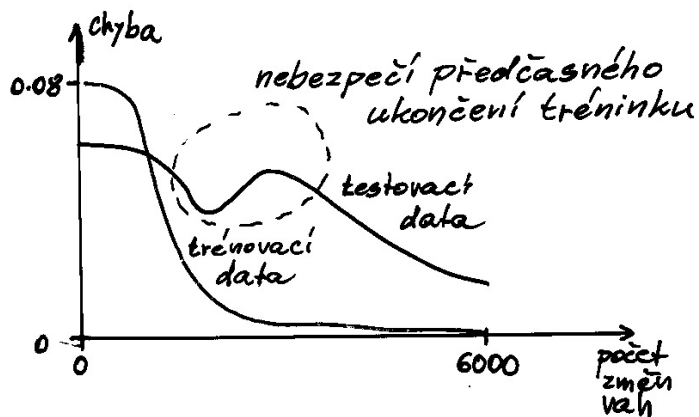
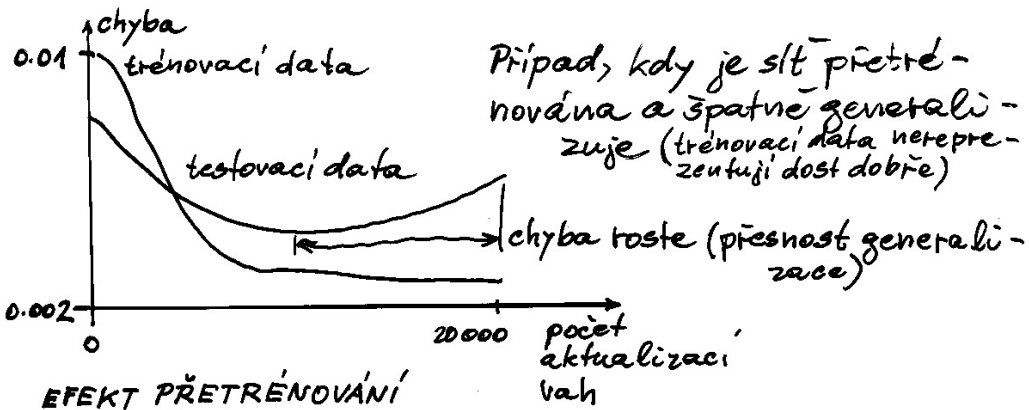


Generalizace, přetřénování, zastavení

Otázka ukončení tréninku není snadná. Zřejmě řešení je trénovat tak dlouho, až chyba E klesne pod nějaký předem stanovený práh. Tato strategie není příliš dobrá, protože NN s BP jsou citlivé na přetřénování vůči trénovacím příkladům (snížená schopnost generalizace při zpracování „neviděných“ příkladů).



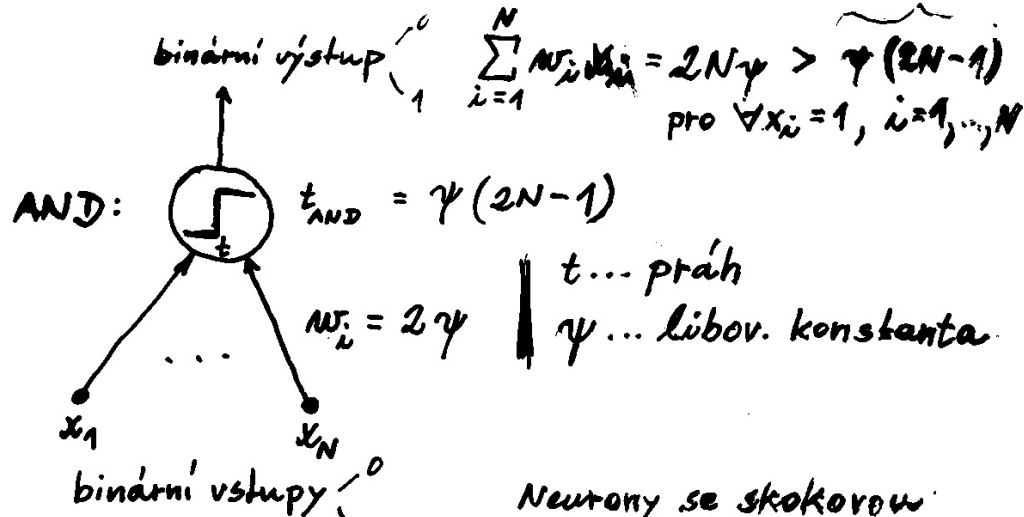
Obě situace pocházejí z dat získaných při učení robotu percepci

Na počátku jsou váhy inicializovány malými hodnotami (a navzájem si blízkými). Proto jsou rozhodovací plochy malé. S postupujícím trénováním některé z vah rostou (aby se zmenšilo E) a zvyšuje se složitost rozhodovací plochy. Pro dosti velký počet iterací je BP schopen vytvořit velmi (přesptilý) komplikovanou plochu, která velmi dobře „přilne“ i k datům s hodnotami ovlivněnými šumem, tj. přizpůsobí se charakteristikám, které nejsou pro data reprezentativní a vyskytují se jen u trénovacích příkladech.

Existuje několik technik, jak zacházet s problémem přetřénování:

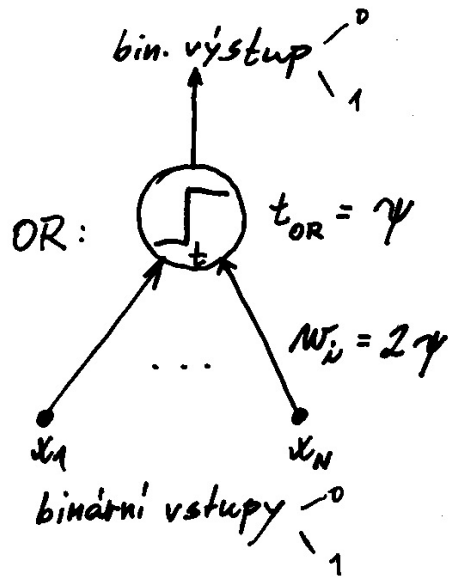
- tzv. ^{snížení} ~~snížení~~ vah: během každé iterace se váhy sníží o malý faktor, což odpovídá modifikaci definice E o přičlenění „pokuty“ odpovídající celkové velikosti vah v síti. Motivací je udržet hodnoty vah malé (tj. zaměřeno proti vývoji složitých rozhodovacích hypetploch).
- úspěšnou metodou je poskytnout algoritmu k trénovacím datům navíc i část testovacích. Sleduje se chyba vzhledem k testovacím datům, přičemž trénovací data řídí gradientní sestup. Potom je počet iterací dán nejmenší chybou vzhledem k testovacím datům. Používají se 2 kopie sítě (pro trénování a pro úschovu dosud nejlepší vzhledem k testovacím datům).

Inicializace umělých NN pomocí rozhodovacích stromů



Neurony se skokovou prahovou funkcí mohou přímo modelovat elementární logické operace.

Negace: pomocí $w < 0$



Schopnost NN učovat koncepty je silnější než v případě tradiční logiky.

Např. necht $t = \psi(2M-1)$, $M \leq N$, poskytuje možnost modelovat tzv. koncepty $M \geq N$ (výstup = 1 kdykoliv alespoň M z N vstupů jsou = 1).

Význam jednotlivých vlastností (atributů) lze navíc učít velikostí vahy:

$w_1 = 1, w_2 = 1, w_3 = 3, t = 1.5$

Neuron bude aktivován nejen kombinovanou přítomností x_1 a x_2 , ale také pouhým x_3 i když x_1 a x_2 nebudou přítomny (= 0).

V realitě jsou $x_i \in \mathbb{R}$, tj. jeden z důsledků je, že vstup s nízkou vahou a vysokou hodnotou může mít silnější efekt než vstup s vysokou vahou a malou hodnotou.

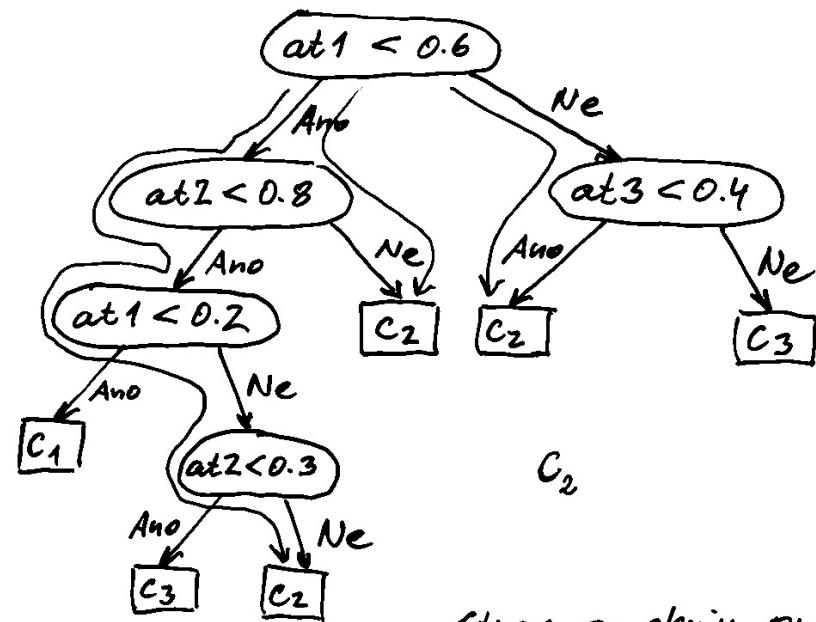
Organizaci neuronů do sítě a náhradou binárních prahů diferencovatelnou funkcí (např. sigmoidou) umožňuje modelovat jakoukoliv tozumnou funkci, která se může v reálných aplikacích vyskytnout.

Strom lze (různými způsoby) mapovat do struktury NN a tím výrazně zvýšit klasifikační schopnost (po natrénování pomocí BP). Jednou možností, jak převést strom na NN, je následující postup:

1. Za použití podmnožiny příkladů, * které jsou k dispozici, indukuj rozhodovací strom. (ID3, C4.5)
2. Přepiš strom do formy pravidel a pravidla (AND-OR pravidla) převed' na odpovídající síťovou strukturu, přičemž existující spoje se nazývají jako regulární. (Váhy zatím nejsou stanoveny a síť není plně propojena.)
3. Plně propoj soušední vrstvy (nové spoje se nazývají jako přídavné); přídavným spojům přiřad' malé počáteční hodnoty vah ϵ .
4. Stanov hodnoty vah regulárních spojů tak, aby síť aproximovala klasifikační schopnost původního stromu.
5. Mírně modifikuj hodnoty vah (všech), aby bylo umožněno následné učení.
6. Změň skokové prahové funkce všech neuronů na sigmoidy.
7. Natrénuj NN pomocí BP za použití všech trénovacích příkladů.

Struktura NN je mnohem bohatší než struktura stromová a proto NN umožňují mnohem "jemnější" zpracování konceptů.

U stromové struktury je každý koncept charakterizován disjunktívní normální formou (testy podél větve stromu jsou dány do konjunkce; popis konceptu je disjunkcí těch větví, které mají v listu totéž označení).

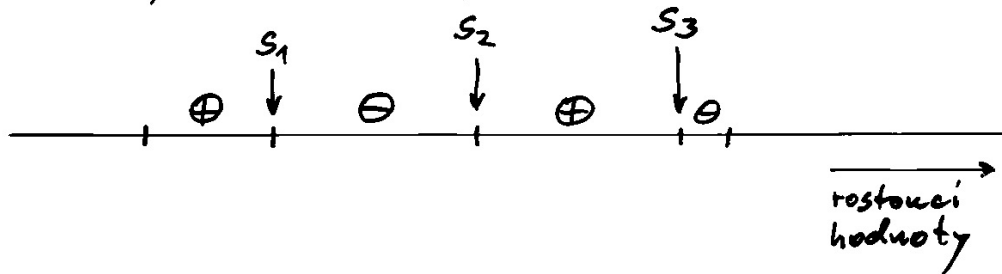


Strom a ekviv. pravidla:

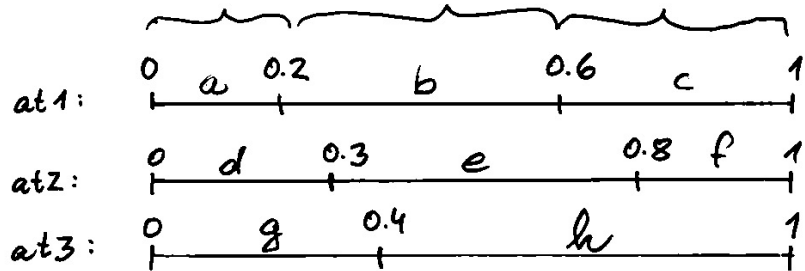
$$\begin{aligned}
 &(at1 < 0.6) \wedge (at2 < 0.8) \wedge (at1 < 0.2) \rightarrow C_1 \\
 &(at1 < 0.6) \wedge (at2 < 0.8) \wedge (at1 \geq 0.2) \wedge (at2 \geq 0.3) \vee \\
 &\quad (at1 < 0.6) \wedge (at2 \geq 0.8) \vee (at1 \geq 0.6) \wedge (at3 < 0.4) \rightarrow C_2 \\
 &(at1 < 0.6) \wedge (at2 < 0.8) \wedge (at1 \geq 0.2) \wedge (at2 < 0.3) \vee \\
 &\quad (at1 \geq 0.6) \wedge (at3 \geq 0.4) \rightarrow C_3
 \end{aligned}$$

V případě (dost častém) numerických dat se rozhodovací strom buduje tak, že číselné universonum atributu se rozdělí na intervaly.

Hodnoty atributů daných příkladů se seřadí:



Klasifikace na kladné příklady \oplus a záporné příklady \ominus rozdělí interval celý rozsah hodnot, na několik oblastí. Kandidáti na oddělovací hodnotu (s_1, s_2, s_3) leží na hranicích mezi oblastmi.



Každému intervalu se přidělí jiněno a dále se s ním zachází jako s Booleovskou proměnnou: hodnota atributu buď do intervalu patří nebo ne.

Např. atribut at_1 je rozdělen do následujících podintervalů: $[0, 0.2)$, $[0.2, 0.6)$, $[0.6, 1]$.

Pravidla obsahující redundanční testy se zjednoduší, např. výraz

$$(at_1 < 0.6) \wedge (at_2 < 0.8) \wedge (at_1 < 0.2)$$

$$\downarrow$$

$$(at_1 < 0.2) \wedge (at_2 < 0.8)$$

tedy

$$(a \wedge \neg f) \vee (c \wedge g)$$

Přepsaná pravidla:

$$(\neg c \wedge \neg f \wedge a) \rightarrow c_1$$

$$(\neg c \wedge \neg f \wedge \neg a \wedge \neg d) \vee (\neg c \wedge f) \vee (c \wedge g) \rightarrow c_2$$

$$(\neg c \wedge \neg f \wedge \neg a \wedge d) \vee (c \wedge h) \rightarrow c_3$$

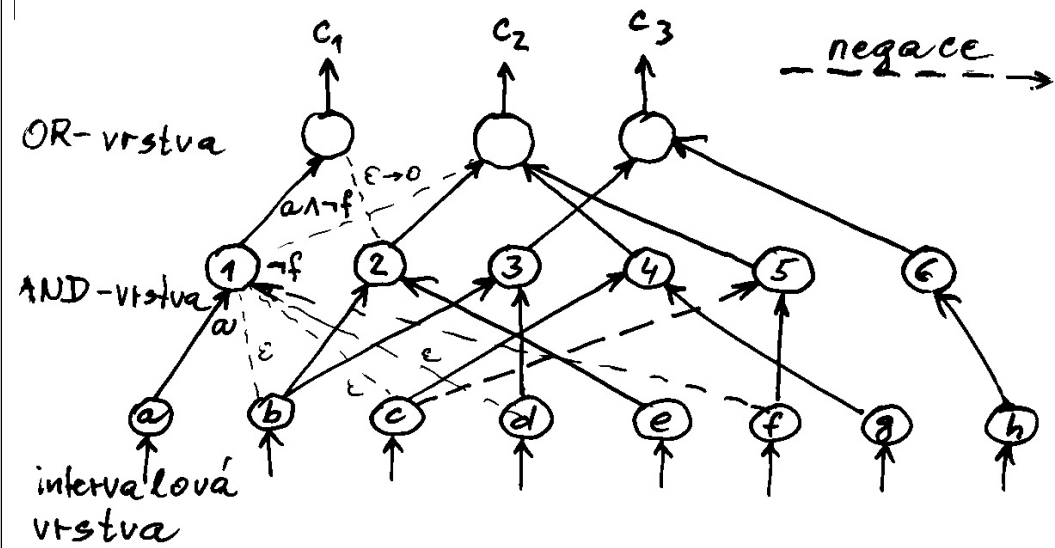
\downarrow zjedn.

$$(a \wedge \neg f) \rightarrow c_1$$

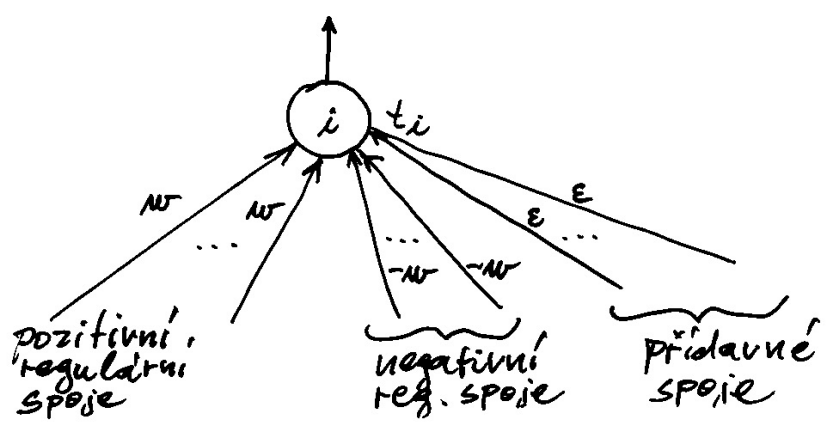
$$(b \wedge e) \vee (\neg c \wedge f) \vee (c \wedge g) \rightarrow c_2$$

$$(b \wedge d) \vee (c \wedge h) \rightarrow c_3$$

Odpovídající NN :



Pozn.: vstupy nejsou tytéž jako u stromu (tři atributy jsou převedeny na 8 intervalů).
 Počet uzlů skryté (AND) vrstvy je $T+1$, kde T je počet neterminálních uzlů stromu (obecně).
 OR-vrstva obsahuje 1 uzel na koncept.



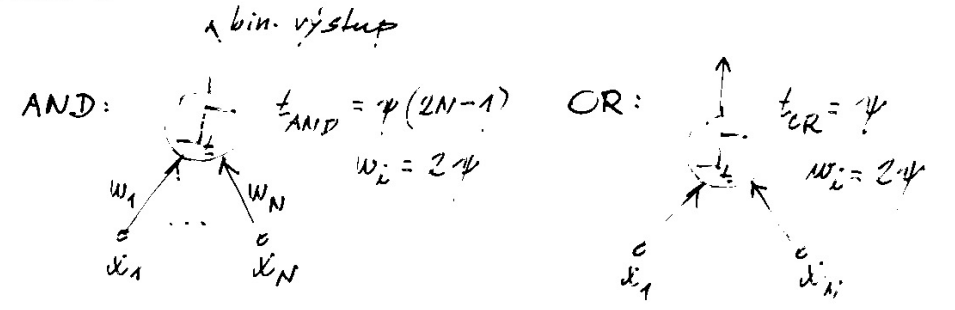
Po transformaci strom \rightarrow neuronová síť je nutno změnit přenosové funkce neuronů z původních skokových práhů na nelineární sigmoidy, aby bylo umožněno následné "vyladění" sítě pokračováním tréninku (který původně dal vznik rozhodovacímu stromu indukivní metodou, např. ID3):

$$f_i(x_i) = \frac{1}{1 + e^{-\beta x_i}}$$

kde x_i označuje i -tý vstup neuronů a β je nastavitelný parametr (strmost sigmoidy).

Klasifikační fáze je založena na tzv. strategii "vítěz bere vše". Znamená to, že předložený příklad je klasifikován jako instance i -tého konceptu, kde i je index OR-neuronu, jehož výstup má nejvyšší hodnotu.

V nejjednodušší verzi jsou logické funkce konjunkce a disjunkce modelovány tak, jak bylo ukázáno na obrázku:



Ponechání tohoto stavu by ovšem znamenalo, že neuronová síť by pouze emulovala původní rozhodovací strom.

Aby se využily mnohem bohatší možnosti síťové struktury, je nutno přidat další stupně volnosti. Toho lze docílit doplněním spojů tak, aby vzniklo úplně propojení, dále náhodnou změnou vah (dosud majících stejné hodnoty) a trénováním za použití gradientní sestupné metody.

Pro trénování je vhodná metoda BP (zpětné šíření chyb) doplněná o tzv. momentum:

$$w_{t+1} = w_t + \eta \cdot \Delta \cdot x + \mu (w_t - w_{t-1})$$

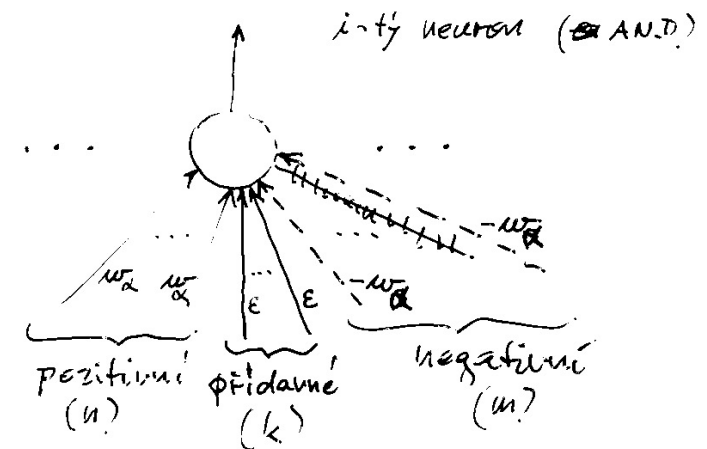
kde w_t je hodnota váhy v okamžiku (kroku) t , Δ je hodnota „zpětné“ chyby, x je předchozí hodnota výstupu neuronu, η je tzv. učicí konstanta (parametr), a μ je tzv. momentum.

