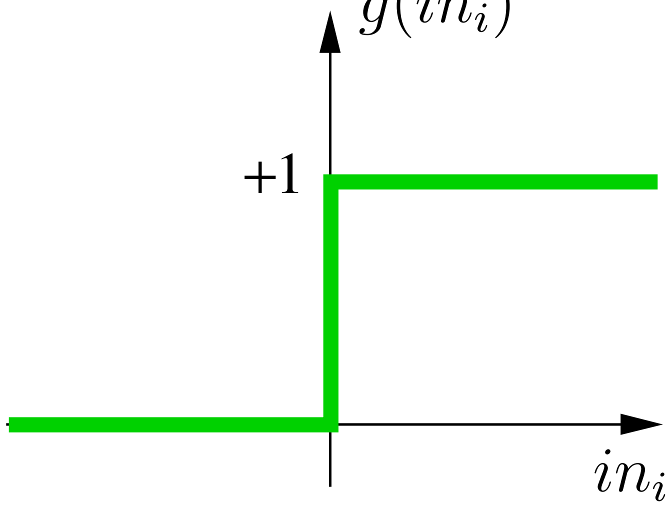
účel aktivační funkce = $\left\langle \begin{array}{l} \text{jednotka má být aktivní } (\approx +1) \text{ pro pozitivní příklady, jinak neaktivní } \approx 0 \\ \text{aktivace musí být nelineární, jinak by celá síť byla lineární} \end{array} \right.$

## AKTIVAČNÍ FUNKCE

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

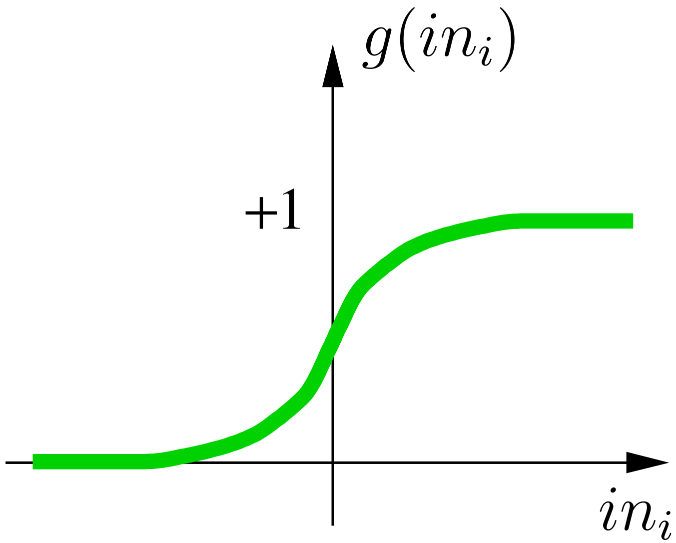
např.

a)




prahová funkce

b)