Aby se předešlo přetřénování sítě, lze použít jednoduchý mechanismus: trénovací příklady se rozdělí na dvě části; jedna část slouží pro trénování, a druhá pro on-line testování. Po každém cyklu přes trénovací příklady (po každé epoše) se ověří výkonnost systému pomocí testovací podmnožiny dat. Trénování se pak zastaví, jakmile se ~~te~~ klasifikační přesnost odchýlí od již dosažené nejlepší hodnoty zpět k hodnotě nižší (přestože střední kvadratická chyba trénovacího souboru stále ještě může klesat).

Pozn.: doplněné přídavné spoje umožňují neuronové síti při trénování detekovat eventuelní relace mezi atributy, které neobjeví induktivní metoda použitá pro konstrukci původního rozhodovacího stromu (vztahy nejsou stromem zachyceny).

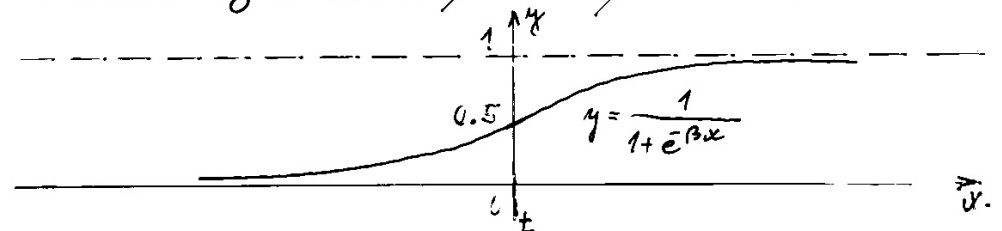
Poznámka ke stanovení hodnot vah vytvořené sítě:

Přídavnými spoji se nastavení prahů a vah zkomplikuje. Uvažme příklad i -tého neuronu s n pozitivními a m negativními regulačními spoji. Předpokládejme, že bude přidáno k přídavných spojů k vytvoření topologie úplně propojené sítě:



Pozitivní váhy jsou nastavovány na $w_{i\alpha}$, negativní na $-w_{i\alpha}$, (tj. pro i -tý AND-neuron), a na $w_{i\beta}$ resp. $-w_{i\beta}$ pro OR-neuron. Všechny váhy přídavných spojů se nastaví na nějakou malou hodnotu ϵ . Dále necht prahy příslušných neuronů mají hodnoty $t_{i\alpha}$ a $t_{i\beta}$.

Pro konečný rozsah hodnot vstupů (v praxi vždy) nedosáhne sigmoida nikdy hodnoty 0 nebo 1:



Proto je nutné pojem aktivovaný stav neuronu stanovit pomocí nějakých prahů. Obvyklá možnost je:

$$f(x) > A, \quad A \in (0.5, 1]$$

kde x je suma váhovaných vstupů neuronu a A je uživatelem stanovená konstanta. Obdobně lze neuron prohlásit za neaktivovaný, pokud $f(x) < 1-A$. Ostatní stavy se považují za nedefinované.

Neuron bude aktivní, pokud součet jeho vstupů překročí kritickou hranici $x = \psi$. Řešením rovnice

$$A = \frac{1}{1 + e^{\beta\psi}} \quad \text{pro } \psi \text{ dostaneme výraz:}$$

$$\psi = \frac{1}{\beta} \ln \frac{A}{1-A}$$

Díky symetričnosti sigmoidální funkce lze tedy říci, že neuron bude neaktivní pokud váhovaný součet jeho vstupů bude menší než $-\psi$.

Poznámka k úplnému propojení vrstvy AND s OR

Všechny vstupy do OR-neuronu jsou pozitivní, proto $m = 0$ (vlastnost plyne z logiky zakódované do rozhodovacího stromu - výstupy z uzlu AND nejsou negativní).

Md-li vrstva OR modelovat originální disjunkci listů stromu po zavedení přidavných vah, pak musí být splněny následující podmínky:

1) i -tý OR-neuron musí být aktivní za těchto podmínek, pro něž původní strom poskytoval příslušnou klasifikaci. Znamená to, že jeho vstup, snížený o hodnotu $t_{i\beta}$ musí být větší než ψ pokud alespoň jeden z AND-neuronů je vzhledem k regulačním spojmům aktivní (tj. poskytuje jako vstup do OR-neuronu hodnotu $1 \cdot A \cdot w_{i\beta}$), i když všechny ostatní neurony vrstvy AND poskytují nulový vstup (tj. $(n-1) \cdot 0 \cdot w_{i\beta}$) a dále i když všechny přidavné spoje poskytují nejvyšší negativní vstupy (tj. $-k \cdot \epsilon$).

2) i -tý OR-neuron musí být neaktivní za těchto podmínek, pro něž původní strom poskytoval příslušnou klasifikaci. Znamená to, že jeho vstup, snížený o práh $t_{i\beta}$ musí být menší než $-\psi$ i když všechny AND-neurony vzhledem k regulačním spojmům poskytují nejvyšší možné hodnoty, pro něž jsou ještě stále považovány za neaktivní (tj. $n \cdot (1-A) \cdot w_{i\beta}$), přestože také přidavné spoje poskytují nejvyšší hodnoty pozitivních vstupů (tj. $k \cdot \epsilon$).

Oba požadavky lze sumarizovat formou nerovnosti:

$$1 \cdot A \cdot w_{i\beta} + (n-1) \cdot 0 \cdot w_{i\beta} - k \cdot \epsilon - t_{i\beta} \geq \psi$$

$$n \cdot (1-A) \cdot w_{i\beta} + k \cdot \epsilon - t_{i\beta} \leq -\psi$$

Změnou obou nerovností na rovnice a jejich symbolickým řešením pro $w_{i\beta}$ a $t_{i\beta}$ dostaneme:

$$w_{i\beta} = \frac{2(\psi + k \cdot \epsilon)}{A \cdot (m+1) - m}$$

$$t_{i\beta} = (\psi + k \cdot \epsilon) \frac{n - A \cdot (m-1)}{A \cdot (m+1) - m}$$

(n je počet regulačních spojů ~~se zápornými~~ ^{s kladnými} váhami).

Poznámka k úplnému propojení vstupů a AND-vrstvy

Zde je navíc oproti předchozí úvaze komplikace, neboť počet záporných vah je obecně $m \geq 0$.

Má-li vrstva z AND-neuronů modelovat převodní konjunkce intervalů za přítomnosti přídavných spojů, musí být splněny následující tři podmínky:

- 1) i -tý AND-neuron musí být aktivní, pokud je příklad "šifru" podél odpovídající větve stromu. Znamená to, že jeho vstup zmenšený o práh $t_{i\alpha}$ musí být větší než ψ pokud všechny předchozí neurony na pozitivních regulačních spojích poskytují na výstupu nejmenší možnou hodnotu, pro niž jsou ještě považovány za aktivní (tj. $m \cdot A \cdot w_{i\alpha}$), i když všechny předchozí neurony na negativních regulačních spojích dávají na výstupu nejvyšší možnou hodnotu, pro niž jsou ještě stále považovány za neaktivní (tj. $m \cdot (1-A) \cdot (-w_{i\alpha})$) a i když všechny přídavné spoje poskytují max. neg. vstupy (tj. $-k \cdot \epsilon$).

- 2) i -tý AND-neuron musí být neaktivní, pokud nejmeně jeden pozitivní regulační spoj je neaktivní (tj. $1 \cdot (1-A) \cdot w_{i\alpha}$) i když všechny ostatní pozitivní regulační spoje na vstupu AND-neuronu poskytují nejvyšší možné hodnoty (tj. $(m-1) \cdot 1 \cdot w_{i\alpha}$), i když všechny neg. regul. vstupy jsou nulové (tj. $m \cdot 0 \cdot (-w_{i\alpha})$) a přídavné spoje poskytují nejvyšší pozitivní hodnoty vstupů (tj. $k \cdot \epsilon$).

- 3) i -tý AND-neuron musí být neaktivní, pokud nejmeně jeden negativní regul. vstup dává nejmenší hodnotu pro niž je považován za aktivní (tj. $1 \cdot A \cdot (-w_{i\alpha})$), přestože všechny regul. posit. vstupy jsou maximálně aktivní (tj. $m \cdot 1 \cdot w_{i\alpha}$), i když všechny ostatní neg. regul. vstupy jsou nulové (tj. $(m-1) \cdot 0 \cdot (-w_{i\alpha})$) a i když přídavné spoje poskytují maximální pozitivní vstupy ($k \cdot \epsilon$).

Uvedené tři požadavky lze sumarizovat takto:

$$m \cdot A \cdot w_{i\alpha} + m \cdot (1-A) \cdot (-w_{i\alpha}) - k \cdot \epsilon - t_{i\alpha} \geq \psi$$

$$1 \cdot (1-A) \cdot w_{i\alpha} + (m-1) \cdot 1 \cdot w_{i\alpha} + m \cdot 0 \cdot (-w_{i\alpha}) + k \cdot \epsilon - t_{i\alpha} \leq -\psi$$

$$m \cdot 1 \cdot w_{i\alpha} + 1 \cdot A \cdot (-w_{i\alpha}) + (m-1) \cdot 0 \cdot (-w_{i\alpha}) + k \cdot \epsilon - t_{i\alpha} \leq -\psi$$

Převodem na rovnosti a řešením pro $w_{i\alpha}$ a $t_{i\alpha}$ dostaneme (poslední dvě nerovnosti jsou identické):

$$w_{i\alpha} = \frac{2(\psi + k \cdot \epsilon)}{A \cdot (m+m+1) - (m+m)}$$

$$t_{i\alpha} = (\psi + k \cdot \epsilon) \frac{A(m+m-1) + (m-m)}{A(m+m+1) - (m+m)}$$

Závěr: nastavíme-li váhy a prahy na hodnoty uvedené v tabulkách pro $t_{i\alpha}$, $t_{i\beta}$, $w_{i\alpha}$, $w_{i\beta}$, pak síť emuluje chování původního stromu i za přítomnosti přídavných spojů.

Pozn.: Považujte si, že pro případ skokové přenosové funkce, $k=0$ (tj. žádné příd. spoje) a $A=1$ degenerují tabulky na vztahy $t_{AND} = \psi(2N-1)$ a $w_i = 2\psi$, resp. $t_{OR} = \psi$ a $w_i = 2\psi$.

Přítomnost přídavných spojů výrazně zvyšuje dimenzionalitu prohledávaného prostoru, což znamená, že je větší šance na nalezení výsledku.

Kromě toho, síť inicializovaná nikoliv náhodně, nýbrž pomocí stromu (váhy a prahy) umožňuje algoritmu BP začít z pozice, o které lze předpokládat, že je velmi blízká buď globálnímu či velmi dobrému lokálnímu minimu.

Poznámka k intervalům na vstupu (fuzifikace)

Rozhodovací strom předpokládá rozhodovací testy ve formě $x_i < t_j$, což implikuje binární intervalovou funkci příslušnosti (bodů do intervalu). Z toho plyne, že hodnota uprostřed intervalu a hodnota blízko hranice intervalu má zcela stejný význam. Tento přístup poněkud limituje flexibilitu sítě.

Je-li funkce příslušnosti "stupňovitá" (obdobně jako je tomu v teorii fuzzy množin), lze uvedené omezení obejít.

Fuzifikaci hranic intervalů lze realizovat následovně: nejvyšší hodnota a_i funkce příslušnosti je uprostřed intervalu, přičemž směrem k hranicím klesá - čím dále od středu, tím menší a_i ; nejvyšší pokles je přímo na hranici.

Blížkost vůči středu intervalu lze vyjádřit jako

$$r_i = \frac{R_i - 2|\mu_i - x_j|}{2R_i}$$

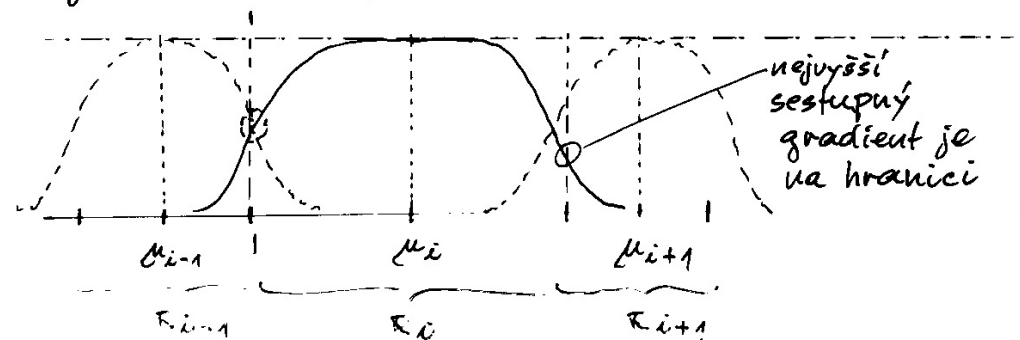
kde μ_i je střed i -tého intervalu, R_i je rozsah i -tého intervalu, x_j je skutečná hodnota j -tého atributu.

Blížkost r_i středu i -tého intervalu je dále parametrem sigmoidální funkce

$$a_i = \frac{1}{1 + e^{-\beta \cdot r_i}}$$

neuronů v intervalové vrstvě.

Tím je mj. umožněno např. to, aby hodnota atributu, která z nějakého důvodu padne mimo hranice (kupř. vlivem šumu), dostala "druhou šanci" ještě v následující vrstvě neuronů.



Radialní bázevé funkce (RBF)

Jedním z možných přístupů k aproximaci funkce je učení pomocí tzv. radialních bázeových funkcí (v ang. „radial basis functions“). Naučena hypotéza je zde představována jako funkce:

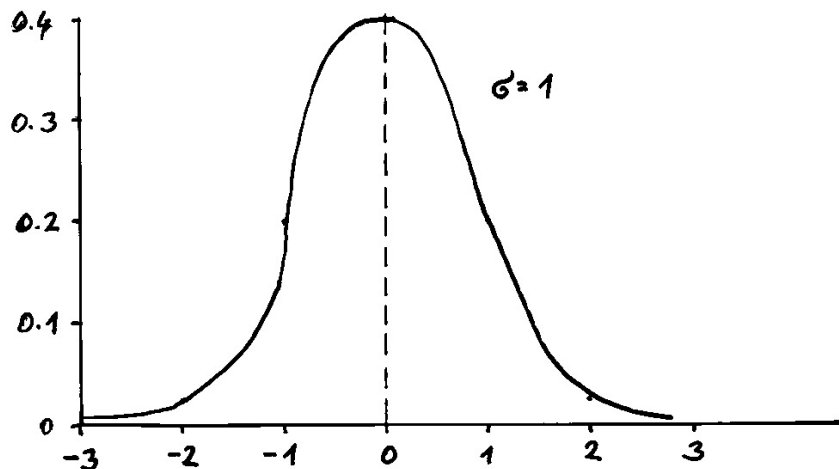
$$\hat{f}(x) = w_0 + \sum_{u=1}^k w_u K_u [d(x_u, x)]$$

kde každé $x_u \in X$ je instance z X a kde K_u je funkce definovaná tak, že se zvyšováním vzdálenosti $d(x_u, x)$ klesá. k je uživatelem poskytnutá konstanta, která specifikuje počet funkcí K_u . $\hat{f}(x)$ je globální aproximací nějaké $f(x)$, přičemž přínos každé $K_u [d(x_u, x)]$ je lokalizován na okolí bodu x_u .

Obvykle se jako K_u volí Gaussova funkce se středem „bode“ x_u a se zvoleným rozptylem σ_u^2 :

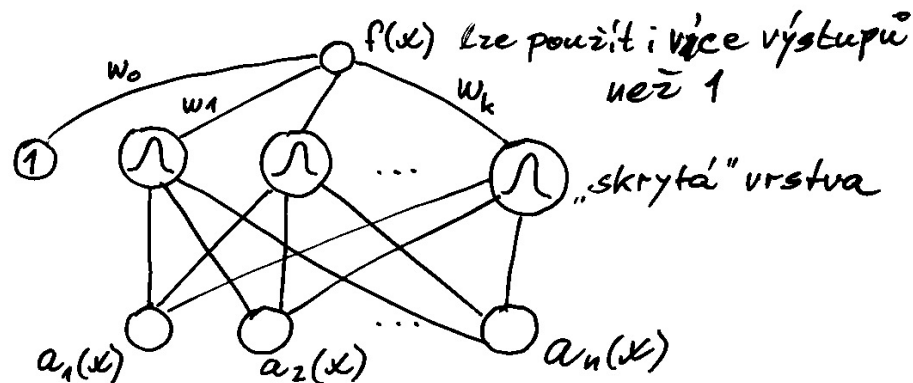
$$K_u [d(x_u, x)] = e^{-\frac{1}{2\sigma_u^2} d^2(x_u, x)}$$

σ je střední kvadratická odchylka či směrodatná odchylka).



Bylo ukázáno v Hartman et al. (1990), že uvedená funkcionální forma (pro $\hat{f}(x)$) je schopna aproximovat každou funkci s libovolně malou chybou za předpokl., že je dáno dostatečně velké k a že σ^2 pro každou funkci K_u může být specifikováno zvlášť.

Na uvedený vztah pro \hat{f} lze také hledět jako na popis dvojvrstvé sítě, kde první vrstva počítá hodnoty různých K_u a druhá vrstva počítá lineární kombinace těchto hodnot:



Sítě se učí pomocí zadané tréninkové množiny.

1. fáze: učí se k (tj. počet uzlů skryté vrstvy) a každý uzel j je definován polohou x_{uj} a σ_u^2 , tj. stanoví se funkce $K_u[d(x_u, x)]$.

2. fáze: trénováním se mění hodnoty vah w_u tak, aby výstup sítě co nejlépe odpovídal funkci $f(x)$, tj. požadovaným (správným) hodnotám $f(x)$ pro daná x . Používá se globální kritérium (chybová funkce):

$$E = \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2$$

kde D je množina trénovacích příkladů. (Cílem je minimalizovat globálně E .) Změny vah:

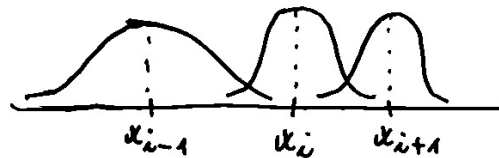
$$\Delta w_j = \eta \sum_{x \in D} (f(x) - \hat{f}(x)) a_j(x)$$

$a_j(x)$ je hodnota j -tého atributu (na vstupu sítě), η je tzv. konstanta učení.

Během 2. fáze se K_u nemění.

Alternativy:

- Gaussova funkce se přiřadí každé instanci trénovací množiny:



Lze též přiřadit každé funkci stejné σ^2 .

RBF se učí globální aproximací cílové funkce $f(x)$ tak, že každá trénovací instance ovlivní $f(x)$ pouze v okolí x_i . Tento způsob umožní RBF aproximovat pro trénovací příklady přesně, tj. pro lib. množinu m trénovacích příkladů lze vahy w_0, w_1, \dots, w_m pro kombinaci m funkcí K nastavit tak, že $\hat{f}(x_i) = f(x_i)$ pro $\forall(x_i, f(x_i))$.

- zvolí se menší počet K než je počet trénovacích instancí (tento přístup je efektivnější zejména pro rozsáhlé množiny trénovacích dat). Gaussovy funkce lze rozložit na univetsu X rovnoměrně i nerovnoměrně s ohledem na rozložení instancí $x_i \in X$.

RBF jsou lineární kombinací mnoha lokálních aproximací $f(x)$. Trénování je efektivnější než u obvyklých dopředných neuronových sítí trénovaných pomocí zpětného šíření chyby (BP).

RBF-sítě a BP-sítě

Obdobně jako umělé neuronové sítě trénované pomocí algoritmu BP, také RBF-sítě mají řadu úspěšných aplikací. Jedná se zejména o aproximace neznámých nelineárních funkčních závislostí a o klasifikátory.

V porovnání se sigmoidálními BP-sítěmi dosahují RBF-sítě na velmi obtížných problémech zcela stovnatelných výsledků, přičemž trénovací čas je o mnoho řádů nižší.

Na druhé straně však RBF sítě vyžadují typicky až desetinašobek potřebných trénovacích dat k dosažení téže přesnosti jako sigmoidální BP-sítě.

Pro velmi obtížné klasifikační problémy dosahují RBF (či jejich modifikace) obvykle lepších výsledků než BP za předpokladu dostatečného množství trénovacích dat a jednotek ve skryté vrstvě.

Kratší doba tréninku je u RBF dána tím, že na konkrétní vstupní vektor reaguje obvykle pouze malá část skrytých jednotek.

Topologie RBF je hybridní v tom smyslu, že výstupní vrstva je složena z "klasických" jednotek, zatímco skrytá vrstva z jednotek RBF, takže odpadá pomalé zpětné šíření chyb z výstupní vrstvy do skryté.

RBF pro aproximaci funkcí vyžaduje mnoho dat. U BP-sítí probíhá aproximace v globálním smyslu, zatímco u RBF v lokálním, což vede k lepší generalizaci u BP.

BP-sítě také využívají své stupně volnosti efektivněji, z čehož plyne ~~menší~~ potřeba nižšího počtu skrytých jednotek.

Je-li zapotřebí využívat neuronovou síť pro extrapolaci, pak je lepším kandidátem BP. "Lokální" vlastnosti RBF sítí jim zabráňují "vidět" mimo trénovací data.

Pro klasifikační úlohy mají RBF-sítě nízký počet chybných klasifikací, a to z těchto důvodů, pro něž jsou špatnými extrapolátory. Oblasti hodnot vstupních vektorů, které jsou vzdáleny hodnotám trénovacích vektorů, jsou u RBF mapovány do oblasti nízkých hodnot bárových funkcí skrytých RBF jednotek. U BP naopak sigmoidální jednotka může poskytnout na výstupu vysokou hodnotu i pro data velmi vzdálená od trénovacích, což má za následek klasifikaci s vysokou "důvěryhodností" i pro nevýznamné vstupy. (Toto lze u BP poněkud omezit speciálním trénováním za použití "hloupých" dat, tj. nesmyslných vstupních kombinací, ovšem pro mnohatozměrná data to vede k neúnosným časům tréninku.)

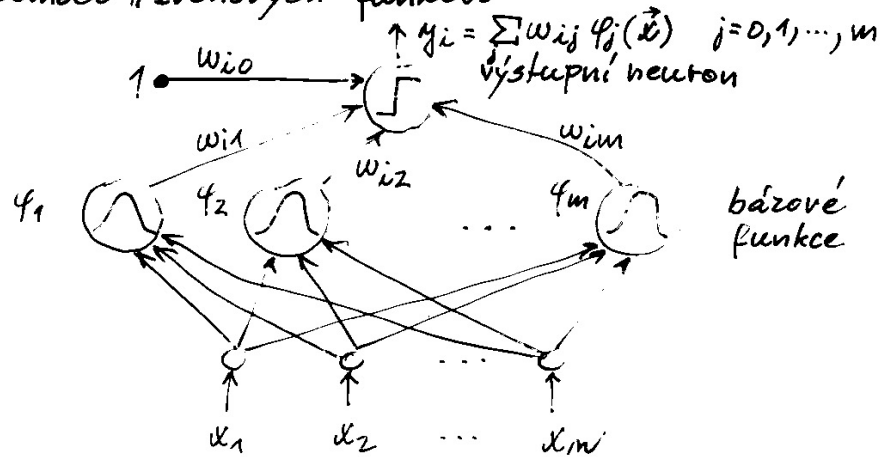
BP: lépe užít při problémech se získáním dostatečného počtu dat (neexistují, jsou drahá, ...).

RBF: velké množství levných dat, požadavek na on-line trénování (adaptivní zpracování signálů, adaptivní řízení, kdy data přicházejí vysokou rychlostí a nemohou být ukládána).

INICIALIZACE RBF-SÍŤI POMOCÍ ROZH. STRUMŮ

RBF-síť (síť s radiálními báзовými funkcemi) patří mezi často užívané klasifikátory.

Klasifikovaný příklad je předán souboru tzv. báзовých funkcí, z nichž každá vrací skalární hodnotu φ_j , $j=1, 2, \dots, m$. Teoretické a praktické výzkumy došly k závěru, že na konkrétním tratu RBF příliš nezáleží (není to kritické). Všeobecně se používají tzv. RBF (radiální BF) implementované pomocí "zvonových" funkcí:



Podstata spočívá v transformaci prostoru instancí tak, aby se zvýšila separabilita příkladů reprezentujících odlišné koncepty.

Předpokládejme, že příklady byly normalizovány a že standardní odchylka pro každý atribut je σ . Potom následující vztah popisuje n -rozměrný gaussovský poutch, se středem v $\vec{\mu}_j = [\mu_{j1}, \dots, \mu_{jn}]$:

$$\varphi_j(\vec{x}) = e^{-\frac{\|\vec{x} - \vec{\mu}_j\|^2}{2\sigma^2}}$$

$\varphi_j(\vec{x})$ je mírou podobnosti: čím větší je vzdálenost mezi \vec{x} a $\vec{\mu}_j$, tím menší je hodnota $\varphi_j(\vec{x})$.

Některé významné klíček:

(\vec{x} je vektor $[x_1, x_2, \dots, x_n]$ popisující jeden příklad).

Instanční prostor je část prostoru obsazená všemi možnými hodnotami příkladů. x_i je proměnná, často nazývaná jako "atribut".

Koncept je binární funkce: $c: \mathbb{R}^n \rightarrow \{-1, +1\}$, kde "+1" je označení pozitivního příkladu a "-1" negativního.

Pro více než 2 koncepty: $c: \mathbb{R}^n \rightarrow L$, kde L je množina označení konceptů.

Úkolem učícího se systému je nalézt funkci h vanou klasifikátor, $h: \mathbb{R}^n \rightarrow \{-1, +1\}$, která co nejlépe aproximuje koncept.

Má-li být nějaký příklad klasifikován, síť napřed přemění $\varphi(\vec{x})$ na $\{\varphi_1(\vec{x}), \varphi_2(\vec{x}), \dots, \varphi_m(\vec{x})\}$. Aktivace i -tého výstupního neuronu se pak vypočítá jako

$$y_i = \sum_{j=0}^m w_{ij} \varphi_j(\vec{x})$$

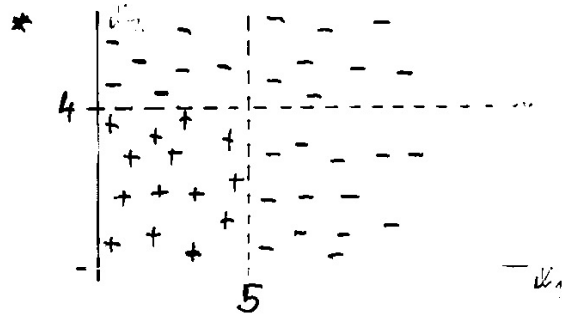
kde w_{ij} je váha spoje mezi j -tým neuronem skryté vrstvy a i -tým neuronem výstupní vrstvy (w_{i0} je jakoby trvale připojena na $\varphi_0 = 1$).

Příkladu je pak přiřazen i -tý koncept když

$$y_i = \max_k (y_k)$$

Nalezení vah w_{ij} pomocí např. Δ -pravidla obvykle nepředstavuje problém.

Obtížné může někdy být stanovení středu každé RBF resp. počtu RBF, dále stanovení d , stanovení nerelevantních atributů (RBF jsou svou podstatou blízké metodě nejbližšího souseda a jsou proto citlivé na irelevantní atributy*).



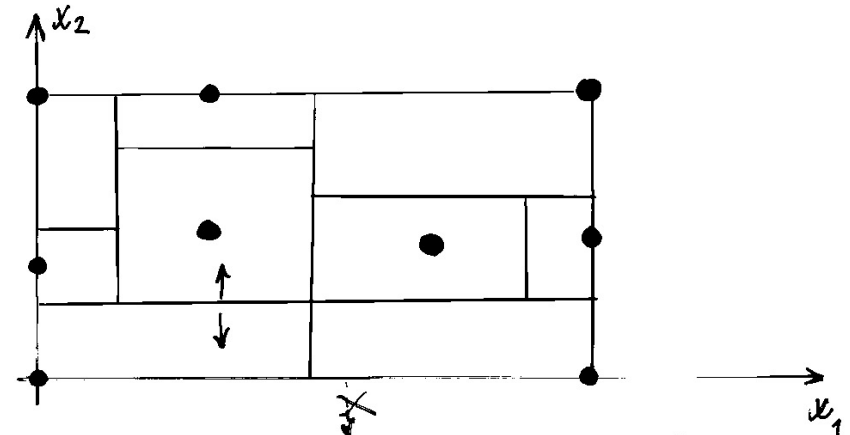
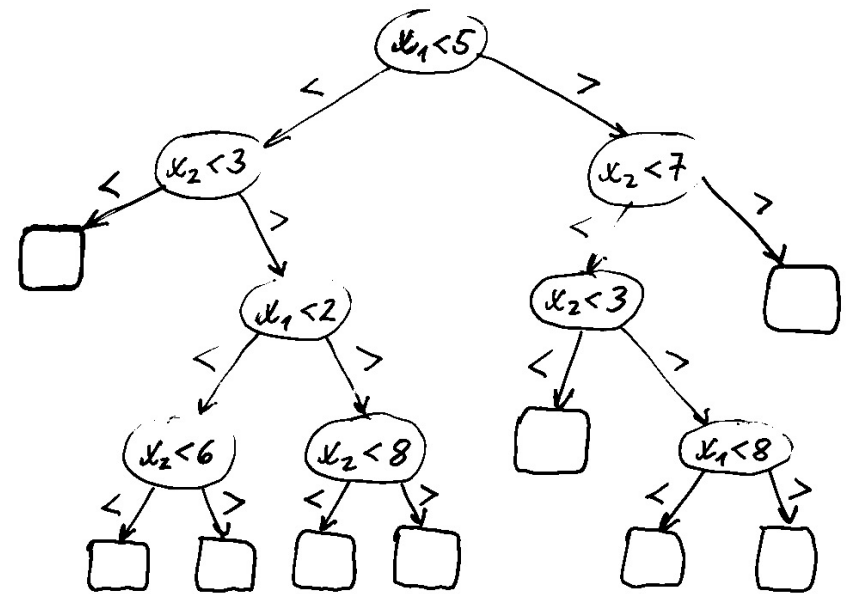
pro $x_1 < 5$ potřebujeme dva
 pro $x_1 > 5$ x_2 je irelevantní
 pro $x_2 > 4$ x_1 je irelevantní

Uvedené problémy může často napomoci vyřešit návrh RBF-sítě pomocí rozhodovacích stromů. Tímto postupem je dosaženo zároveň dvou důležitých cílů: usnadněný návrh RBF-sítě a zlepšené klasifikační schopnosti původního rozhodovacího stromu.

Inicializace RBF-neuronové sítě pomocí stromu

Systémy využívající indukci rozhodovacích stromů rozdělují instancovní prostor na oblasti po dvojicích disjunktivní. V ideálním případě pak každá oblast obsahuje příklady jediného konceptu. Indukování rozhodovacích stromů je řízeno heuristikami, které mají zajistit co nejmenší stromy s co nejvyšší

tzv. "čistotou" oblasti ("nečistá" oblast obsahuje příklady více konceptů).



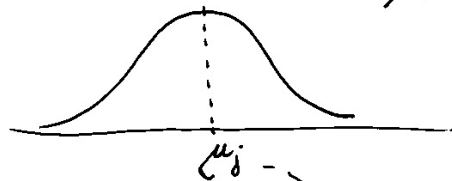
Příklad rozhodovacího stromu, který rozděluje dvoudimenzionální instancovní prostor na rektangulární atributy. ● označuje některý střed μ

Myslenka dělení instanačního prostoru je blízká podstatě RBF-síti, když funkce φ_j vrací vyšší hodnoty pro příklady blízké $\vec{\mu}_j$. Např. lze φ_j interpretovat jako důkaz podporující koncept v okolí nějakého $\vec{\mu}_j$; tento důkaz může být váhován.

Dalším krokem může být představa, že každá φ_j je asociována s jednou z oblastí definovanou rozhodovacím stromem. Na obrázku (strom a regiony) je umístění středů RBF vyznačeno pomocí \bullet .


(Pozn.: instanační prostor je ohraničený, přičemž hranice jsou dány extrémními hodnotami atributů tak, jak byly získány z trénovací množiny.)

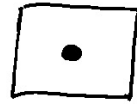
Umístění středu $\vec{\mu}_j$:



Jsou tři možnosti:

1) Leží-li na hranici instanačního prostoru pouze jediná strana oblasti, pak $\vec{\mu}_j$ se umístí doprostřed této strany. 

2) Leží-li dvě sousední strany oblasti na hranicích instanačního prostoru, pak $\vec{\mu}_j$ se umístí do toho těchto stran. 

3) Leží-li ^{všechny} hranice oblasti na hranicích s ostatními oblastmi, umístí se $\vec{\mu}_j$ do geometrického středu oblasti. 

Nyní, jak stanovit parametry zvonovitých (gaussovských) funkcí?

$$\varphi_j(\vec{x}) = e^{-\frac{\|\vec{x} - \vec{\mu}_j\|^2}{2\sigma^2}}$$

Nechť σ_{jk} označuje k -tou souřadnici $\vec{\mu}_j$. Nechť standardní odchylka j -té gaussovské funkce pro k -tou dimenzi je σ_{jk}^2 . Pak lze výše uvedenou rovnici přepsat za použití vztahu $\|\vec{x} - \vec{\mu}_j\|^2 = \sum_k (x_k - \mu_{jk})^2$ a pomocí skutečnosti, že $e^{-\sum_i x_i} = \prod_i e^{-x_i}$:

$$\varphi_j(\vec{x}) = \prod_{k=1}^m e^{-\frac{(x_k - \mu_{jk})^2}{2\sigma_{jk}^2}}$$

Nevázané σ_{jk}^2 určují, jak strmě $\varphi_j(\vec{x})$ klesá s rostoucí vzdáleností $\vec{\mu}_j$ podél každé dimenze. Lze stanovit intuitivní pořadavek, aby tento parametr závisel na rozsahu hyperrektangulární oblasti tak, aby hodnota φ_j na hranici oblasti byla vždy stejná.

Nechť I_{jk} je rozsah k -té dimenze j -té oblasti. Dále nechť parametr $\alpha^2 = I_{jk}^2 / \sigma^2$ (α je stejné pro všechny dimenze a pro všechna j , tj. poměr I_{jk} a σ je konstantní). Potom:

$$\varphi_j(\vec{x}) = \prod_{k=1}^m e^{-\frac{\alpha^2 (x_k - \mu_{jk})^2}{2I_{jk}^2}}$$

Je důležité si uvědomit, že jedna větev stromu velmi často testuje pouze podмноžinu atributů. Protože ostatní atributy jsou pro daný podprostor nerelevantní, stačí počítat produkt Ji pouze pro ty hodnoty k, které odpovídají relevantním atributům (pro irelevantní je hodnota exponenciální funkce = 1).

Po stanovení RBF (tj. $\varphi_j(\vec{x})$) je nutno určit váhy do výstupní vrstvy:

Uspořádají-li se trénovací příklady do matice X tak, že každý řádek reprezentuje jeden příklad a j -tý sloupec obsahuje hodnotu φ_j pro tento příklad, lze použít metody tzv. pseudoinversní matice X^P .

Je nutno také přidat atribut, jehož hodnota je vždy 1 pro φ_0 .

Nechť C označuje tzv. klasifikační matici, kde každý sloupec ~~označuje~~ reprezentuje jeden koncept:

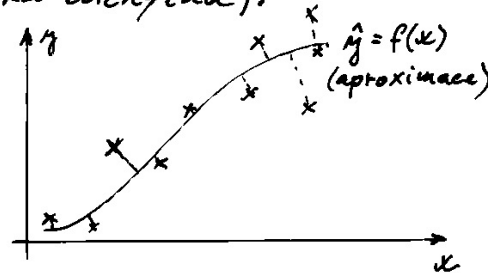
je-li r -tý příklad označen jako i -tý koncept, pak i -tá hodnota v r -tém řádku matice C se nastaví na 1 a všechny ostatní hodnoty na -1.

Jsou-li tedy dány C a X , je cílem nalézt matici vah W , která minimalizuje střední kvadratickou chybu $X \cdot W - C$. Tého lze dosáhnout výpočtem tzv. pseudoinversní matice X^P . Bylo ukázáno, že střední kvadratická chyba je minimální, pokud platí $X^T X$ je nesingulární!

$W = (X^T X)^{-1} X^T C = X^P C$; X není čtvercová matice.
 (lze snadno počítat v systému MATLAB!)
 $X^P X = I$ (lineární nezávislost sloupců není)
 nebo X^{-1} nemusí existovat

Alternativa ke stanovení vah - MNČ

Pro malé hodnoty učicí konstanty lze metodu gradientního sestupu "dobře" aproximovat pomocí aplikace metody MNČ (metoda nejmenších čtverců), často užívané pro aproximaci funkcí průběhů na základě hodnot bodů (hledá se takový průběh funkce, aby byla minimalizována střední kvadratická odchylka).



Aproximační funkce se položí známými body tak, aby součet kvadratických vzdáleností těchto bodů od bodů na aproximační křivce byl minimální.

$$w^* = X^P \cdot d$$

kde w^* je vektor vah v globálním minimu konvexní chybové kritériální funkce, X^P je tzv. pseudoinversní matice (či zobecněná inverzní matice), d je ~~vektor~~ požadovaných ^{vektorů} výstupů, X necht' je matice vektorů vstupů:

$$X = [x^1 \ x^2 \ \dots \ x^m] \quad d = [d^1 \ d^2 \ \dots \ d^m]$$

$$X^P = (X^T X)^{-1} X^T \quad \text{pro } m > n + 1 \quad (\mathbb{R}^n \rightarrow \mathbb{R})$$

Extremy (minimum a maximum) funkce $E(w)$ jsou řešením vztahu:

$$\nabla E(\vec{w}) = 0$$

Proto pro minimum kritériační funkce

$$E(\vec{w}) = \frac{1}{2} \sum_{i=1}^m (d^i - y^i)^2$$

musí platit:

$$\nabla E(\vec{w}) = - \sum_{i=1}^m [d^i - (\vec{x}^i)^T \vec{w}] \vec{x}^i = X(X^T \vec{w} - \vec{d}) = 0$$

což lze přepsat na

$$XX^T \vec{w} = X\vec{d}$$

což pro nesingulární matici XX^T dává řešení:

$$\vec{w}^* = (XX^T)^{-1} X\vec{d}$$

Řešení ovšem pouze zužuje výběr na to, že \vec{w}^* je z lokálního hlediska minimum, maximum, či sedlový bod, takže je zapotřebí řešení otestovat vyhodnocením tzv. Hessiánské matice (její druhé derivace):

$$\nabla \nabla E = \left[\frac{\partial^2 E}{\partial w_i \partial w_j} \right]$$

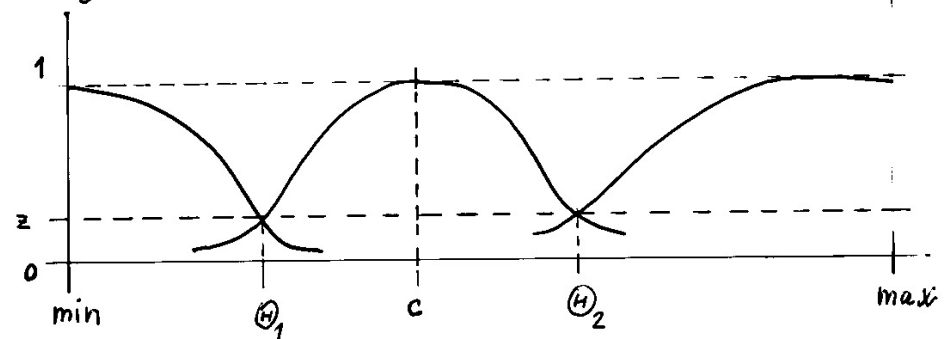
a ukázat, že je pozitivně definitní. Pak \vec{w}^* je minimum.

($n \times n$ reálná symetrická matice A je pozitivně definitní, pokud kvadratická forma $x^T A x$ je striktně pozitivní pro všechny nenulové sloupcové vektory x v \mathbb{R}^n .)

Algoritmus inicializace RBF lze sumarizovat:

1. Indukuj rozhodovací strom, který rozdělí je prostor na hyperrektangulární oblasti.
2. Transformuj strom na topologii RBF-sítě tak že každá oblast je reprezentována jedním neuronem.
3. Urči váhy do výstupní vrstvy pomocí pseudoinverzní matice.

Ilustrace transformace hyperrektangulárních oblastí na gaussovské funkce:



Zde, pro jednoduchost, definoval indukovaný rozhodovací strom tři oblasti. Nad každou oblastí je stanovena gaussovská funkce s maximumem = 1 a s hodnotami na hranicích = α (tato hodnota je dána pomocí α). Citlivost klasifikátoru na hodnotu α není velká, protože hodnoty sousedních φ_j jsou na hranicích stejné.

Hodnoty funkcí jsou váhovány vahami w_{ij} , takže o zařazení do konceptu i rozhoduje $y_i = \sum_j w_{ij} \varphi_j$

Učení simulováním evoluce

Prohledávání prostoru řešení je možné i pomocí dalších algoritmů inspirovaných biologickými systémy. Jednou ze slibných cest je napodobování přirozeného vývoje. Tato skupina výpočetních metod bývá obvykle nazývána jako genetické algoritmy. Genetické algoritmy jsou založeny na analogii darwinovské teorie evoluce přírodních druhů a používá obdobnou terminologii: jedinec, párování, křížení chromosomů, mutace genů, přizpůsobenost, přírodní výběr.

Důležité pro stanovení selekčního mechanismu je nejen výběr nejlepších jedinců, ale také různorodost mezi jedinci.

V podstatě jde o hledání globálního maxima (či minima) v prostoru s možným značným počtem lokálních maxim (či minim), přičemž lokální maxima jsou "obydlena" jedinci, kteří byli "obětováni" proto, aby umožnili ostatním se dostat od lokální "pasti" dostatečně daleko a nakonec najít globální maximum.

Biologická inspirace

U vyšších rostlin a živočichů obsahuje každá buňka jádro, v němž je obvykle mnoho tzv. chromosomů. Chromosomy jsou složeny z genů, které jsou odpovědné za přenos vlastností živého jedince při dělení buněk a při vzniku potomků z rodičů. Geny jsou v chromosomech uloženy za sebou.

Princip "Ti nejprizpůsobivější přežívají"

Ve svém hlavním díle "The Origin of Species" ("O původu druhů"), publikovaném r. 1859, vyslovil Charles Darwin princip vývoje pomocí přírodního výběru:

- každý jedinec má sklon předávat své vlastnosti potomkům;
- příroda přesto vytváří jedince s odlišnými vlastnostmi;
- nejprizpůsobivější jedinci (tj. ti s nejžádanějšími vlastnostmi) mají tendenci produkovat více potomků než ostatní, čímž směřují celou populaci směrem k těmto "žádaným" vlastnostem;
- během dlouhých období se mohou nahromadit variace vytvářející zcela nové druhy, jejichž vlastnosti jim umožňují obzvláště dobrou schopnost přežít ve zvláštních ekologických oblastech.

Z perspektivy molekulárního hlediska je přirozený výběr umožněn změnami způsobenými křížením a mutací.

Křížení sestavuje existující geny do nových kombinací.

Mutace produkuje geny nové, dosud nevidané.

Geny u biologických systémů

Nositelkou dědičnosti je DNK (deoxyribonukleová kyselina), obsažená v buněčných jádrech. Buněčné jádro má velikost cca 0,02 mm. Vlákno DNK má délku přibližně 2 m.