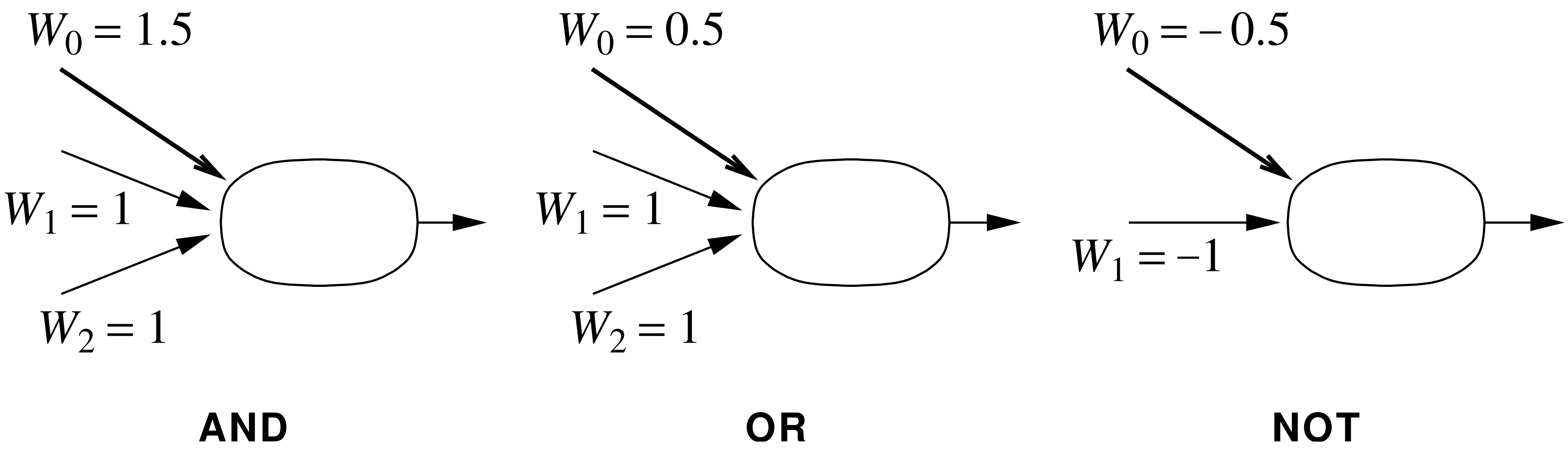



sigmoida $\quad 1/(1+e^{-x})$

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

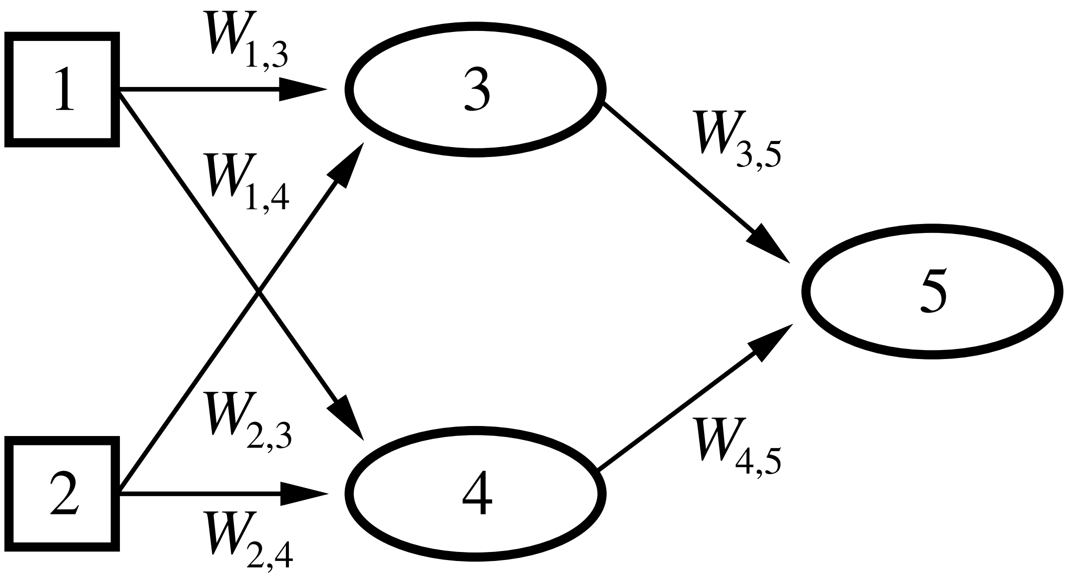
$\Rightarrow$ kombinacemi jednotek do sítě můžeme implementovat libovolnou Booleovskou funkci

## 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

## Příklad sítě s předním vstupem

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



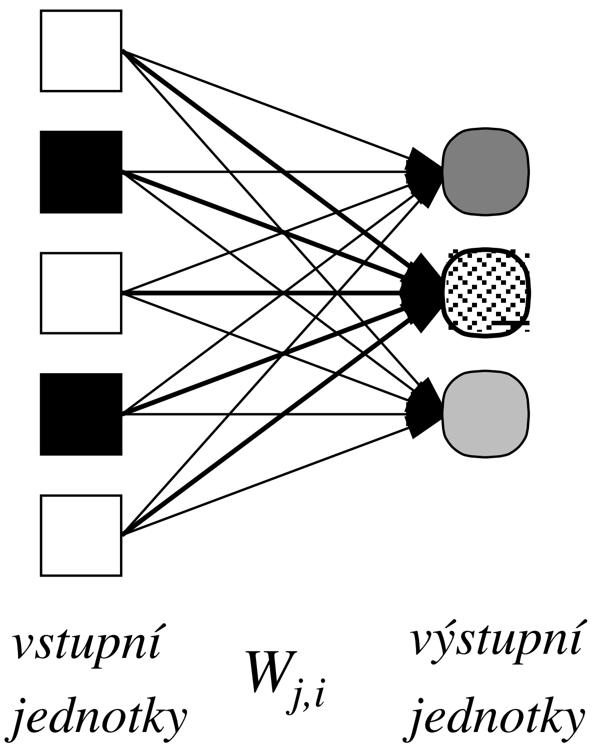

síť s předním vstupem = parametrizovaná nelineární funkce vstupu

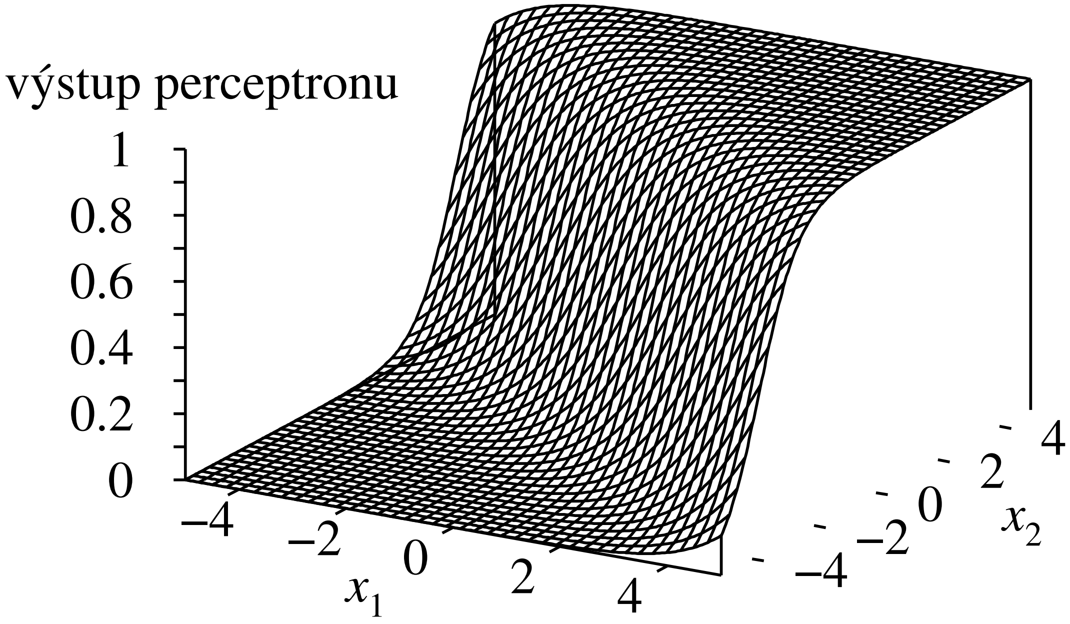
$$\begin{aligned} a_5 &= g(W_{3,5} \cdot a_3 + W_{4,5} \cdot a_4) \\ &= g\big(W_{3,5} \cdot g(W_{1,3} \cdot a_1 + W_{2,3} \cdot a_2) + W_{4,5} \cdot g(W_{1,4} \cdot a_1 + W_{2,4} \cdot a_2)\big) \end{aligned}$$