^(DNA)
Řetěz DNK je složen z deoxyribózy (druh cukru) a čtyřmi druhy jednoduchých chemických látek (nazývaných v kontextu genetiky jako „písmena“):

- A - adenin
- G - guanin
- C - cytosin
- T - thymin

Počet a pořadí „písmen“ je určující pro genetickou informaci, uloženou v řetězu DNK. Jednotka dědičnosti gen bývá tvořen několika tisíci písmeny (u člověka cca ~~100~~ 1000 genů). V současnosti je známo o řetězci velmi málo, pouze o necelých 10% existuje určitá znalost. Geny kódují tvorbu bílkovin (tím i stavbu a funkci buněk). Odhaduje se, že snad 30% genů odpovídá za stavbu a činnost mozku.

Potomci dědí 1/2 genů po biologické matce a 1/2 po biologickém otci.

Vliv dědičnosti na inteligenci byl (dle pozorování) rostoucí s věkem dítěte (v 1 roku věku je inteligence ovlivněna dědičností z malé části; v 7 letech je to již 4x více).

Lidskou DNA tvoří asi 3 miliardy „písmen“, v jejichž řetězu je ukryto oněch ~~100~~ ³⁰ tisíc genů; je to ~~100~~ ^{asi 3} % z délky řetězu.

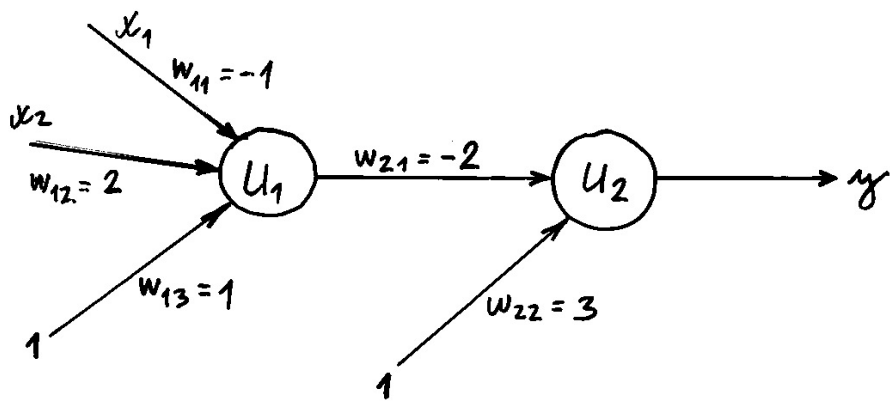
Např. bakterie *Hemophilus influenzae* (způsobuje těžký zápal plic u dětí a starých lidí a rovněž i smrtící zánet mozkových blan) má dědičnou informaci tvořenou 1,8 miliony „písmen“. Gen je tvořen obvykle několika tisíci písmeny v určitém místě DNA. *Hemophilus* má dědičnou informaci tvořenou 1743 geny. Na rozdíl od vyšších organismů tvoří tyto geny téměř celý řetěz.

V roce 1990 byl zahájen rozsáhlý projekt HGP (Human Genome Project, tj. Projekt lidský genom). Genom je dědičná informace nějakého tvora. Cílem projektu je zjistit pořadí všech 3 miliard „písmen“ lidské dědičné informace, včetně složení a umístění všech lidských genů. Význam bude značný, např. v boji se zhoubnými nádory

3% délky řetězce u lidí tvoří geny. Ze zbylých 97% část funguje jako „vypínače“ (kontrolují činnost správných genů ve správné době, např. v určité době se tvoří buňky jaterní, kostní...).

O 85% se v podstatě neví nic. Odhalení jejich účelu bude pravděpodobně mít zásadní vliv na medicínu.

Optimalizace NN pomocí GA



Každou váhu lze zakódovat do binárního řetězce pomocí tří bitů, kde nejlevější bit je znaménkový, např. 110 reprezentuje -2, 011 t. +3, atp.

Reprezentace sítě z obrázku:

$s = (101|010|001|110|011)$ odpovídá řetězci vah

$$(w_{11}, w_{12}, w_{13}, w_{21}, w_{22}) = (-1, 2, 1, -2, 3)$$

Začít lze s náhodně vygenerovanou populací ~~vah~~ řetězců vah (tj. sítí, náhodných sítí).

Funkci přizpůsobivosti populace lze definovat jako $-E$, kde $E = E(\vec{w})$:

$$E(\vec{w}) = \frac{1}{2} \sum_{k=1}^m \sum_{l=1}^L (d_k - y_k)^2$$

Operaci křížení řetězců lze chápat jako výměnu částí neuronových sítí navzájem, v naději, že vznikne síť s vyšší hodnotou funkce přizpůsobení.

Další potenciální možnosti aplikací GA na NN:

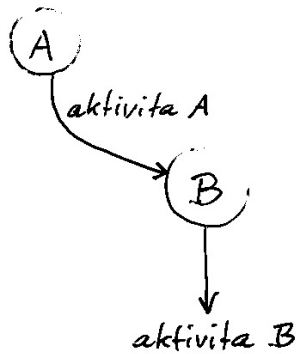
- optimalizace výkonnosti NN hledáním vhodné architektury a parametrů;
- minimalizace rozměrů NN;
- maximalizace rychlosti odezvy NN;
- optimalizace HW implementace VLSI minimalizací počtu spojů;
- optimalizace učicí konstanty (zrychlování učení);
- evoluce pravidel pro učení NN (odhalování procesu učení).

Hebbovo učicí pravidlo

Donald Hebb poprvé popsal v r. 1949 změny, ke kterým dochází na buňkové úrovni u zvířat, když se učí. Hebbovo pravidlo ve stručnosti říká, že když nějaký neuron stimuluje jiný neuron v okamžiku, kdy je přijímající jednotka aktivní, posiluje se spojení mezi oběma buňkami. Popis buněčných změn v mozku během učení tvoří jeden z klíčů poznání, ovlivňujících biologické modely učení.

Hebbovy výzkumy úzce souvisejí s tzv. Pavlovovým podmíněným reflexem (pes slůtká při zaslechnutí zvonku, protože se naučil, že s potravou je spojen zvuk).

Původní Hebbovo pravidlo má některé nedostatky.
Model Hebbova učení:



Pseudokód Hebbova učení:

Pro každý časový krok t :

- {
- vypočítej aktivitu A;
- vypočítej aktivitu, kterou B přijímá od A;
- vypočítej aktivitu B;
- IF aktivita B > 0 AND aktivita, kterou B přijímá > 0
- THEN zvýš sílu spojení z A do B;
- }

Hebbovo pravidlo však např. nespecifikuje, jak mnoho se má síla spojení zvýšit, ani jak potřebné aktivity počítat. Není ani řečeno, zda se může síla spojení někdy zeslabovat. Kvantity, které nejsou omezeny žádnými hranicemi, tvoří při tvorbě počítačových programů potíže. Dále, původní Hebbovo pravidlo nestanoví, jaké přesně jsou podmínky zesilování spoje.

Neo-Hebbovské učení

Neo-Hebbovské učení je pokus o přepsání původní poněkud vágní myšlenky do konkrétní matematické formy. Bylo rozvinuto v r. 1960 a poskytuje solidní matematickou teorii Pavlovovského podmíněného učení. Prodané počítačové podmínky je stanoveno, jak se v následujících krocích mají počítat aktivity a síla spojení (váhy). Byly rovněž definovány dva pojmy, aplikovatelné na neurony libovolné sítě:

outstar - "vyzařující hvězda": představa neuronové jednotky jako objektu, rovnoměrně vysílající svůj výstupní signál do okolí zaplněného dalšími neurony;

instar - "přijímající hvězda": představa neuronové jednotky jako objektu, přijímajícího signály z okolí více-méně rovnoměrně vyplněného ostatními neurony.

Každý neuron je tedy chápán jako centrum vyzařující i přijímající "hvězdy", tj. vydává i přijímá stimulační signály.

To, co je příčinou činnosti neuronové sítě, pak je právě neustálá změna interakcí mezi oběma typy „hvězd“.

Původní Hebbovský model je založen na diferenciálních rovnicích. Praktické použití metody často dává přednost využití diferenciálních rovnic (kvůli zjednodušení výpočtů): časový krok mezi dvěma výpočty je tvořen velmi malou hodnotou, avšak konečnou. Tím je zároveň předurčena existence chyby, protože velikost kroku tuto chybu ovlivňuje.

Mezi požadovanou přesností a dobou výpočtu je v realitě vždy nutno udělat jistý kompromis, proto se náročné simulační výpočty (např. modelování atmosférických dějů pro předpovědi počasí) provádějí na superpočítačích, aby se minimalizovaly ztráty přesnosti.

Simulátor Hebbova učení pro outstar/instar používá tyto rovnice:

Výstupní aktivita pro outstar/instar

$$y_j(t+1) - y_j(t) = \Delta y_j(t) = -A y_j(t) + I_j(t) + \sum_{i=1}^M w_{ij}(t) [y_i(t - \tau) - T]$$

aktivita
externí vstup
dobu přenosu signálu
práh
konstanta poklesu aktivity
0 < A < 1

Změna vah pro outstar/instar

$$w_{ij}(t+1) - w_{ij}(t) = \Delta w_{ij}(t) = -F w_{ij}(t) + G y_j(t) [y_i(t - \tau) - T]$$

učicí konst.
zapomínací
práh

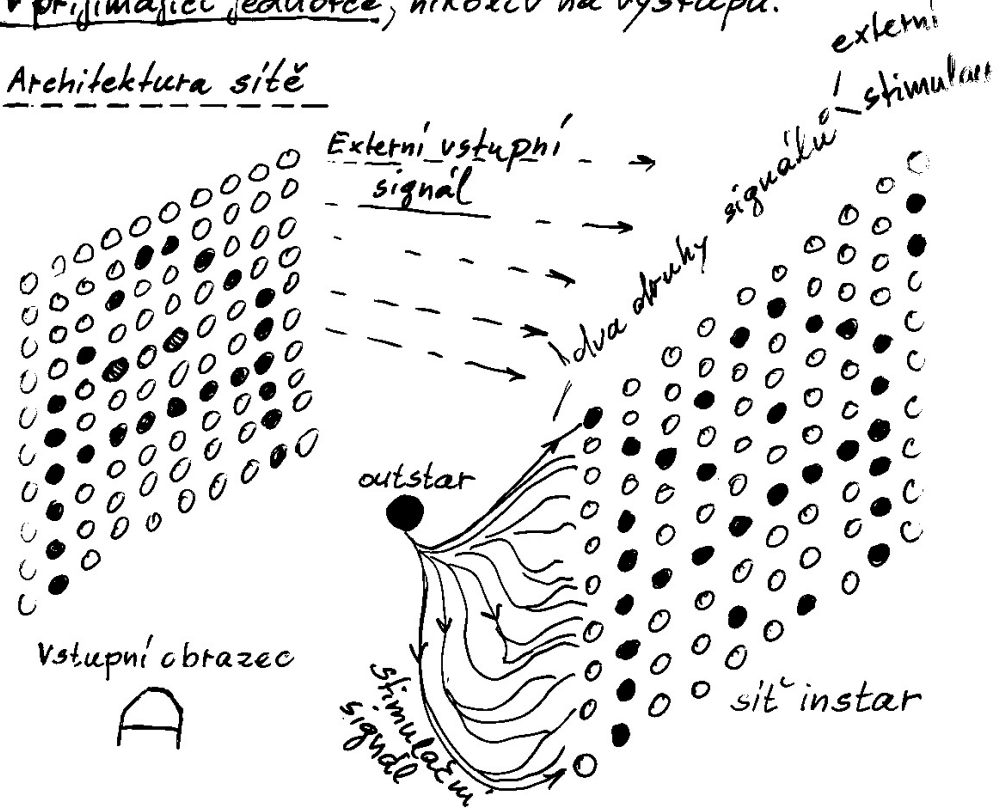
(zde je i -tý neuron jako outstar a j -tý jako instar; existuje celkem n neuronů.)

- $0 < F < 1$ „zapomínací konst.“
 - $0 < G < 1$ „učicí“ konst.
 - τ ... doba přenosu signálu
 - I ... externí vstup
- T ... práh z i do j
(prahy jsou v přijímači)

V uvedených rovnicích se předpokládá, že výstup neuronů je ekvivalentní jejich aktivizačnímu úrovní, takže výpočet aktivace rovněž poskytuje hodnotu výstupu.

Rovnice také, jak je zřejmé ze zápisu, předpokládají, že jakékoliv prahy v síti jsou implementovány v přijímající jednotce, nikoliv na výstupu.

Architektura sítě



Kromě stimulačních signálů z neuronu outstar (vysílajícího) přijímá jednotka v síti instar také externí vstupní signál z odpovídající pozice externího vstupního obrazce.

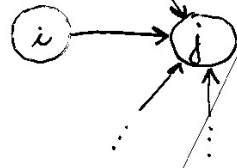
Na uvedeném obrázku působí jeden neuron jako tzv. outstar a vyzařuje své výstupní signály do každé jednotky sítě instar. Přijímající neurony obdrží dva druhy signálu: jeden z outstar neuronu, druhý z odpovídající pozice externího stimulatoru (Externí stimulus je signál, který přichází do každého neuronu sítě instar a který zároveň nepochází z neuronu outstar.)

Externí signál svou povahou odpovídá např. dopadu fotonu na fotoreceptor nebo signálu přicházejícímu z jiné části sítě či dokonce z jiné sítě.

Rovnice pro aktivitu (viz též výše) i : vliv pouze z OUTSTAR!

$$\Delta y_i(t) = -A \cdot y_i(t) + I_i(t) + \sum_{i=1}^n w_{ij}(t) \cdot [y_i(t-\tau) - T]$$

Rovnice vyjadřuje změnu aktivity j -té instar jednotky. V rovnici vystupují celkem 3 členy:



člen $-A \cdot y_i(t)$ vyjadřuje míru, s níž klesá aktivita instar neuronů, pokud se s neuronem nic jiného neděje. Pokles je exponenciální a je řízený velikostí konstanty A . Vysoká hodnota způsobí velmi rychlý pokles téměř k nule, nízká hodnota znamená pokles pomalý. Rychlost poklesu ovlivňuje to, jak rychle je instar neuron připraven na další stimul.

Člen č. 2, tj. $I_i(t)$, znamená velikost libovolného externího stimulu (tj. stimul např. z vnějšího vzoru A), který může být přijat v časovém okamžiku t . V biologických systémech může být externím stimulem cokoliv počínaje vstupem z jiné části sítě neuronů a konče signálem ze situace oka, kam dopadl foton.

Důležitě je, že se jedná o vnější systém stimulace vůči výpočetnímu systému.

Třetí člen je $w_{ij}(t) \cdot [y_i(t-\tau) - T]$. Tento člen vyjadřuje účinek stimulu vysílajícího neuronu (outstar neuronu) na aktivitu neuronu v přijímající (instar) síti. Zde w_{ij} je velikost váhy přiřazená spoji z outstar neuronu do instar j -tého neuronu. Část výrazu, tj. $[y_i(t-\tau) - T]$, vyjadřuje velikost signálu, jenž právě přichází do jednotky instar z jednotky outstar. Doba přesunu signálu je τ a velikost prahu vstupního připojení je dána pomocí T .

Znamená to, že aktivita přijímajícího neuronu je ovlivněna pouze vstupními stimuly z vysílajícího neuronu, a to stimuly právě v určitém časovém okamžiku přijatými (na rozdíl od situace, kdy outstar již vysílá další signál, který však doposud ještě nebyl přijat). Kromě toho to znamená, že pouze ty signály, které jsou přinejmenším tak velké jako prahová hodnota T , mohou ovlivnit instar jednotku. (Pozn.: je-li vstupní signál $< T$, pak se celý třetí člen považuje za nulový!)

V příkladu na obrázku existuje pouze jeden vysílající (outstar) neuron, takže není nutno počítat přínosy jednotlivých outstar neuronů. Obecně je ovšem možné mít více outstar neuronů, takže potom je zapotřebí sumovat třetí člen $w_{ij}(t) \cdot [y_i(t-\tau) - T]$ přes všechny outstar jednotky, které v síti existují.

Rovnice pro stanovení vah (viz též výše)

$$w_{ij}(t+1) - w_{ij}(t) = \Delta w_{ij}(t) = \underbrace{-F \cdot w_{ij}(t)}_{\text{zapomínání}} + \underbrace{G \cdot y_j(t)}_{\text{učicí konst. } y_j \text{ a zároveň } y_i!} [y_i(t-\tau) - T]$$

První část výrazu, $-F \cdot w_{ij}(t)$, je tzv. člen zapomínání. Určuje pokles hodnot vah, pokud nejsou obnovovány novým učením. „Zapomínací“ konstanta F řídí rychlost zapomínání – velké hodnoty znamenají rychlé zapomínání, malé hodnoty pak pomalé zapomínání.

Druhá část výrazu, tj. $G \cdot y_j(t) \cdot [y_i(t-\tau) - T]$, je tzv. Hebbovský učicí člen. Vyjadřuje vlastně matematicky Hebbovo pravidlo, které říká, že významné učení se vyskytuje pouze tehdy, když jak aktivita přijímajícího neuronu $y_j(t)$, tak velikost právě přijatého signálu $[y_i(t-\tau) - T]$ jsou silné. Je-li kterákoliv z obou aktivit nízká, je součinem příslušně nízký také.

(Vliv parametrů A, T, τ, F a G bude předmětem experimentů ve cvičení.)

Diferenční Hebbovské učení

Hebbovské učení je jedním ze základních učicích modelů v oblasti neuronových sítí. Problém ve srovnání s biologickým učením spočívá v tom, že Hebbovské učení je poněkud „hrubé“, nesleduje detaily biologického učení.

Významným problémem je, že při Hebbovském učení hodnoty vah mohou pouze vzrůstat, což je pro biologické systémy nemožné (neexistence hranic hodnot).

Tato vlastnost je ovšem přímo obsažena v pravidle Hebbovského učení.

Neo-Hebbovské učení (novo-Hebbovské učení) poskytuje určitou metodu, jak zmíněný problém vyřešit. Různé modifikace Hebbovského učení mají společnou vlastnost v tom, že využívají tzv. diferenční učicí pravidlo.

Diferenční učení umožňuje ^(počítat) změny síly spoji (vahy) pomocí rozdílů mezi aktivitou přijímajícího neuronu a změnou vstupního stimulačního signálu.

Jednoduchá verze pravidla je popsána vztahem:

$$\Delta w_{ij} = \beta \cdot \Delta y_i \cdot \Delta y_j$$

změna spojení = β · změna in. · změna out.

kde β je tzv. učicí konstanta, $0.0 \leq \beta \leq 1.0$. Symboly Δ se týkají změn odpovídajících hodnot mezi předchozím okamžikem $t-1$ a současným t . Změna může být pro každý neuron pozitivní (aktivita vzrůstá), negativní (aktivita klesá), či nulová (aktivita zůstává nezměněna).

Tento způsob výpočtu se od prostého Hebbovského pravidla značně liší. V diferencním Hebbovském učení se žádné učení neuskutečňuje, pokud je aktivita neuronu beze změny. Rovněž je možno vahy snížovat v případě klesající aktivity a vyšovat rostoucí aktivitou.

Pseudokód pro diferenciální Hebbovské učení

Pro každý časový okamžik:

- {
- vypočítej aktivitu A;
- vypočítej aktivitu, kterou od A přijme B;
- vypočítej aktivitu B;
- stanov změnu v aktivitě od A, kterou přijme B;
- stanov změnu aktivity B;
- vypočti změnu síly spojení z A do B.
- }

Teorie řízeného posilování (DRT)

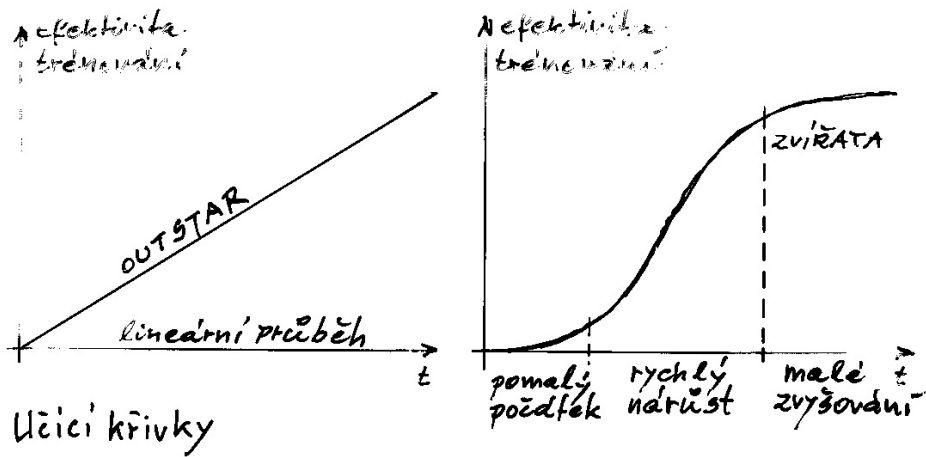
Přestože je diferenciální Hebbianské učení lepším modelem biologického učení než prosté Hebbianské či neo-Hebbianské, stále existují některé problémy, zejména vzhledem k jejich schopnosti modelovat klasickou podmíněnost u zvířat.

Největším problémem je čas. U experimentů se zvířaty bylo zjištěno, že učení probíhá rychleji, pokud podmiňující stimul (zvonek u Pavlova experimentu) působí (zazvoní) před nepodmiňujícím stimulem (jídlem).

Avšak u modelu „outstar“ (^{vysílač} ~~průhled~~) se umělé neurony učí ze stimulů, které se objevují ^{vysílač} současně. To je vážný problém při snaze využít ^{vysílač} ~~průhled~~ k vysvětlení klasického podmíněování. ^{vysílač} ~~průhled~~ působí pouze v jediném, spojitě se měnícím okamžiku. Nemá žádnou minulost ani budoucnost. Reaguje pouze na stimul v právě probíhajícímu momentu.

Oproti tomu mají biologické systémy schopnost zapamatovat si více než jednu instanci (v čase).