## Jednovrstvá síť – perceptron

perceptron – pro Booleovskou funkci 1 výstupní jednotka

– pro složitější klasifikaci – více výstupních jednotek



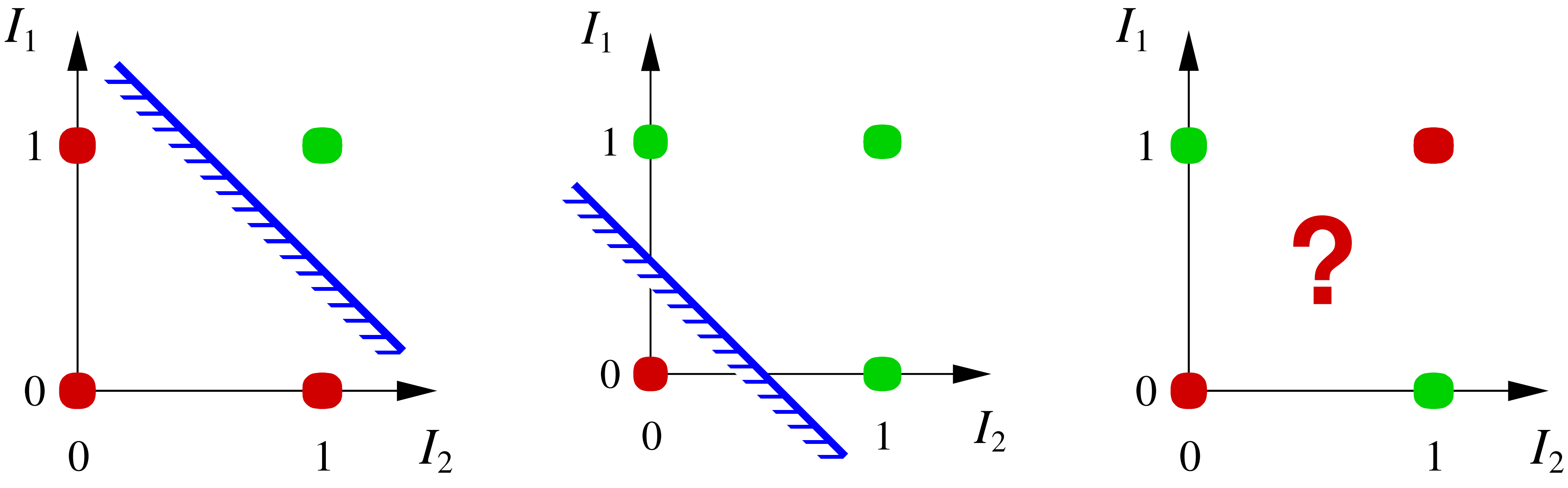
# Vyjadřovací síla perceptronu

předpokládejme perceptron s $g$ zvolenou jako prahová funkce ( )

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 tak, aby se snížila chyba na trénovací sadě

## UČENÍ PERCEPTRONU

výhoda perceptronu – existuje jednoduchý učící algoritmus pro libovolnou lineárně separabilní funkci

učení perceptronu = upravování vah tak, aby se snížila chyba na trénovací sadě

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

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

# Učení perceptronu

výhoda perceptronu – existuje jednoduchý učící algoritmus pro libovolnou lineárně separabilní funkci

učení perceptronu = upravování vah tak, aby se snížila chyba na trénovací sadě

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

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

váhy pro minimální chybu pak hledáme optimalizačním prohledáváním spojitého prostoru vah

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

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

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

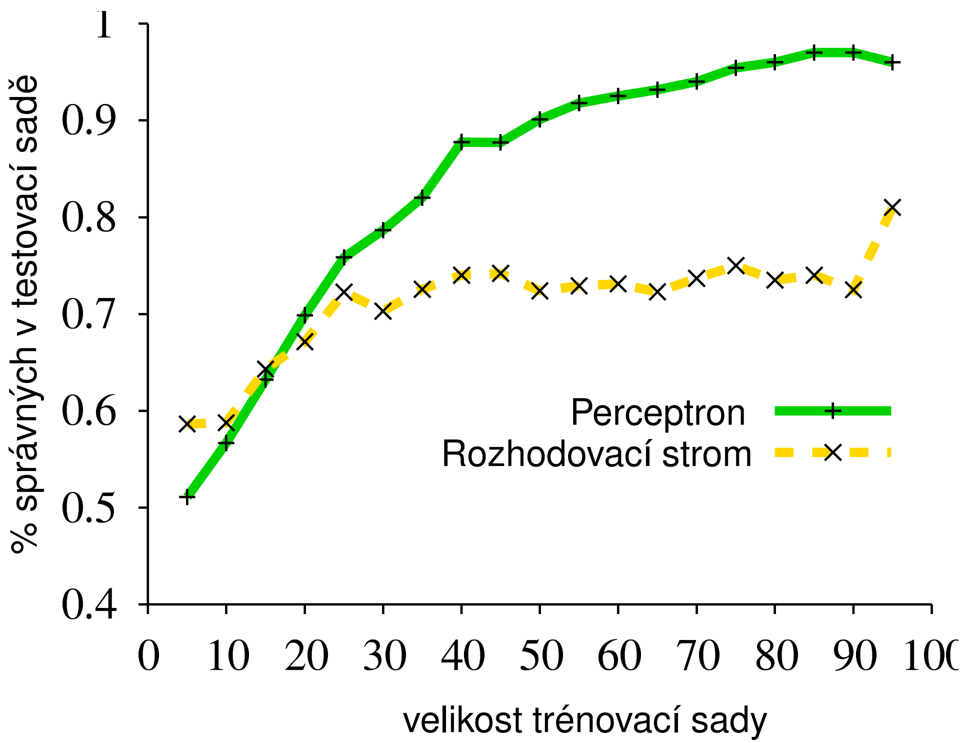
$\Rightarrow$ váhy se musí *zvýšit* pro pozitivní příklady a *snížit* pro negativní

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

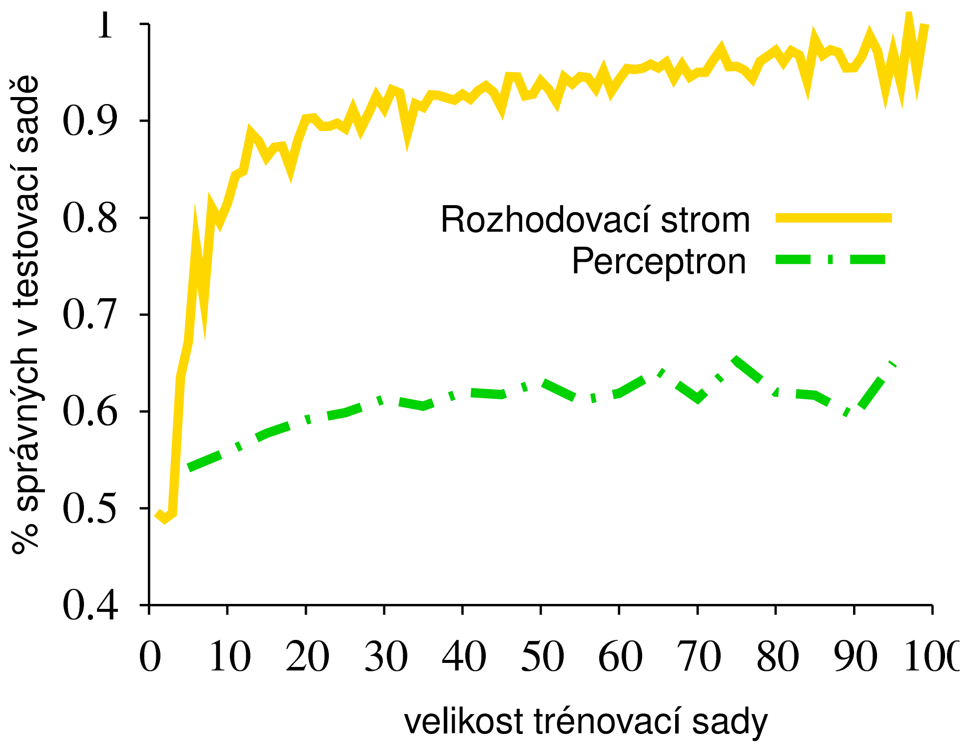
# UČENÍ PERCEPTRONU pokrač.