Dalším problémem je tzv. akviziční křivka: čára, znázorňující průběh učení v čase. U ^{vysílač} ~~průhled~~ je to průhled; ^{vysílač} ~~průhled~~ se učí lineárně a po dvojnásobném počtu trénovacích impulsů je dvojnásobně schopnější. U zvířat však má akviziční křivka tvary písmene S. Zvířata se zpočátku učí velmi pomalu. Později, se vzrůstající zkušeností, se učení značně zrychluje. A ke konci opět učení probíhá pomaleji (delší učení zvyšuje výkonnost jen nepatrně).



Modifikovaná verze Hebbianského učení (Harry Klopf) byla použita pro modelování klasického podmíněvání: systém zvaný **(DRT)** (drive-reinforcement theory) je jedním z nejlepších umělých systémů pro modelování biologických systémů. Podobně jako ostatní umělé systémy je utčen dvěma rovnícemi:

- rovnici aktivity:

$$y_i(t) = \sum_{j=1}^n w_{ij}(t) [y_j(t) - T]$$

Aktivita

což je v podstatě totéž jako u metody outstar. Aktivita y_i této jednotky závisí na váhovaném součtu aktivit přicházejících z každého i -tého neuronu (těch je celkem n). Každý vstupní signál musí být individuálně větší než práh T před tím, než může přispět k váhovanému součtu. Vstupní signály $\leq T$ jsou považovány za nulové.

- rovnici učení:

$$\Delta w_{ij}(t) = \Delta y_j(t) \sum_{k=1}^{\tau} \beta |w_{ij}(t-k)| \Delta y_i(t-k)$$

Učení

Tento vztah je složitější než pro outstar. Rovnice vyjadřuje diferencenci učící pravidlo, kde změna vah závisí na součinu změny aktivity přijímačci jednotky (Δy_j) a změny aktivity vstupního signálu (Δy_i).

Dále je učicí konstanta β umístěna už za Σ , což znamená, že nemusi být stejná pro všechny signálové spoje po celou dobu. Vahy jsou brány v absolutní hodnotě. Zároveň rovněž na relativních silách spojů, nikoliv na tom, zda vahy jsou pozitivní nebo negativní (i když pro výpočet aktivity přijímačci jednotky na vahách záleží). A konečně sumování se provádí přes sérii časových impulsů, počínaje impulsem před okamžitým časovým bodem a konče celkovým počtem τ impulsů před daným okamžikem.

Tj. DRT berou do úvahy nejen vstupní stimuly v daném časovém okamžiku, nýbrž i určitou historii vstupů přes nějakou časovou periodu!

Obecněji se používá ještě komplikovanějšího vztahu, který zahrnuje dobu přechodu signálů mezi neurony a také učicí konstanta β může být funkcí času, $\beta(t)$.

Na síti typu DRT se obvykle kladou i další požadavky:

Normálně je každý synaptický styčný bod předurčen k tomu, že je buď excitační ($w > 0$) nebo inhibiční ($w < 0$). Není umožněno, aby $w = 0$; pokud je váha některé synapse nulová, zůstává taková napořád, protože všechny budoucí změny vah jsou za takových podmínek automaticky nulové. Důsledkem je, že žádná váha nemůže přejít nulový bod; pokud začne jako pozitivní, zůstává navždy pozitivní, a naopak. Toto omezení je konsistentní s biologii: nejsou žádné synapse, které by byly někdy excitační a někdy inhibiční.

Obvykle také bývají aktivací hodnoty každého neuronu v síti omezeny (nejčastěji na interval $0.0 \dots 1.0$).

Rovněž změny vstupní aktivity (Δy_i) jsou obvykle omezeny pouze na kladné změny (při klesající síle signálů nedochází k žádnému učení).

Modifikovaná DRT může také využívat místo amplitudy pulsů jejich frekvenci:

$$y(t) = \mu \sum_{k=1}^{\infty} x(k) e^{-\mu(t-k)}$$

kde μ je konstanta poklesu determinující relativní význam současných a minulých událostí.

$x(k) = 1$ když puls dorazí v čase k , jinak $x(k) = 0$.

Změnu signálu lze počítat jako:

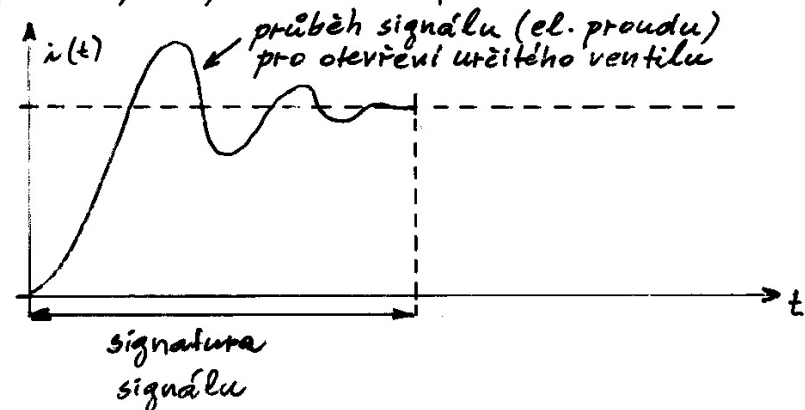
$$\Delta y_i(t) = \mu [x(t) - y(t)]$$

Učení se posloupnosti vzorů

V řadě aplikací je užitečnější se naučit rozpoznávat sekvence vzorů než jednotlivé vzory. Např. pro poskytnutí realistického obrazu o napětí v zásuvce nepostačuje vědět, že má 220 V, protože se jedná o časově proměnný signál mající sinusový průběh s hodnotami mezi $+a$ a $-a$, přičemž signál má periodu 50 Hz.

Je tedy lepší mít k dispozici cyklickou signaturu signálu než pouhé jedno měření.

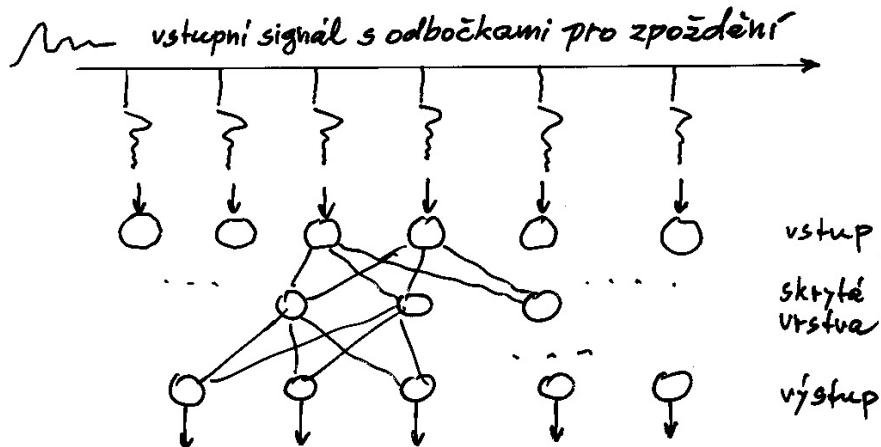
Časově proměnné signály nemusí mít pouze podobu cyklicky se opakujícího průběhu jako např. napětí v zásuvce. Např. elektricky ovládaný hydraulický ventil je otevírán a zavírán, přičemž se k tomu využívá proměnné množství el. proudu. Vzniklý přechodový signál pak je charakteristický pro konkrétní ventil i pro konkrétního výrobce ventilů. Ze zaranžovaného průběhu proudu při otevření/zavření může odborník poznat, který ventil akci provedl:



Jiným problémem bývá reprodukce určitého signálu (na rozdíl od rozpoznání).

Tento druh problému se vyskytuje např. v robotice, kdy systém má uloženy v paměti různé sekvence pohybu ramene robota. Asociativní paměť (heteroasociativní) na základě určitého stimulu vyvolá místo jediné akce celou posloupnost. Tomuto typu zařízení se říká „kolovrátek“ nebo „flašinet“ podle analogického principu.

„Kolovrátková“ asociativní paměť je konstruována tak, že jsou v ní uloženy vzory v souladu s jejich sekvencními asociacemi: stimul ^{START} je asociován s akcí ~~START~~ A, akce A s B, B s C atd. Výstup sítě je zaveden zpět na její vstup jako další vstupní vzor. Síť tak automaticky projde celou sekvencí vzorů až do konce; poslední vzor je asociován s prázdným (zádným) výstupem. Existuje zde problém spojený s požadavkem na výskyt určitého vzoru v sekvenci právě jednou: pokud by např. C bylo asociováno jednou s D a jindy s K, pak po A-B-C neumí síť určit, zda má následovat D či K.



Existuje množství signálů uvedeného typu (měsíční fáze úplněk → nov → úplněk, měření EKG, EEG apod., sonogramy ptáčího zpěvu nebo lidské řeči, atd.).

Předpokládejme, že je nutno řešit problém, kdy neuronová síť má rozpoznat konkrétní signál a zařadit ho do jedné z několika kategorií (určit ventil, nebo člověka...). V tomto případě se charakteristická sekvence měření vyskytne pouze jednou pro danou událost a sestává z průběhů přechodů signálu. Trvání signálu nechtě je 100 ms a měření se provádělo každou ms. Tím je k dispozici 100 prvkový vektor hodnot, což je ve skutečnosti „navzorkovaná“ charakteristika reálného signálu.

Získaný vektor je použit jako vstup sítě.

Uvedenou techniku vzorkování časově závislých signálů lze kombinovat prakticky s libovolnou sítovou architekturou, dokonce i s back-propagation, která je velmi pomalá z hlediska doby učení.

V reálných aplikacích se vzorkování implementuje hardwarově tak, že signál je poslán cestou, z níž vedou odbočky. Každá odbočka zpozdí signál o předem stanovenou dobu.

Pro 10 odboček a vzorkování po 5 ms to znamená, že první odbočka zpozdí signál o 45 ms, druhá o 40 ms, atd., a poslední o 0 ms, tj. předá síti vstup ihned. Výsledkem je, že se na vstupu sítě ocitne všech 10 hodnot naráz. Jinou metodou je použití Fourierovy transformace a zpracování signálu ve frekvenční oblasti.

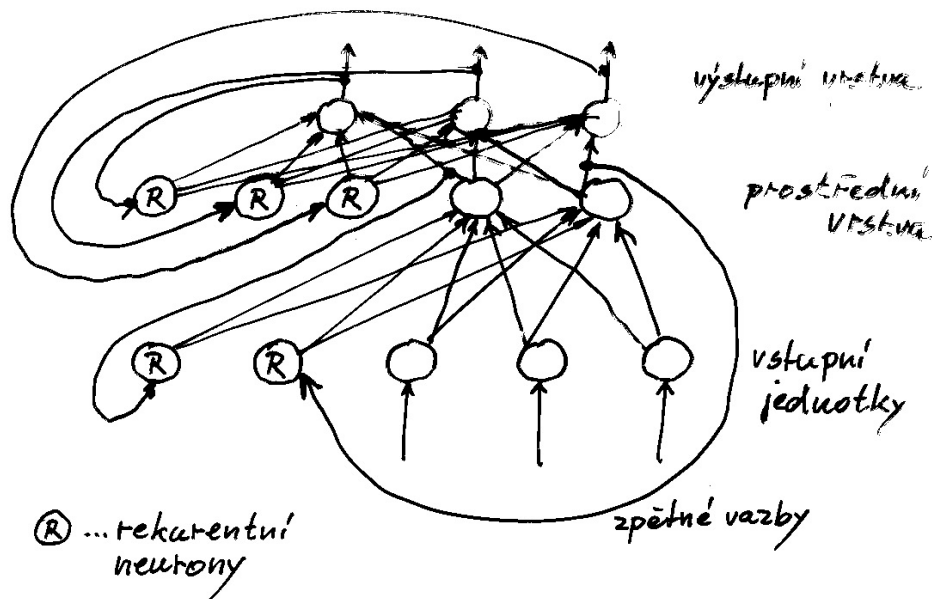
Rekurentní síť

Rekurentní síť patří k neúspěšnějším v oblasti učení se posloupnosti vzorů.

Rekurentní síť rozumíme takovou sítí, v níž vstupní aktivita projde sítí několikrát před tím, než je vygenerován výstupní vzor. (Tím jsou vyloučeny standardní síť typu back-propagation, kde je aktivita šířena jediným směrem vpřed od vstupu k výstupu.)

Mezi rekurentní síť patří síť, které disponují zpětnou vazbou na vstup, např. Hopfieldovy síť.

Netrekurentní síť (např. BP) je ovšem možné modifikovat na rekurentní, takže získáme např. rekurentní formu back-propagation:



Ⓡ ...rekurentní neurony

○ ..."obyčejné" neurony

Síť na obrázku přijímá 3 vstupní signály a vytváří 3 výstupy.

Kromě toho má vstupní vrstva navíc 2 neurony, tzv. rekurentní, připojené výstupy na neurony prostřední vrstvy.

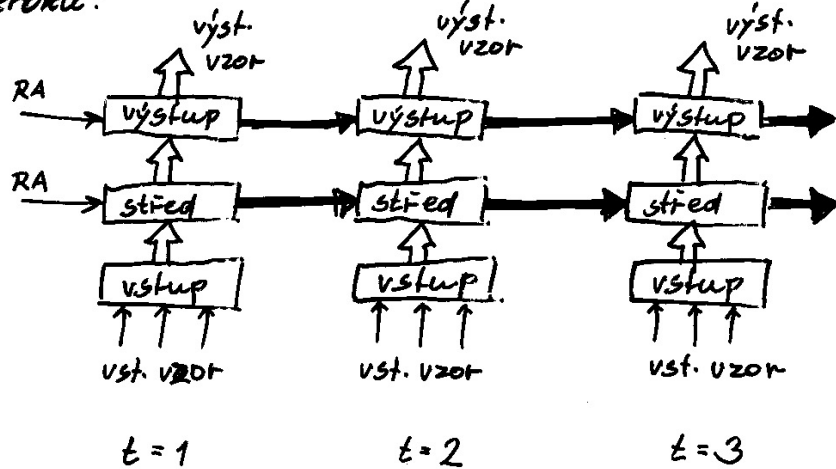
Podobně jsou přidány rekurentní neurony do prostřední vrstvy.

Rekurentní síť vyžadují podstatně více propojení a mnohem více paměti pro simulátory. Síť na obrázku má 12 adaptivních vah: 6 mezi vstupem a prostředkem a 6 mezi prostředkem a výstupem (architektura je 3-2-3). Přidáním rekurentních jednotek stoupne počet spojů na 25 (12 zelených plus 13 červených). (Vahy modrých jsou obvykle konstantní = 1.0)

Kromě zvýšeného počtu adaptivních vah je nutno při simulaci rekurentní sítě sledovat úroveň aktivit ne-rekurentních jednotek po několik časových kroků.

Pokud se vzorová sekvence skládá z N časových kroků, pak je třeba udržovat N kopií úrovní aktivity.

Ilustrace operací rekurentní sítě po několik časových kroků:



V čase $t=1$ je aplikován vstupní signál. Výstup a střed mají nějaké počáteční inicializační hodnoty ($\approx t=0$). Vstup je zpracován obvyklým způsobem a je vytvořen výstupní obrazec. Tento obrazec je uložen pro další výpočet chyb a změn vah.

V čase $t=2$ je aplikován druhý vstupní signál a šířen síť. Nyní však střed a výstup disponují přidavnými vstupy, které pocházejí z aktivit těchto vrstev v čase $t=1$. Tato aktivita je zkombinována se stimuly na vstupu a je generován výstup. Výsledek je opět uložen pro pozdější výpočty chyb a oprav.

Uvedený proces probíhá po N časových krocích sekvence vzorů. V každém kroku je generován nový výstup a uchována aktivita.

Teprve po proběhnutí všech N kroků se spočítají chyby a potřebné opravy vah. Výsledná oprava vah pro celou sekvenci je součtem N váhových změn.

Váhy nejsou změněny do doby, než jsou zpracovány všechny vzorové sekvence z trénovací množiny. Pro opravu vah se tedy použije kumulativní součet oprav (změn) za každou vzorovou sekvenci.

Popsané trénovací proceduru se říká dávková, protože všechny vzory jsou načteny před skupinovou změnou vah.

Rekurentní BP-algoritmus

```

repeat
    { vynuluj globální pole změn vah;
      for každou vzorovou sekvenci z trénovací množiny:
          nastav výstupní aktivity středů a výstupů
            na počáteční hodnotu (inicializace konstantou);
          vynuluj pole kumulativních změn vah;
          for každý vzor (časový krok) dané vzorové
            sekvence ( $t=1, 2, \dots, N$ ):
                aplikuj vstupní vzor pro daný krok  $t$ ;
                zpracuj vzor v síti;
                ulož hodnoty aktivit všech vrstev;
                 $t \leftarrow t+1$ ;
    }
  
```

uchovává změny vah pro celou tréninkovou množinu

pole pro uschování změn vah v kroku t

(9)

chyby se počítají
rekursivně od posledního
kroku.

for každý krok $N, N-1, N-2, \dots, 1$:
spočítej chyby výstupní a prostřední
vrstvy daného kroku;
 $t \leftarrow t-1$;

for každý krok $1, 2, 3, \dots, N-1, N$:
spočítej změny vah pro každý spoj pro
daný krok;
přidej váhové změny do kumulativního
pole změn vah
 $t \leftarrow t+1$;

(1) přidej kumulativní změny do globálního pole
změn vah;

...použij další vzorovou sekvenci z trén. množiny;
aplikuj globální změny na síťové spoje;

4} until trénování je hotovo.

Rovnice pro výpočty chyb a vah jsou složitější než pro
standardní BP, protože tektivity každého časového
kroku závisejí jak na aktuálním vzoru na vstupu,
tak na útorních aktivit předchozího kroku.

Rovnice jsou proto rekurzivní:

Napřed se spočítá dopředu tok aktivit sítě.
Předpokl, že používáme 3-rozměrný vstupní vektor
 \vec{x} , skládající se ze složek (x_1, x_2, x_3) , a síť z
obrázku (viz výše). Počáteční stav střední a výstupní
vrstvy je určen nějakými neutrálními hodnotami,
např. 0.3.

V čase $t=1$ každý z obou neuronů střední vrstvy
obdrží následující stimuly:

$$I_j^{mid} = \underbrace{\sum_{i=1}^3 W_{ij}^{in-mid} \cdot y_i^{in}}_{\text{přínos vstupu (jako stand. BP)}} + \underbrace{\sum_{k=1}^2 W_{kj}^{mid-mid} \cdot y_k^{mid}}_{\text{přínos zpětné vazby}}$$

mid ... střední vrstva
in-mid ... mezi vstupem (input) a středem (middle)
mid-mid ... ze střední do střední vrstvy

Pro $t=0$ platí, že $y(t-1)$ obdrží ~~akti~~ je počáteční
hodnota (v našem případě inicializace 0.3).

Vstup pro výstupní vrstvu:

$$I_j^{out} = \sum_{i=1}^2 W_{ij}^{mid-out} \cdot y_i^{mid} + \sum_{k=1}^3 W_{kj}^{out-out} \cdot y_k^{out}$$

out ... výstupní vrstva
mid-out ... mezi střední a výstupní
out-out ... z výstupní do výstupní

Pro $t=1$ se použije $y(t-1)$ inicializační hodnota (0.3).

V obou vrstvách se výstupní aktivita počítá stejně
jako u BP se sigmoidální nelinearitou:

$$y_j(t) = f(I_j(t)); \quad f(I) = \frac{1}{1 + e^{-I}}$$

(Sigmoida může být samozřejmě nahrazena i
jinou funkcí, např. ~~tan~~ arctg(I).)

Jakmile je spočtena veškerá aktivita v síti přes všechny časové kroky, může být stanovena chyba výstupního signálu.

Tento výpočet probíhá zpětně ($t = N, N-1, N-2, \dots, 3, 2, 1$).

$$E(N)_j^{out} = \left[y(N)_j^{\text{žadání}} - y(N)_j^{\text{skutečné}} \right] \cdot \frac{df(I(N))}{dI}$$

$$e(N)_j^{out} \equiv \left[y(N)_j^{\text{žadání}} - y(N)_j^{\text{skutečné}} \right]$$

$$E(N)_j^{out} = E(N)_j^{out} \cdot \frac{df(I(N))}{dI}$$

Musíme vzít do úvahy, že čas $t = N$ je orlívně časem $t = N-1$. Proto pro $t < N$ platí:

$$E(t)_j^{out} = \left[\underbrace{E(t)_j}_{\text{přimo měřená chyba}} + \underbrace{\sum_{i=1}^3 w_{ij}^{out-out} \cdot E(t+1)_i}_{\text{chyba rekurentních signálů šířená z následného kroku}} \right] \cdot \frac{df(I(t)_j)}{dI}$$

Předpokládáme zde, že současný výstup sdílí odporůdnost za své chyby s výstupem předchozího časového kroku. Výpočet je zjevně rekursivní ($N \rightarrow N-1 \rightarrow N-2 \rightarrow \dots \rightarrow 2 \rightarrow 1$).

Obdobné úvahy platí pro prostřední vrstvu:

$$E(N)_j^{mid} = \left[\sum_{i=1}^3 w_{ij}^{mid-out} \cdot E(N)_i^{out} \right] \cdot \frac{df(I(N)_j^{mid})}{dI}$$

Pro $t = N-1, N-2, \dots, 1$ platí:

$$E(t)_j^{mid} = \left[\sum_{i=1}^3 w_{ij}^{mid-out} \cdot E(t)_i^{out} + \underbrace{\sum_{k=1}^1 w_{kj}^{mid-mid} \cdot E(t+1)_k^{mid}}_{\text{rekurentní chyby mid-mid}} \right] \cdot \frac{df(I(t)_j^{mid})}{dI}$$

Jsou-li spočteny chyby pro $t = N-1, \dots, 1$, pak změny vah lze stanovit pro $t = 1, 2, \dots, N$.