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

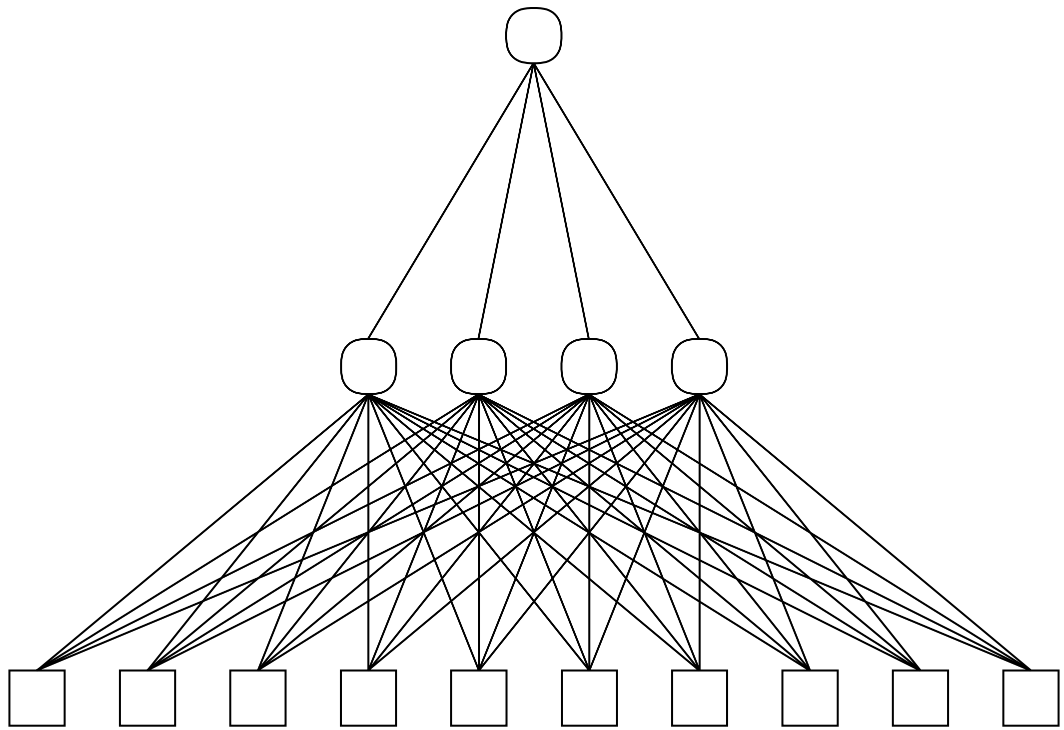
výstupní jednotky $a_i$

$W_{j,i}$

skryté jednotky $a_j$

$W_{k,j}$

vstupní jednotky $a_k$



# Vyjadřovací síla vícevrstvých sítí

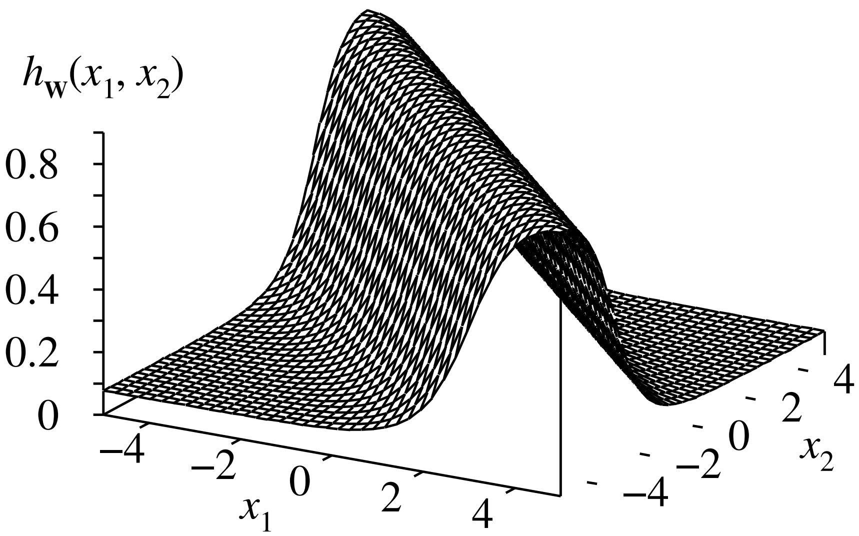
s jednou skrytou vrstvou – všechny spojité funkce

se dvěma skrytými vrstvami – všechny funkce

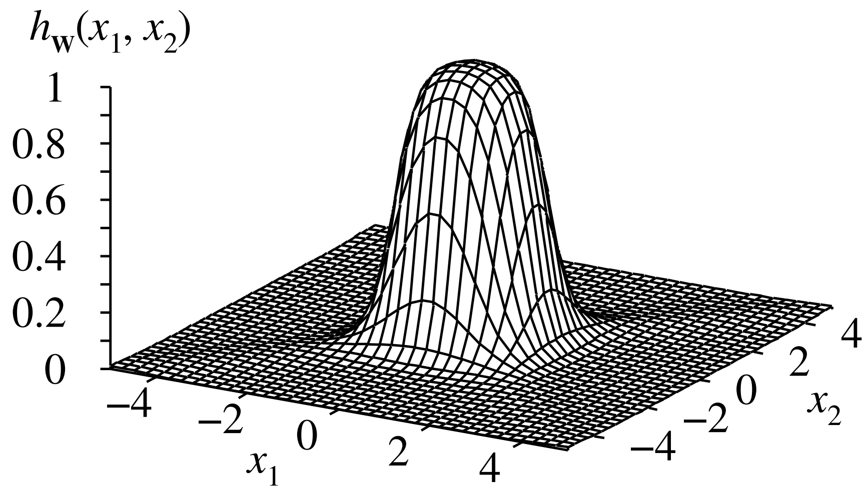
těžko se ovšem pro konkrétní síť zjišťuje její prostor reprezentovatelných funkcí

např.