Začínáme v $t = 1$:

$in\text{-}mid/mid\text{-}out$ - platí pro obojí propojení

$$\Delta w(1)_{ij} = \beta \cdot E(1)_i \cdot y(1)_j$$

Pro rekurentní váhy:

$mid\text{-}mid/out\text{-}out$

$$\Delta w(1)_{ij} = \beta \cdot E(1)_i \cdot y(0)_j$$

$$\Delta w(t)_{ij} = \beta \cdot E(t)_i \cdot y(t)_j$$

↳ pro nerekurentní váhy

$$\Delta w(t)_{ij}^{mid-mid/out-out} = \beta \cdot E(t)_i \cdot y(t-1)_j$$

↳ pro rekurentní váhy

Po spočítání změn vah je přidáme do kumulativních změn pro danou vzorovou sekvenci. Sumu pak přidáme do globálních změn pro celý tréninkový běh. Teprve nyní teálně provedeme změnu vah.

Tyto sítě se učí pomalu. Jsou schopny se naučit velmi složitě temporální vzory.

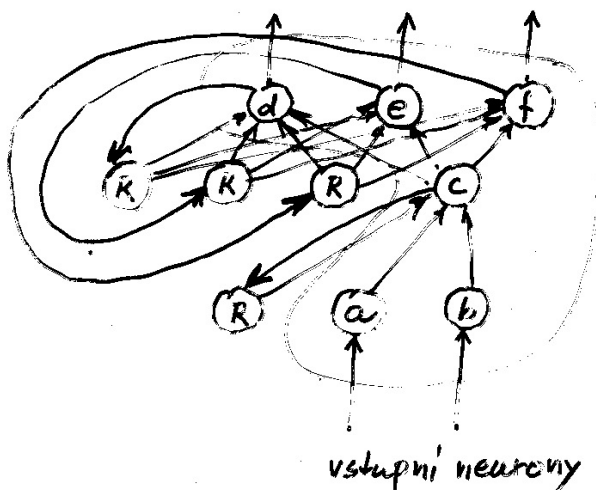
Příklad trénovací rekurentní sítě

Pro jednoduchost uvažme binární hodnoty, dva vzory, každý se třemi kroky:

| čas | vstup | výstup | Vzor B: | |
|-----|-------|--------|---------|--------|
| | | | vstup | výstup |
| 1 | 1 0 | 1 0 0 | 0 1 | 0 0 1 |
| 2 | 1 0 | 0 1 0 | 0 1 | 1 0 0 |
| 3 | 1 0 | 0 0 1 | 0 1 | 0 1 0 |

Učící konstanta β necht' je 0.5 a velikost RA necht' je 0.3 (tj. reset activity - po každé vzorové sekvenci se nastaví aktivity v prostřední a výstupní vrstvě na 0.3).

Dále uvažme následující jednoduchou síť:



Trénování probíhá následovně

1. Začneme zpracováním vzoru A. V čase = 0 je aktivita všech neuronů výst. a prostř. vrstvy definována jako 0.3.

Hodnoty vah na počátku necht' jsou tyto:

| | |
|----------------------|-----------------------|
| <u>vstup - střed</u> | <u>střed - výstup</u> |
| a → c: 0.2 | c → d: 0.3 |
| b → c: -0.2 | c → e: -0.1 |
| | c → f: 0.1 |

rekurentní váhy

| | | |
|-------------|-------------|-------------|
| c → c: 0.3 | e → d: 0.1 | f → e: -0.1 |
| d → d: 0.4 | e → e: 0.3 | f → f: 0.3 |
| d → e: -0.1 | e → f: -0.4 | |
| d → f: -0.2 | f → d: -0.2 | |

Spočítáme aktivitu střední vrstvy v čase $t=1$:

$$I_j^{mid}(t) = \sum_{i=1}^2 w_{ij}^{in-mid} \cdot y_i^{in}(t) + \sum_{k=1}^3 w_{kj}^{mid-mid} \cdot y_k^{mid}(t-1)$$

vst. vzor 10

$$I_c(1) = [1 \cdot 0.2 + 0 \cdot (-0.2)] + [0.3 \cdot 0.3] = 0.2 + 0.09 = \underline{0.29}$$

a→c b→c c→c RA

$$f(I_c) = 1 / (1 + e^{-I}) = 1 / (1 + 0.748) = \underline{0.57}$$

Aktivita výstupní vrstvy v čase $t=1$:

$$I_j^{out}(t) = \sum_{i=1}^3 w_{ij}^{mid-out} \cdot y_i^{mid}(t) + \sum_{k=1}^3 w_{kj}^{out-out} \cdot y_k^{out}(t-1)$$

$$I_d(1) = [0.3 \cdot 0.57] + [0.4 \cdot 0.3] + 0.1 \cdot 0.3 + (-0.2) \cdot 0.3 = \underline{0.26}$$

$$f(I_d) = 1 / (1 + e^{-I}) = 1 / (1 + 0.771) = \underline{0.565}$$

$$I_e(1) = [(-0.1) \cdot 0.57] + [(-0.1) \cdot 0.3 + 0.3 \cdot 0.3 + (-0.1) \cdot 0.3] = \underline{-0.027}$$

$$f(I_e) = 1 / (1 + e^{-I}) = 1 / (1 + 0.973) = \underline{0.493}$$

stejným způsobem vypočítáme $I_f(1)$ a $f(I_f)$.

2. Výše popsaným způsobem v bodě 1. vypočítáme potřebné hodnoty I a f pro časové kroky $t=2$ a $t=3$.

Aktivity rekurentních neuronů (R) v čase $t=2$ jsou totožné s aktivitami nerekurentních neuronů v čase $t=1$ (právě bylo spočteno).

Např. aktivita (R) ve vstupní vrstvě v $t=2$ je prostě aktivita neuronu (C) v $t=1$.

3. Spočítáme chybu výstupní vrstvy. Požadovaný výstup pro $t=3$ je 001 (pozor: chybu počítáme pozpátku pro $t=3$, $t=2$ a $t=1$!). $f'(I) = f(I)(1 - f(I))$ kde $f(I)$ jsou hodnoty z předchozích výpočtů.

Chyba výstupu v $t=3$ pro výst. vrstvu:

$$E(N)_j^{out} = \left[\overset{\text{požadovaná}}{y(N)_j} - \overset{\text{skutečná}}{y(N)_j} \right] \cdot \frac{df(I(N)_j)}{dI} \text{ pro } N=3.$$

N je celkový počet kroků ve vzorové sekvenci použité pro trénování; $y(t)$ je aktivita neuronu v čase t z kroků v bodech 1. a 2.

$$E(3)_d = (0 - y(3)_d) \cdot [y(3)_d \cdot (1 - y(3)_d)]$$

Obdobně se spočte $E(3)_e$ a $E(3)_f$.

Chyba prostřední vrstvy v $t=3$:

$$E(N)_j^{mid} = \left[\sum_{i=1}^3 w_{ij}^{mid-out} \cdot E(N)_i^{out} \right] \cdot \frac{df(I(N)_j^{mid})}{dI}$$

Vzorec je stejný jako pro normální back-propagation síť. Index j představuje skutečnost, že existuje více neuronů ve střední vrstvě (1 normální a 3 (R)).

Spočítáme $E(3)_c$.

4. Krok 3. opakujeme pro $t=2$ a $t=1$, výsledky zaznamenejme. Pro chyby výst. vrstvy v $t \neq N$ musíme "zpětně šířit" chyby z pozdějších kroků včetně uvažování členu "požadovaný - skutečný".

Krok $t=2$ získává chyby z $t=3$, a $t=1$ chyby z $t=2$ (proto se musí začít v $t=3$).

Pro libovolný časový krok se chyba výstupu spočte:

$$E(t)_j^{out} = \left[E(t)_j + \sum_{i=1}^3 w_{ij}^{out-out} \cdot E(t+1)_i^{out} \right] \cdot \frac{df(I(t)_j)}{dI}$$

Pro prostřední vrstvu jsou chyby zpětně šířeny ze současného časového okamžiku z výstupu a také z výstupu následného časového kroku:

$$E(t)_j^{mid} = \left[\sum_{i=1}^3 w_{ij}^{mid-out} \cdot E(t)_i^{out} + \sum_{k=1}^1 w_{kj}^{mid-mid} \cdot E(t+1)_k^{mid} \right] \cdot f'(I(t)_j^{mid})$$

Spočítáme tedy chyby pro $t=2$ pro d, e, f (výstupní neurony) a pro c (neuron střední vrstvy).

Stejně tak pro $t=1$ (d, e, f, c).

5. Nyní vypočítáme změny vah pro všechny váhy sítě v čase $t=1$. $y(0)$ pro výstupní a střední váhy je konst = 0.3 (hodnota RA). Pro pozdější časy $t > 0$ jsou váhy mezivrstvy (rekurentní) modifikovány takto:

$$\Delta w(t)_{ij}^{\text{in-mid/mid-out}} = \beta \cdot E(t)_i \cdot y(t)_j$$

Rekurentní váhy (pro prostřední i výstupní vrstvu) se mění takto:

$$\Delta w(t)_{ij}^{\text{mid-mid/out-out}} = \beta \cdot E(t)_i \cdot y(t-1)_j$$

Spočítají se tedy změny vah v $t=1$, ~~pro všechny~~ rekurentní ($a \rightarrow c, b \rightarrow c, c \rightarrow d, c \rightarrow e, c \rightarrow f$) a tekuté ($c \rightarrow c, d \rightarrow d, d \rightarrow e, d \rightarrow f, e \rightarrow d, e \rightarrow e, e \rightarrow f, f \rightarrow d, f \rightarrow e, f \rightarrow f$).

6. Krok 5. se opakuje pro $t=2$ a $t=3$.

7. Kroky 1. až 6. se zopakují pro vzor B.

Pozor: mezi jednotlivými vzory se inicializuje aktivita na RA (0.3)!

8. Nyní se aplikují změny vah na síť. Aplikují se globální kumulativní změny vah (součet změn vlivem vzoru A i B).

Lavinové sítě (avalanche networks)

Sítě založené na modelu „kolovratku“ mají řadu nedostatků, které jim brání v praktickém použití (např. jakmile jsou odstartovány, sekvence proběhne beze změn - chybí variabilita).

Lavinový model je založen na modelu sítě typu outstar. Outstar poskytuje velmi dobrý model klasického podmiňování. V tomto případě se využívá velká hodnota konstanty A (řídí úbytek aktivity neuronu) v rovnici pro výstup neuronu:

$$y_j(t+1) - y_j(t) = \Delta y_j(t) = \underbrace{(-A y_j(t))}_{\text{míra poklesu aktivity}} + I_j(t) + \sum_{i=1}^n \underbrace{w_{ij}(t)}_{\text{velikost váhy}} [y_i(t-\tau) - T] \underbrace{\text{signál}}_{\text{doba přesunu}}$$

$$w_{ij}(t+1) - w_{ij}(t) = \Delta w_{ij}(t) = -F w_{ij}(t) + G y_j(t) [y_i(t-\tau) - T] \underbrace{\text{učicí konstanta}}_{\text{signál}}$$

Výhoda rychlého poklesu aktivity spočívá v tom, že neuron nemůže delekovat nový přichozí signál dříve, než pohasne signál předchozí. Rychlost poklesu aktivity je asociována s časovým rozdílem, který se musí vyskytnout mezi událostmi, aby mohly být rozlišeny jako separátní.

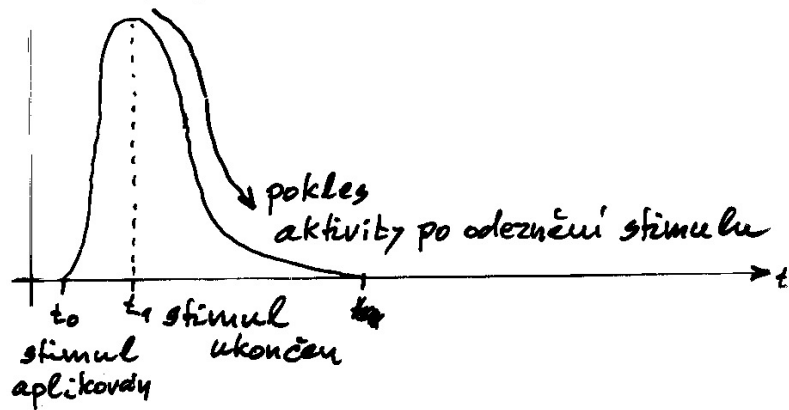
Čím rychlejší pokles, tím citlivější je síť (schopnější rozlišit dvě po sobě jdoucí události).

Je-li cílem naučit se sekvenci vzorů, pak se může použít modifikace modelu outstar/instar.

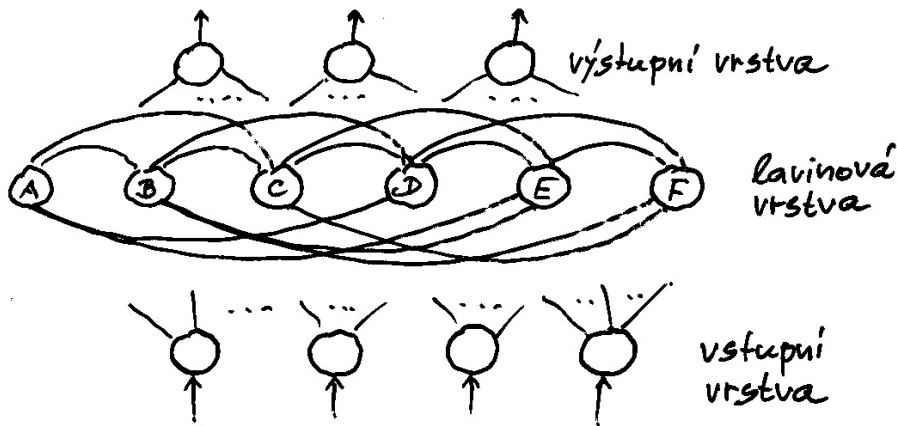
Modifikace probíhá na dvou úrovních:

- změna A (útlumu)
- změna architektury

Odezva neuronu
na měkký podnět



Tzv. lavina (lavinová síť) má následující strukturu, jejímž základem jsou 3 vrstvy:



Neurony lavinové (prostřední) vrstvy pracují jako outstar jednotky vůči výstupní vrstvě.

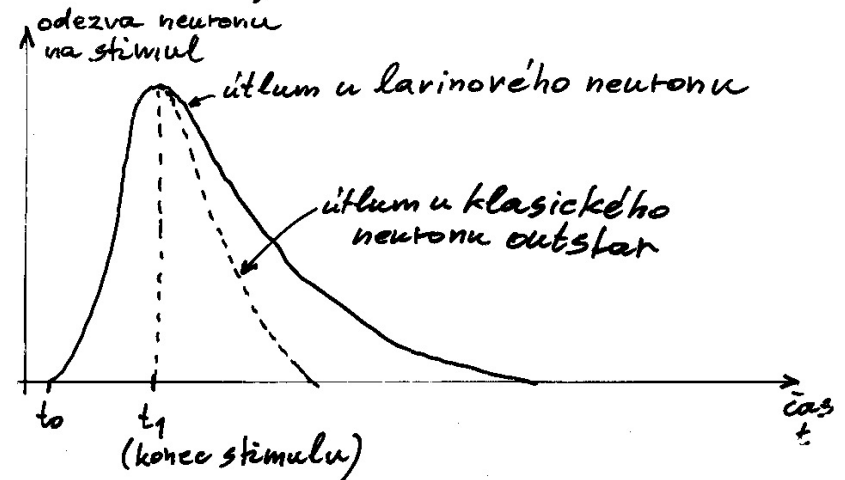
Neurony výstupní vrstvy fungují jako výstupní mřížka u klasické outstar sítě: jsou podmínovány a učí se reprodukovat požadovaný vzor.

Neurony vstupní vrstvy pouze distribuují vstupní hodnoty.

Základem je lavinová vrstva. Ta svým zapojením připomíná Kohonenovu vrstvu s komplexním systémem vzájemného propojení jednotek vrstvy. Avšak tato propojení nerealizují laterální inhibici (jako tomu je u Kohonenovy sítě), nýbrž jsou to propojení adaptivní. Adaptivní propojení dávají síti její časové schopnosti.

Není nutno, aby vzájemné propojení lavinové vrstvy bylo úplné (každý s každým). Netrénovaná síť tak může začínat, ale po natrénování je počet propojení podstatně nižší.

Každá z lavinových jednotek disponuje parametrem A (útlum aktivity), který je relativně malý (avšak stále podstatně větší než parametr útlumu vah F). Znamená to, že útlum aktivity neuronu probíhá přes několik časových kroků



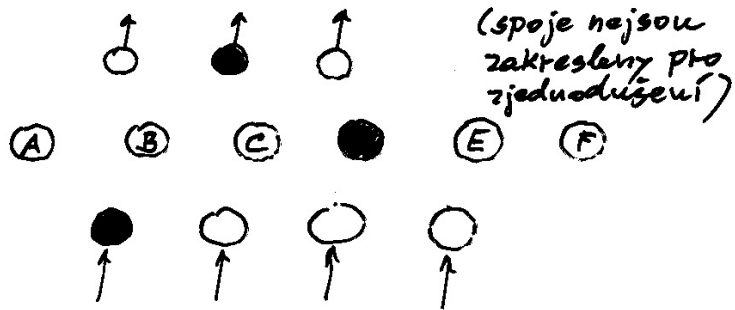
Činnost trénované laviny

Předpokládejme, že síť byla natreénována k reprodukci sekvence tří vzorů, po spuštění vhodným stimulem ze vstupní vrstvy.

Všechny neurony ve vstupní a lavinové vrstvě používají metodu postupného útlumu. Předpokl., že aktivční konstanta je nastavena tak, aby aktivita všech neuronů klesla během 1 časového kroku na polovinu. Dale necht' neurony v lavině mají vstupní práh 0.3 (menší signály jsou ignorovány). Proto každý neuron uchovává svůj výstup podobu 1 časového kroku.

Pro jednoduchost necht' je signál pro spuštění určité sekvence 1000.

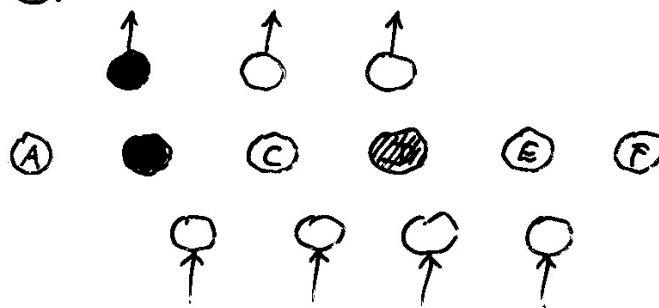
V čase $t=0$ spustí vstupní vrstva aktivitu v lavině. Necht' je aktivován neuron D - tj. jenom D má váhy na svých vstupech takové, že je aktivován vstupem 1000. Aktivita je předána na výstup v čase $t=1$; výstupní neurony jsou trénovány k reprodukci prvního vzoru ze tří. Trénovací pravidlo je totéž jako u outstar.



Neuron D pošle signál nejen do výstupu, ale také do jednotek v lavinové vrstvě.

Tedy: v čase $t=1$ obdrží každá ze zmíněných jednotek $1/2$ signálu ze vstupní vrstvy plus plný signál z D.

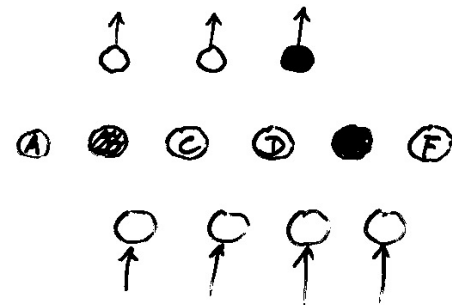
Tato kombinace stimulů aktivuje v čase $t=2$ neuron B (v důsledku nastavení jeho vstupních vah). Tj. B obdrží $1/2$ signálu ze vstupní vrstvy a plný signál z D.



Pozor: výstupní vrstva NEMÁ pomalý útlum, má normální rychlý útlum ~~z~~ modelu outstar!

Proto výstup rychle reprodukuje sekvenci vzorů. (Předpokládáme zde také okamžité šíření signálů.)

Podobně bude aktivován neuron E:



$t=3$: D má ~~aktivitu~~ aktivitu pod prahem 0.3, B má aktivitu poloviční větší čase $t=2$. E nemůže být spuštěn samotným B v čase $t=2$. Atd.

Trénování lavinové sítě

Základem je hebovský algoritmus: váhy mezi neurony (mezi dvojicí neuronů) se posilují, pokud přichází stimulační signál v tužež dobu, kdy je přijímající neuron aktivní. Váhy jsou ovšem také snižovány během každého časového kroku, kdy nedochází k posilování pomocí hebovského učícího členu (viz rovnice pro $\Delta w_{ij}(t)$ a $\Delta w_{ij}(t)$).

Na rozdíl od jiných třívrstevných sítí má lavinová síť tři druhy (soubory) vah pro outstar k natrénování:

- 1) váhy mezi vstupní a prostřední vrstvou,
- 2) váhy mezi prostřední a výstupní vrstvou,
- 3) váhy propojení mezi neurony prostřední vrstvy.

Pro všechny uvedené váhy však platí totéž trénovací pravidlo.

Váhy výstupní vrstvy

Úkolem výstupní vrstvy je reprodukovat správný vzor v každém kroku posloupnosti. Vrstva je trénována vynucováním požadovaného vzoru. Současně s tím získává výst. vrstva aktivní signál z prostřední (lavinové) vrstvy. Pokud je aktivita lavinové vrstvy konsistentní, pak pro dostatečný počet opakování se výstupní vrstva naučí reprodukovat každý krok posloupnosti vzorů správně. Problémem tedy zůstává:

Trénování lavinové vrstvy

Úkolem je, aby se lavinová vrstva naučila spouštět posloupnost („lavinu“) aktivních signálů založenou

pouze na jediném vstupním stimulu (podnětu).

Často používaná procedura trénování lavinové vrstvy probíhá následovně:

Aplikuje se tzv. spouštěcí vstupní vzor na vstupní vrstvu a výsledná aktivita je přes propojení předána lavinové vrstvě.

V lavinové vrstvě se utčí neuron s maximální odzvou na daný (použitý) vstupní vzor. Tento neuron pak slouží jako stimul z lavinové vrstvy směrem k výstupní vrstvě pro použitý vstupní stimul (vzor). Výstup stimulačního neuronu lavinové vrstvy je nastaven na hodnotu +1 (či jakoukoliv velkou hodnotu), zatímco u všech ostatních neuronů lavinové vrstvy je dočasně potlačen jejich výstup.