dvě "opačné" skryté jednotky vytvoří *hřbet*




dva hřbety vytvoří *homoli*



# Učení vícevrstvých sítí

pravidla pro úpravu vah:

- **výstupní vrstva** – stejně jako u perceptronu

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

## UČENÍ VÍCEVRSTVÝCH SÍTÍ

pravidla pro úpravu vah:

- **výstupní vrstva** – stejně jako u perceptronu

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

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

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

# Učení vícevrstvých sítí

pravidla pro úpravu vah:

- **výstupní vrstva** – stejně jako u perceptronu

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

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

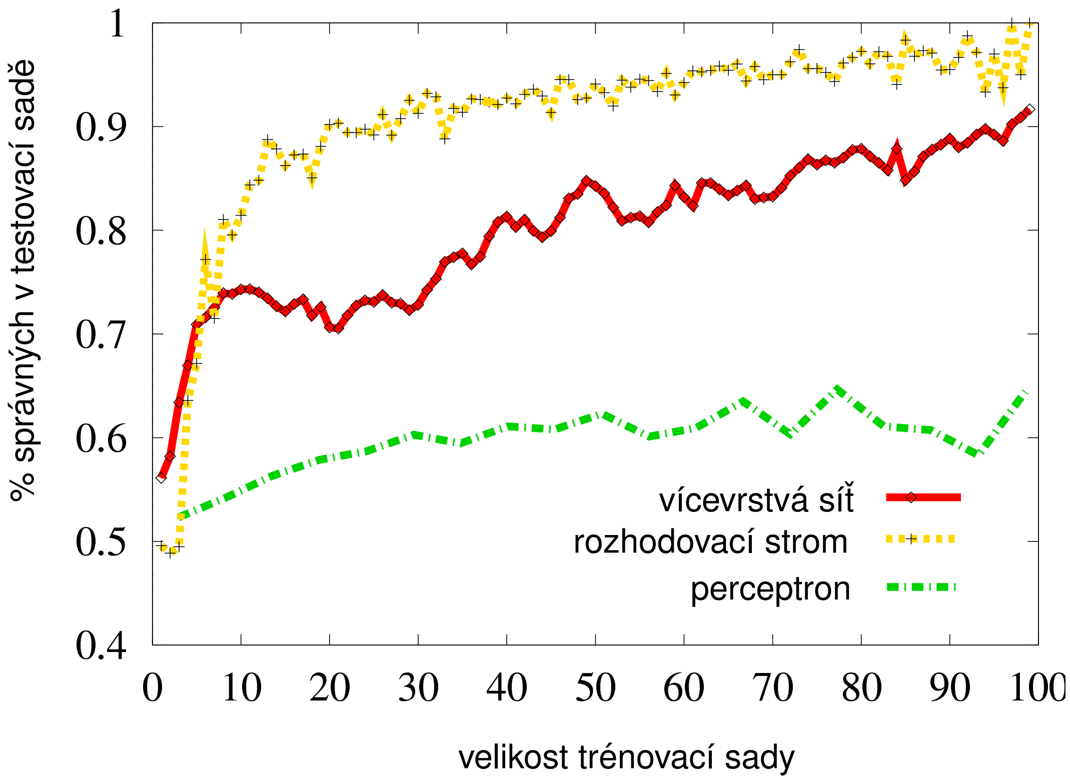
$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j \qquad \text{kde} \quad \Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i$$

problémy učení:

- dosažení lokálního minima chyby
- příliš pomalá konvergence
- přílišné upnutí na příklady $\longrightarrow$ neschopnost generalizovat

## UČENÍ VÍCEVRSTVÝCH SÍTÍ pokrač.

vícevrstvá síť se problém čekání na volný stůl v restauraci učí znatelně líp než perceptron



## NEURONOVÉ SÍTĚ – SHRNUTÍ

- většina mozků má velké množství neuronů; každý neuron $\approx$ lineární prahová jednotka (?)
- perceptrony (jednovrstvé sítě) mají nízkou vyjadřovací sílu
- vícevrstvé sítě jsou dostatečně silné; mohou být trénovány pomocí zpětného šíření chyby
- velké množství reálných aplikací
  - rozpoznávání řeči
  - řízení auta
  - rozpoznávání ručně psaného písma
  - ...