Protože se používá hebovské trénování, váhy mezi stimulačním neuronem a každým z aktivních neuronů vstupní vrstvy jsou posíleny. Ostatní váhy spojující do stimulačního neuronu jsou mírně oslabeny. Podobně jsou oslabeny všechny váhy spojující do ostatních neuronů lavinové vrstvy.

Z uvedených důvodů v příštím trénovacím kroku je snazší přinutit onen stimulační neuron lavinové vrstvy vygenerovat výstupní signál. Po dostatečném počtu iterací za použití daného vstupního vzoru je stimulační neuron spouštěn automaticky po přiložení daného vzoru na vstup.

Avšak výstup vítězného neuronu (jímž byl v dříve uvedeném příkladu neuron ⑤) není připojen pouze k výstupní vrstvě, ale také k ostatním neuronům lavinové vrstvy. Z důvodu změn vah propojení v lavinové vrstvě (interní propojení vrstvy) poskytují tyto neurony různé odezvy na daný signál.

V časovém kroku $t=1$ již má vstupní signál, vzniklý přiložením vzoru na vstup v kroku $t=0$, poloviční účinek vlivem poklesu.

V kroku $t=1$ však v lavinové vrstvě existuje neuron (v našem příkladu ③), který má maximální odezvu na kombinaci vzniklých stimulů (silný vstup z ⑤ a $1/2$ vstup ze vstupní vrstvy).

Stejně jako v předchozím případě se na tomto neuronu ③ vynutí silný výstup, který posílí spoj mezi ⑤ a ③ a současně jsou oslabeny ostatní vahy.

Je samozřejmě možné místo 1 neuronu s maximálním výstupem použít n neuronů s nejvyšší odezvou, avšak pro účely naší diskuse se omezíme na 1.

Trénování lavinové vrstvy tedy probíhá popsáním způsobem po krocích, s jednoduchou klasickou stimulací pro zesilování jedné vah a oslabování druhých.

Pokud se pro každý krok v časové posloupnosti použije pouze 1 neuron, je kapacita sítě výrazně omezena.

Lavinová síť z n neuronů umí si zapamatovat nejvýš n vzorových kroků. Pro zvýšení kapacity je nutné každý vzorový krok uložit ve více neuronech lavinové vrstvy. V principu se nemění nic, ale na rozdíl od jedno-neuronového případu jsou interní propojení a operace ve vrstvě mnohem složitější a těžko vizualizovatelné.

Trénovací proces pak probíhá stejně, výjimkou je využití m neuronů s nejsilnější odezvou (v okamžitém časovém kroku stimulace) pro posílení interních vah. Vztose ovšem složitost interního propojení.

Další možnou metodou zvýšení paměťové kapacity je rozšíření aktivizačního útlumu v tom smyslu, že se každému neuronu umožní pokles aktivity přes 3 (nebo více) časové kroky, než aktivita klesne pod daný práh T . Znamená to, že aktuální stimul se stává vzorem pro 3 (nebo více) částečně až plně aktivní neurony, a tím se zvýší počet naučených vzorů.

Popsaná trénovací metoda připomíná Kohonovo učení. U Kohonových sítí se používá laterální inhibice k učení neuronu s nejvyšší odezvou na stimul. U lavinových sítí se obdobně interní propojení v rámci jedné vrstvy ovšem nepoužívají pro laterální inhibici, nýbrž pro vytváření jakýchkoli spouštěčů ("promptů") z jednoho časového kroku do druhého, čímž síť umožní reprodakovat různé vzory na základě jednoho vst. stimulu.

Sítě vyššího řádu

Padaline

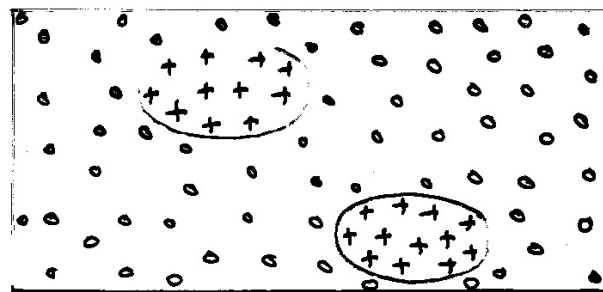
Problém správné kategorizace vstupních vzorů pro případ nelineární separace tříd byl až do nalezení algoritmu back-propagation neřešitelný. BP-sítě byly rozvinuty koncem 70. a počátkem 80. let.

Výjimkou byl systém vytvořený koncem 60. let Donaldem Spechtem - tento systém byl původně nazván polynomičká diskriminační metoda, později se vzil název PADALINE (tj. polynomičká ADALINE, kde adaline je slovo vytvořené z adaptivní lineární neuron).

Padaline je složitější variantou původní lineární adaline. Základní myšlenka vychází z poznání, že lineární rozhodovací plocha (hyper-rovina) neumí v mnoha případech oddělit kategorie, které jsou však oddělitelné velmi dobře pomocí nelineárních hyperploch. Specht nahradil lineární plochu plochou polynomičkou libovolné složitosti, takže je možné se učit jakýkoliv kategorizační problém.

Padaline nevyžaduje, aby byly všechny trénovací příklady uloženy v systému - příklady jsou předkládány po jednom a pak odloženy. Tato vlastnost je důležitá tehdy, když je množství trénovacích příkladů velké nebo je jejich počet neznámý.

Příklad nelineárního oddělení dvou tříd: 0 +



Padaline vytvoří polynom, který oddělí obě kategorie. Polynom se skládá z řady členů:

$$c_{20}x^2 + c_{11}x \cdot y + c_{01}y + c_{00} = 0 \quad (\text{příklad})$$

c jsou konstanty, jejichž indexy odpovídají mocnínám proměnných (c_{20} odpovídá x^2y^0 atd.). Obecně může existovat nekonečně velký počet konstant a členů. Stanovením hodnot konstant je polynom definován (vynechání členu je stanovení $c=0$).

V praxi se počet členů omezuje na několik desítek max.

Nalezení příslušného padaline tedy odpovídá nalezení konstant specifikujících polynomičký separující povrch.

Každý vstupní vektor vzorů \vec{x} se např. může skládat ze 2 prvků x_1 a x_2 . Dále nechtě trénovací soubor obsahuje 100 vzorů (vektorů), z nichž každý patří do jedné ze dvou kategorií (60% do A a 40% do B).

Obecný formát pro dvojitřmětňý vstup:

$$C_{z_1, z_2} = \frac{1}{z_1! z_2! s^{2h}} \left[\frac{1}{m} \sum_{i=1}^m x_{A_{i1}}^{z_1} x_{A_{i2}}^{z_2} e^{E_{A_i}} - \frac{K}{n} \sum_{i=1}^n x_{B_{i1}}^{z_1} x_{B_{i2}}^{z_2} e^{E_{B_i}} \right]$$

z_1 a z_2 jsou příslušné mocniny složek vst. vektoru \vec{x}
(v našem případě pro jednoduchost jen dvojsložkového).

$$h = z_1 + z_2$$

Procedura učení

stanov konstantu vyhlazení (s);

spočítej K pro trénovací množinu;

for každý vzor v trénovací množině

spočítej čtverec délky vektoru (L);

spočítej exponenciální člen $e^{(L/s^2)}$ pro vektor;

for každou konst. c_{ij} , která má být spočítána

if vzor $\in A$ přidej přínos vzoru do prvního členu sumárního vzorce;

else /* vzor je B */ přidej přínos do druhého členu sumárního vzorce;

end if;

do next konstanta;

do next vzor;

po ukončení uloř hodnoty konstant.

Procedura klasifikace

for každý neznámý ~~vst~~ vektor \vec{x}
použij vypočtené konstanty pro výpočet $P(\vec{x})$;

if $P(\vec{x}) > 0$ přiřad vzor kategorii A;

else přiřad vzor kategorii B;

end if;

do next vektor \vec{x} .

Příklad

Mějme dvě kategorie, A a B:

$$A_1 = (1, 3)$$

$$B_1 = (3, -1)$$

$$A_2 = (3, 1)$$

$$B_2 = (1, -1)$$

$$A_3 = (-1, 3)$$

$$B_3 = (1, -3)$$

$$A_4 = (-3, 1)$$

$$B_4 = (3, -3)$$

$$A_5 = (-3, -1)$$

$$A_6 = (-1, -3)$$

Spočítáme globální hodnoty: L pro každý vzor a E pro každý vzor. Necht $s = 2$. Určime $K = 4/6 = 0,67$.

$$\text{Pak pro } A_1: L = 1^2 + 3^2 = 1 + 9 = 10$$

$$E_{A_1} = e^{(-10/8)} = 0.2865$$

Podobně spočteme $(A_2, \dots, A_6, B_1, \dots, B_4)$.

Protože problém, ač silně nelineární, je relativně jednoduchý, použijeme 6 konstant.

Vstupní vzory mají formu (x_1, x_2) . Hledaný polynom má tvar:

$$P(x_1, x_2) = C_{00} + C_{10}x_1 + C_{01}x_2 + C_{11}x_1x_2 + C_{20}x_1^2 + C_{02}x_2^2$$

Hledáme tedy 6 polynomičeských konstant: $C_{00}, C_{10}, C_{01}, C_{11}, C_{20}, C_{02}$.

Pro výpočet použijeme obecný vztah:

$$C_{z_1 z_2} = \frac{1}{z_1! z_2! s^{2h}} \left[\frac{1}{m} \sum_{i=1}^m x_{A_{i1}}^{z_1} x_{A_{i2}}^{z_2} e^{E_{A_i}} - \frac{K}{n} \sum_{i=1}^n x_{B_{i1}}^{z_1} x_{B_{i2}}^{z_2} e^{E_{B_i}} \right]$$

kde $h = z_1 + z_2$, $m = 6$, $n = 4$, $K = 0.67$.

Spočítáme např. C_{11} (ostatní výpočty jsou ponechány jako cvičení).

Obecná forma modifikovaná pro C_{11} :

$$C_{11} = \frac{1}{1! 1! s^4} \left[\frac{1}{m} \sum_{i=1}^m x_{A_{i1}}^1 x_{A_{i2}}^1 e^{E_{A_i}} - \frac{K}{n} \sum_{i=1}^n x_{B_{i1}}^1 x_{B_{i2}}^1 e^{E_{B_i}} \right]$$

$$z_1 = z_2 = 1, h = z_1 + z_2 = 1 + 1 = 2, s^{2h} = s^4 \Rightarrow$$

$$1/1! 1! s^4 = 1/16 = 0.0625 \Rightarrow$$

$$C_{11} = 0.0625 \left[\frac{1}{6} \sum_{i=1}^6 x_{A_{i1}} x_{A_{i2}} e^{E_{A_i}} - \frac{0.67}{4} \sum_{i=1}^4 x_{B_{i1}} x_{B_{i2}} e^{E_{B_i}} \right]$$

Spočítáme přínos každé trénovací instance pro C_{11} :

$$A_1: x_1 = 1, x_2 = 3, L = 10, e^{E_{A_1}} = e^{-10/8} = 0.2865$$

Přínos pro druhý sumáčnický člen není žádný, pro první sumu tedy je:

$$x_1 x_2 e^{E_{A_1}} = (1) \cdot (3) \cdot (0.2865) = 0.8595$$

Podobně pro A_2 :

$$A_2: x_1 = 3, x_2 = 1, L = 10, e^{E_{A_2}} = e^{-10/8} = 0.2865$$

Není žádný přínos pro druhý sumáčnický člen, proto:

$$x_1 x_2 e^{E_{A_2}} = (3) \cdot (1) \cdot (0.2865) = 0.8595$$

Ukázaným způsobem se vypočtou přínosy A_3, \dots, A_6 pro C_{11} a dále B_1, \dots, B_4 pro C_{11} (zde není přínos pro první sumáčnický člen).

Obdobně určíme přínosy A_i a B_j ($i = 1, \dots, 6$, $j = 1, \dots, 4$) pro ostatní polynomičeské konstanty $C_{00}, C_{10}, C_{01}, C_{20}, C_{02}$.

Spočteme průměr pomocí $1/m = 1/6$ a vážený průměr pomocí $K/n = 0.67/4$.

Vypočítáme rozdíl ^{průměrovaných} sum a výsledky násobíme faktorem $1/z_1! z_2! s^{2h}$.

Tim získáme hodnoty konstant c_{ij} polynomu. Body kategorie A a B zakreslíme do grafu. Zakreslíme polynom a ověříme, zda jsou A a B polynomem odděleny.

Nakonec ověříme klasifikační schopnosti pro množinu testovacích dat:

- $c_1 = (0, 0) \Rightarrow P(0, 0) = ?$ Kategorie = ?
 $c_2 = (1.5, -1) \Rightarrow P(1.5, -1) = ?$ Kategorie = ?
 $c_3 = (-2, 0) \dots$
 $c_4 = (0, 1) \dots$
 $c_5 = (2, 1) \dots$
 $c_6 = (0, -1) \dots$

(Rozhodující pro klasifikaci je, zda $P(\cdot, \cdot) \geq 0$)

Mnohazměrný PADALINE

Většina reálných problémů má vicesložkový vektor, tj. počet dimenzí větší než 2. Mnohazměrná procedura je identická s dvourozměrnou; přechod k obecné formě není v principu obtížný.

Jeden problém však stále zůstává: polynom může mít neomezený počet členů/konstant. Experimenty ukazují, že většinou postačují max. kvadratické členy a prakticky nikdy nejsou členy vyššího stupně než 3.

Často vycházejí konstanty c_{ij} velmi malé. Pokud se blíží nule, lze je za nulové považovat a tak redukovat množství výpočtů (redukce se týká klasifikace, nikoliv trénování - zde je nutno spočítat vše).

Obecně větší počet konstant a složitější polynomy dávají jemnější diskriminační schopnosti, avšak za cenu značné výpočetní náročnosti.

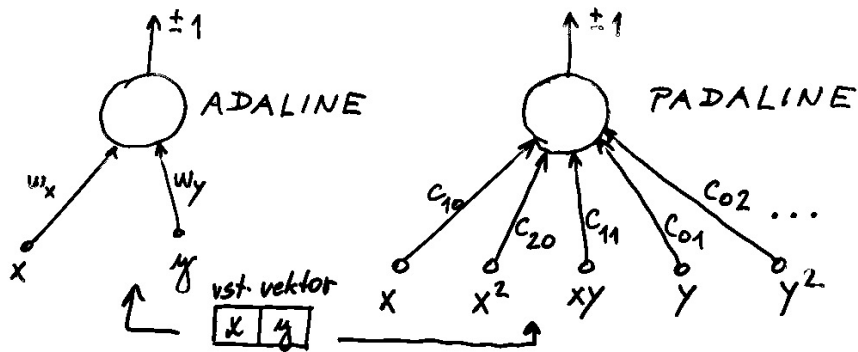
Obecný vzorec pro výpočet c_{z_1, z_2, \dots, z_p} :

$$c_{z_1, z_2, \dots, z_p} = \frac{1}{z_1! z_2! \dots z_p! s^{2h}} \cdot \left[\frac{1}{m} \sum_{i=1}^m X_{A_{i1}}^{z_1} X_{A_{i2}}^{z_2} \dots X_{A_{ip}}^{z_p} e^{E_{A_i}} - \frac{K}{m} \sum_{i=1}^K X_{B_{i1}}^{z_1} X_{B_{i2}}^{z_2} \dots X_{B_{ip}}^{z_p} e^{E_{B_i}} \right]$$

Jedním ze způsobů, jak redukovat počet konstant c , které je nutno spočítat během tréninku, je použití menší vyhlazovací konstanty s . Menší s totiž poskytuje polynomu složitější rozhodovací povrch pro tentýž počet členů. Větší s naopak způsobuje "hladší" povrch. Použitím malého s se kompenzuje menší počet členů polynomu.

Padaline je příkladem neuronové sítě "vyššího řádu". Znamená to, že zpracovává vstupní vektory pomocí nelineární formy. Pro dvousložkový vstupní vektor může polynom obsahovat pouze členy $x_1, x_1^2, x_1 x_2, x_2^2, x_2$.

Odpovídající schéma pro lineární a nelineární adaptivní jednotku (ADALINE a PADALINE):



Výpočet polynomičeských konstant odpovídá hledání vstupních vah během tréninku.

Multikategoriální PADALINE

Padaline umí zpracovat i případy, kdy je nutno přiřadit neznámý počet vstupních vzorů různým kategoriím, jejichž počet > 2 , tj. jde o odlišný případ než $A/\rightarrow A$.

V tomto případě existuje několik jednotek PADALINE, každá z nich má za úkol prohlížet vstupní vzor pro jednu s kategorií.

Výpočet konstant pro každou jednotku PADALINE se poněkud zjednoduší. Obecná forma vzorce:

$$C_{z_1 z_2 \dots z_p}^A = \frac{1}{z_1! z_2! \dots z_p! S^{2h}} \left[\frac{1}{m} \sum_{i=1}^m X_{A i 1}^{z_1} X_{A i 2}^{z_2} \dots X_{A i p}^{z_p} e^{E_{A i}} \right]$$

$C_{z_1 z_2 \dots z_p}^A$: horní index A znamená, že se jedná o konstantu pro polynom, který odděluje vzory do $A/\rightarrow A$ kategorií.

Hlavní rozdíl je v tom, že chybí člen pro B, protože existuje samostatný polynom pro tožhodnotu typu $B/\rightarrow B$.

Pokud by problém vyžadoval výpočet souboru 25 konstant k určení rozumné oddělovací hyperplochy a vstupní data by mohla náležet do některé z 6 kategorií, pak musí existovat 6 souborů po 25 konstantách c.

Viděno formou fyzické sítě to znamená 6 neuronů typu PADALINE, každý s 25 vstupy a 25 váhami těchto vstupů.

Každá jednotlivá jednotka PADALINE je trénována stejně jako v předchozí kapitole. Poté je začleněna do jediného velkého systému (viz následující obrázek).

Neznámý vstupní vzor je současně předložen všem jednotkám PADALINE. Jednotka s nejsilnějším výstupem pak reprezentuje správnou kategorii pro daný vzor.

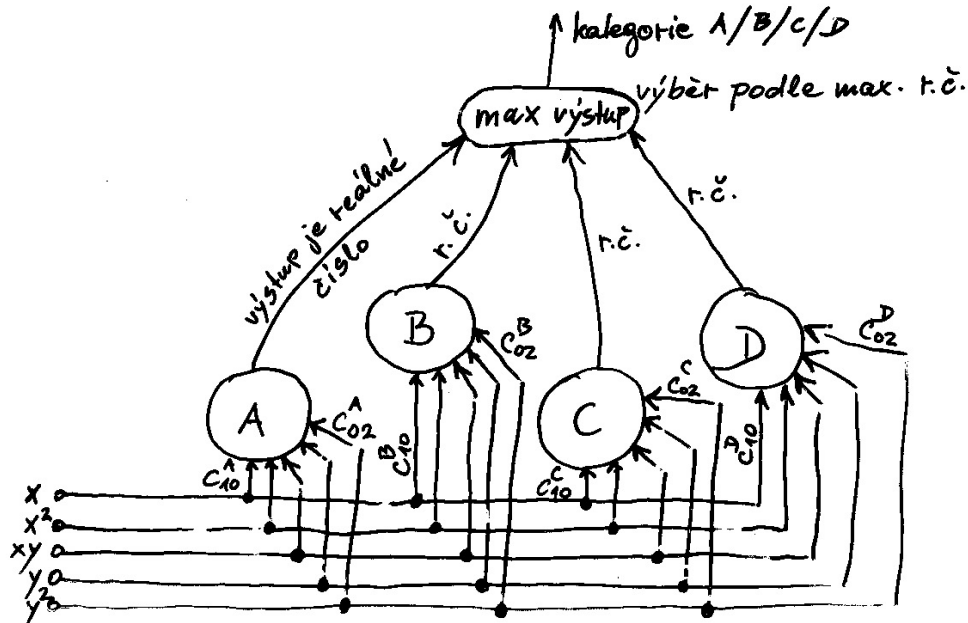
Algoritmus PADALINE má „rozumnou“ výpočetní náročnost. Je snadno implementovatelný a velmi účinný při klasifikaci neznámých vzorů. Umožňuje daleko více, než ADALINE.

Výpočet dodatečných konstant polynomu lze získávat stále přesněji a přesněji aproximaci rozhodovacího povrchu, takže se lze přiblížit libovolně rozhodovací spolehlivosti, omezené pouze časem a výpočetními možnostmi.

Další výhodou PADALINE je, že trénovací data jsou zpracována tak, že po předložení vektoru je možné jej uadde „zapomenout“. Uchovávají se pouze konstanty c . Po jediném průchodu je síť připravena zpracovávat neznámé vzory.

A nakonec klasifikační proces je velmi jednoduchý a rychlý: spočítá se hodnota polynomu pro každou jednotku a zjistí se maximální hodnota.

Příklad čtyřkategoriatřluko PADALINE:



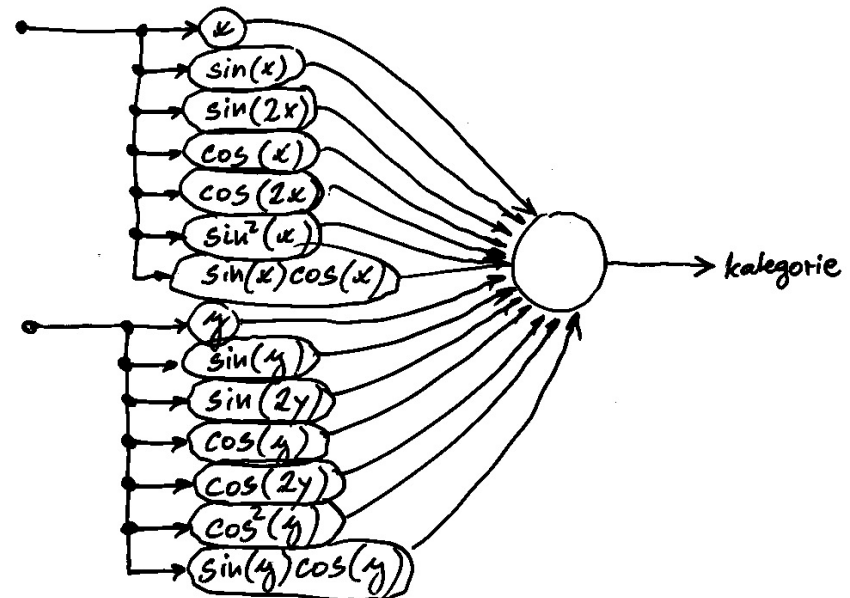
Sítě s funkcionálními spoji

Padaline je pouze jedním příkladem sítě vyššího řádu. Do této kategorie spadají i jiné typy sítí. Jednou z nich, často využívanou v komerčních aplikacích, je síť s funkcionálními spoji, kterou vyvinul Yoh-han Pao.

Tato síť se strukturou podobá PADALINE s tou výjimkou, že vstupní elementy nejsou omezeny na jednoduché mocninné funkce vstupů.

V síti s funkcionálními spoji se mohou vyskytovat funkce sinus, cosinus a další nelineární funkce.

Například pro dvourozměrný případ:



Sítě s funkcionálními spoji získaly svůj název podle toho, že spoje jsou vlastně funkcemi aplikovanými i na konkrétní vstupní hodnoty. Většina neuronových sítí používá spoje pouze jako "lineární routy", které jenom posílají signály mezi body sítě. Váhy modifikující signály jsou aplikovány pouze v okamžiku, když signál přijde do styčného bodu spoje a neuronu.

U funkcionálních spojů se jedná o jiný koncept: přímo spoje fungují jako funkce aplikované na signály.

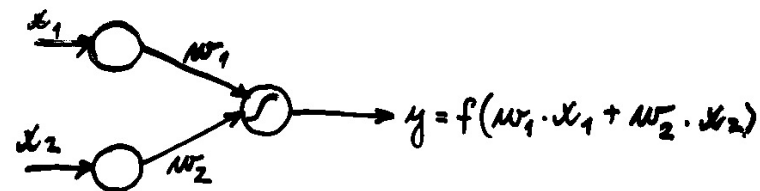
Sítě s funkcionálními spoji, stejně jako ostatní sítě vyššího řádu, poskytují síti bohatší soubor informací. Takto je možné lépe vyjádřit vztahy mezi vstupními a výstupními vzory. Pro mnoho obtížných problémů lze takto dosáhnout snadnějšího naučení sítě.

Další výhodou těchto typů sítí je, že vystačí jen s 1 nebo 2 vrstvami. (BP vyžaduje mezivrstvu pro vyjádření složitých vztahů mezi elementy vstupních vzorů.)

Sítě s funkcionálními spoji obvykle používají pro trénování Δ -pravidlo. Je rovněž možné je učit pomocí Kohonenova pravidla (jeho obdoby).

Problémem bývá učít funkce. Zde záleží značně na porozumění řešeného problému a na tom, jak lze nejlépe aproximovat závislost vstup-výstup.

Fuzzy perceptron



Všechny váhy a vstupní signály jsou reálná čísla. Oba vstupní neurony nemění vstup (výstup = vstup). Signál x_i interaguje s vahou w_i :

výstup:
$$p_i = w_i x_i, \quad i=1,2$$

Informace p_i je agregována sečítáním:

$$net = p_1 + p_2 = w_1 x_1 + w_2 x_2$$

Přenosová funkce f (bývá většinou sigmoida):

$$f(x) = \frac{1}{1 + e^{-x}}$$

Výstup y :

$$y = f(net) = f(w_1 x_1 + w_2 x_2)$$

Popsaný perceptron (jednoduchá NN), který využívá násobení, sečítání, sigmoidální funkci f , se užívá jako standardní (regulární) neuronová síť.

Použijeme-li alternativní operátory, např. t -normy a t -konormy, pro kombinování hodnot vstupních dat, dostáváme tzv. hybridní perceptron (hybridní neuronovou síť). Tyto modifikace vedou k fuzzy neuronové architektuře založené na fuzzy aritmetických operacích.

Necht' vstupy (obvykle stupně příslušnosti) a váhy jsou z jednotkového intervalu: $x_1, x_2, w_1, w_2 \in [0, 1]$. Hybridní NN nepoužívá $*$, $+$, f_{sigm} neboť výsledky těchto operací mohou být mimo $[0, 1]$.

Hybridní NN je NN s ostrými signály a vahami a ostrou přenosovou funkcí, avšak x_i a w_i jsou kombinovány pomocí t -normy a t -konormy (či jinou spojitou operací); p_1 a p_2 jsou agregovány pomocí t -normy, t -konormy (či jinou spojitou funkcí); f může být libovolná spojitá funkce ze vstupu na výstup.

Všechny vstupy, výstupy a váhy HNN jsou reálná čísla z jednotkového intervalu $[0, 1]$.

Element HNN se nazývá fuzzy neuron.

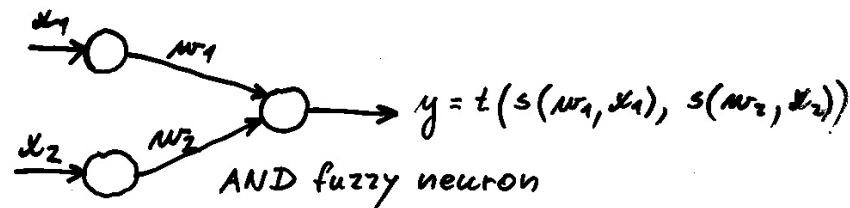
AND fuzzy neuron

Signál x_i a váha w_i jsou kombinovány triangulární konormou s : $p_i = s(w_i, x_i)$, $i = 1, 2$
 Vstupní informace p_i je agregována triangulární normou t : $y = \text{AND}(p_1, p_2) = t(p_1, p_2) =$

$$= t(s(w_1, x_1), s(w_2, x_2))$$

Je-li $t = \min$, $s = \max$ potom AND-neuron realizuje min-max kompozici:

$$y = \min \{ w_1 \vee x_1, w_2 \vee x_2 \}$$



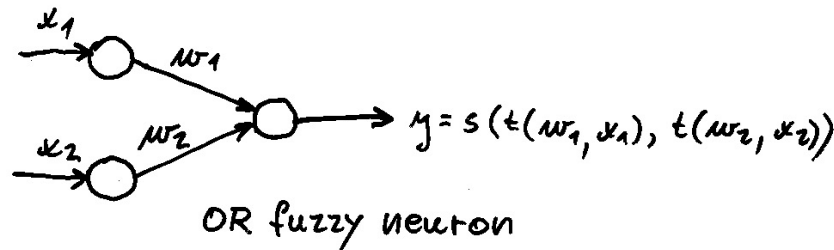
OR fuzzy neuron

$$p_i = t(w_i, x_i), i = 1, 2$$

$$y = \text{OR}(p_1, p_2) = s(p_1, p_2) = s(t(w_1, x_1), t(w_2, x_2))$$

Pro $t = \min$, $s = \max$ získáme OR-neuron realizující max-min kompozici:

$$y = \max \{ w_1 \wedge x_1, w_2 \wedge x_2 \}$$



Implementace fuzzy pravidel IF-THEN pomocí
trénovatelných neuronových sítí

IF $x = A_i$ THEN $y = B_i$: R_i

A_i, B_i ... fuzzy čísla, $i = 1, 2, \dots, n$ (počet pravidel)

Trénovací množina: $\{(A_1, B_1), \dots, (A_n, B_n)\}$
vstup výstup

IF $x = A_i$ AND $y = B_i$ THEN $z = C_i$: R_i

Trénovací množina: $\{(A_i, B_i), C_i\}$, $1 \leq i \leq n$
vstup výstup

IF $x = A_i$ AND $y = B_i$ THEN $t = C_i$ AND $s = D_i$

Trénovací množina: $\{(A_i, B_i), (C_i, D_i)\}$, $1 \leq i \leq n$
vstup výstup

Použití standardní BP-NN

(1) Metoda Umano - Ezawa: fuzzy množina je reprezentována konečným počtem hodnot funkce příslušnosti.

$[\alpha_1, \alpha_2]$... support všech A_i (plus všech A_i'), které se mohou vyskytnout na vstupu.

$[\beta_1, \beta_2]$... support všech B_i (plus B_i'), které se mohou objevit na výstupu. $i = 1, \dots, n$

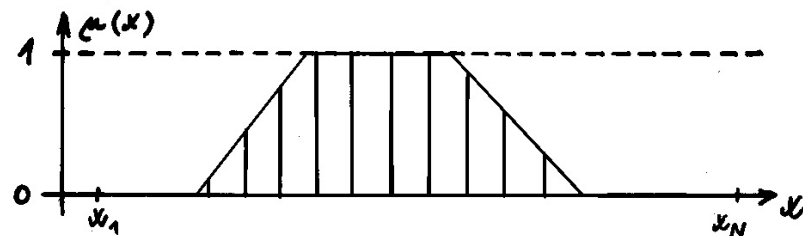
$M \geq 2, N \geq 2$ kladná celá čísla

$x_j = \alpha_1 + (j-1)(\alpha_2 - \alpha_1)/(N-1)$ $1 \leq j \leq N$

$y_i = \beta_1 + (i-1)(\beta_2 - \beta_1)/(M-1)$ $1 \leq i \leq M$

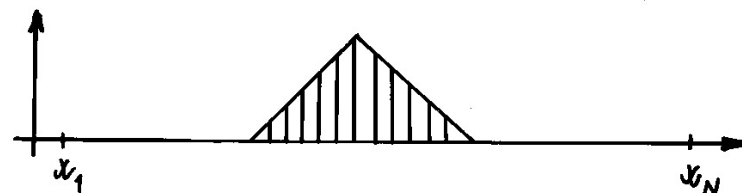
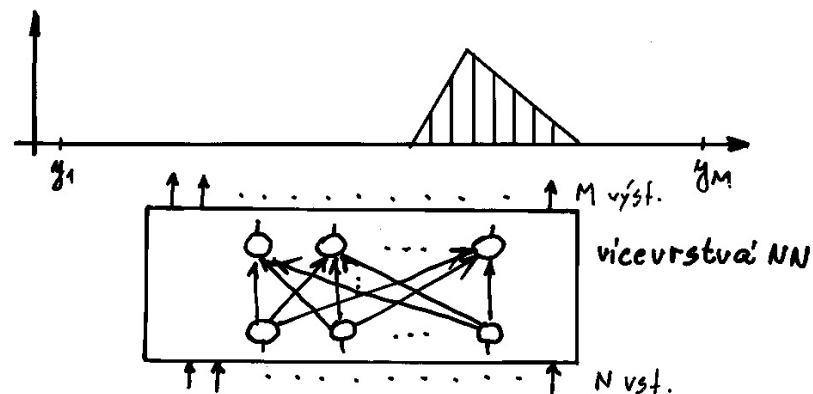
Diskrétní verze spojité trénovací množiny:

$\{(A_i(x_1), \dots, A_i(x_N)), (B_i(y_1), \dots, B_i(y_M))\}$
 $i = 1, 2, \dots, n$



Nechť $a_{ij} = A_i(x_j)$, $b_{ij} = B_i(y_j)$

Pak lze fuzzy NN považovat za ostrou NN s N vstupy a M výstupy, kterou lze trénovat BP:



Příklad: Necht' fuzzy znalostní báze sestává ze 3 pravidel:

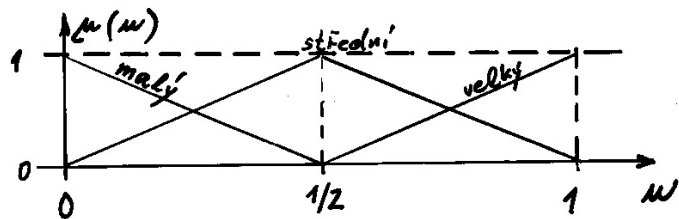
- R_1 : IF x ^{je} malý THEN y ^{je} negativní
 R_2 : IF x ^{je} střední THEN y ^{je} přibližně-nulový
 R_3 : IF x ^{je} velký THEN y ^{je} pozitivní

kde fuzzy množiny malý, ... jsou definovány takto:

$$\mu_{\text{malý}}(u) = \begin{cases} 1-2u & \text{pro } 0 \leq u \leq 1/2 \\ 0 & \text{jinak} \end{cases}$$

$$\mu_{\text{velký}}(u) = \begin{cases} 2u-1 & \text{pro } 1/2 \leq u \leq 1 \\ 0 & \text{jinak} \end{cases}$$

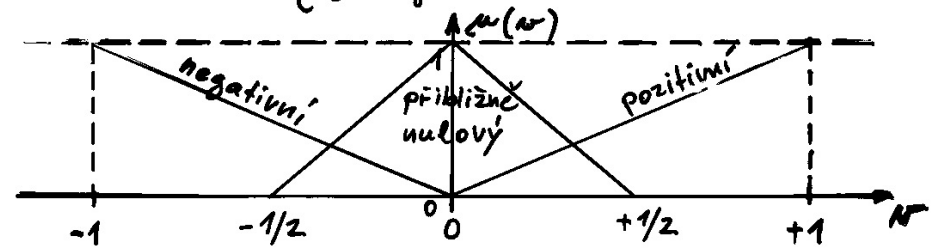
$$\mu_{\text{střední}}(u) = \begin{cases} 1-2|u-1/2| & \text{pro } 0 \leq u \leq 1 \\ 0 & \text{jinak} \end{cases}$$



$$\mu_{\text{negativní}}(v) = \begin{cases} -v & \text{pro } -1 \leq v \leq 0 \\ 0 & \text{jinak} \end{cases}$$

$$\mu_{\text{přibližně-nulový}}(v) = \begin{cases} 1-2|v| & \text{pro } -1/2 \leq v \leq 1/2 \\ 0 & \text{jinak} \end{cases}$$

$$\mu_{\text{pozitivní}}(v) = \begin{cases} v & \text{pro } 0 \leq v \leq 1 \\ 0 & \text{jinak} \end{cases}$$



Trénovací množina odvozená z fuzzy znalostní báze:

{(malý, negativní), (střední, přibl.-nulový), (velký, pozitivní)}

$[0, 1]$ obsahuje suport všech fuzzy množin, které se mohou objevit na vstupu (předpoklad).

$[-1, 1]$ obsahuje obdobně suport všech fuzzy množin, které se mohou objevit na výstupu.

$\beta_2 - \beta_1 = (+1) - (-1) = 2$
 Necht' $M = N = 5$ a $x_j = (j-1)/4$ pro $1 \leq j \leq 5$ a
 $y_i = -1 + (i-1) \cdot 2/4 = -1 + (i-1)/2 = -3/2 + i/2$ pro
 $1 \leq i \leq M$ a $1 \leq j \leq N$.

| | |
|--------------|--------------|
| $x_1 = 0$ | $y_1 = -1$ |
| $x_2 = 0.25$ | $y_2 = -0.5$ |
| $x_3 = 0.5$ | $y_3 = 0$ |
| $x_4 = 0.75$ | $y_4 = 0.5$ |
| $x_5 = 1$ | $y_5 = 1$ |

Diskrétní verze spojité trénovací množiny:

- {(a₁₁, ..., a₁₅), (b₁₁, ..., b₁₅)}
- {(a₂₁, ..., a₂₅), (b₂₁, ..., b₂₅)}
- {(a₃₁, ..., a₃₅), (b₃₁, ..., b₃₅)}

kde $a_{ij} = \mu_{\text{malý}}(x_j)$, $a_{2j} = \mu_{\text{střední}}(x_j)$, $a_{3j} = \mu_{\text{velký}}(x_j)$
 $b_{1i} = \mu_{\text{neg.}}(y_i)$, $b_{2i} = \mu_{\text{př.-nul.}}(y_i)$, $b_{3i} = \mu_{\text{pos.}}(y_i)$
 pro $j = 1, \dots, 5$ a $i = 1, \dots, 5$.

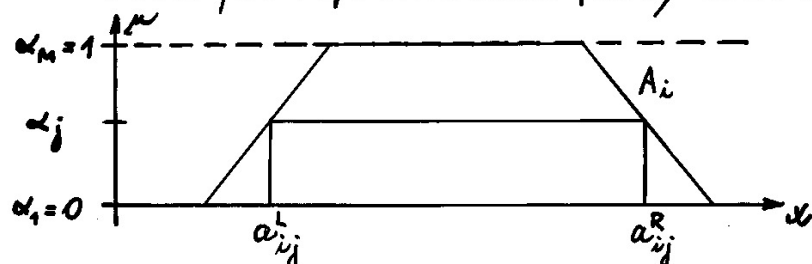
Vyjádřeno maticově dle předchozího:

$$\{(1, 0.5, 0, 0, 0), (1, 0.5, 0, 0, 0)\}$$

$$\{(0, 0.5, 1, 0.5, 0), (0, 0, 1, 0, 0)\}$$

$$\{(0, 0, 0, 0.5, 1), (0, 0, 0, 0.5, 1)\}$$

(2) Uehara a Fujise použili konečný počet α -řezů pro reprezentaci fuzzy čísel:



α -řez $[A]^{\alpha}$

$$[A]^{\alpha} = \{x \in X \mid A(x) \geq \alpha\}$$

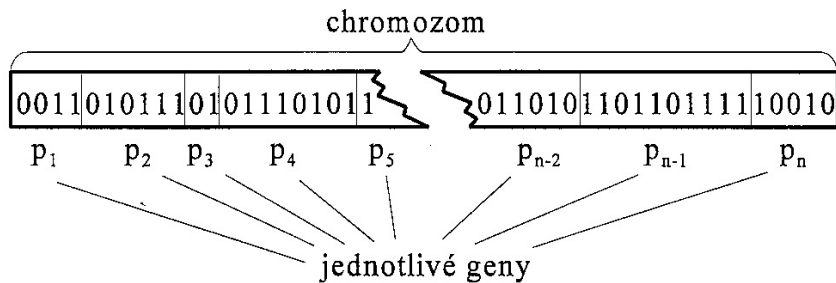
$$0 \leq \alpha \leq 1$$

FUZZY-GENETICKÉ MODELOVÁNÍ

- Fuzzy pravidla lze v mnoha případech stanovit přímo jako reprezentaci expertních znalostí (např. řízení procesů).
- Existuje však mnoho situací, kdy pravidla známa nejsou.
- Fuzzy systémy jsou obecně schopny modelovat nelineární mapování a v tomto smyslu jsou podobny dopředným umělým neuronovým sítím. Fuzzy systémem rozumíme funkci mapující vstupní reálné hodnoty na reálné výstupní hodnoty. Fuzzy systémy pracují jako universální aproximátory (Kosko, 1992) tak, že mohou aproximovat obecné nelineární funkce s libovolně zvoleným stupněm přesnosti (podobně jako neuronové sítě). Na rozdíl od dopředných neuronových sítí, které splňují klasický Stone-Weierstrassův teorém, je důkaz pro fuzzy systémy založen na pokrytí univers fuzzy množinami.
- Fuzzy systémy mohou být aplikovány na stejně široké spektrum problémů jako dopředné neuronové sítě. Praktické aplikace fuzzy systémů potřebují nějaké prostředky k nastavení parametrů systému tak, aby výstup systému odpovídal trénovacím datům.
- Fuzzy systémy se k modelování používají místo dopředných neuronových sítí tehdy, když fungování neuronové sítě jako "černé skříňky" představuje problém. K tomu může dojít např. tehdy, když testovací data jsou velmi odlišná od trénovacích a je obtížné predikovat výstup natrénované sítě – hodnoty vah neuronových propojení neumožňují stanovit, co síť s daty udělá.
- Fuzzy systém oproti tomu je "průhledná skříňka", neboť analýzou pravidel lze stanovit, jaký výstup vznikne odezvou na zadaný vstup. Kauzálním vztahům mezi antecedentem a konsekventem lze porozumět mnohem snadněji než mapování určené hodnotami vah.
- Fuzzy systémy také umožňují analýzu distribuce trénovacích dat vůči testovacím. Je-li obojí radikálně odlišné, lze očekávat, že model nebude dávat spolehlivé předpovědi. Naopak, podobná rozdělení opravňují k domněnce, že predikce bude dostatečně přesná.

Adaptace parametrů genetickou optimalizací

- Fuzzy systémy mají řadu parametrů: počet, tvar a rozmístění fuzzy množin pro vstupní a výstupní proměnné, váhy pravidel (relativní důležitost pravidel), aj.
- Adaptace probíhá formou minimalizace chybové funkce. Na rozdíl od neuronových sítí, kde se váhy vyskytují v analytických výrazech a lze počítat derivace chybové funkce např. pro BP-algoritmus, u fuzzy systémů tomu tak není. Proto bývá nebytné použít neanalytické prostředky – jedním z nich je genetická optimalizace založená na aplikaci genetických algoritmů (GA).
- GA nepracují s originální formou dat, nýbrž s jejich formou zakódovanou do řetězců, velmi často binárních, tzv. chromozomů. Chromozom sestává z částí, tzv. genů, což jsou parametry systému, které se optimalizují – tj. hledá se chromozom složený z optimálních hodnot parametrů.



- Pro genetickou optimalizaci jsou k dispozici různé operátory pro práci s chromozomy, zejména:
 - ☞ generování populace chromozomů
 - ☞ křížení chromozomů
 - ☞ mutace chromozomů
 - ☞ výběr chromozomů určených pro další generaci
 - ☞ aj. dle potřeby

Fuzzy k-NN (nejbližší soused)

k-NN patří k jednoduchým a používaným klasifikačním algoritmům.

Fuzzy k-NN je zobecnění klasického k-NN.

V principu k-NN pracuje tak, že pro danou množinu klasifikovaných dat určí klasifikaci podle třídy nejbližších k sousedů.

Hlavní výhodou je výpočetní jednoduchost.

Existují dva hlavní problémy:

1) Pro klasifikaci se považuje každý ze „sousedů“ za stejně významného – vzdálenější stejně jako blízký.

2) Algoritmus pouze vstupním datům přiřadí třídu, ale neurčí „sílu příslušnosti“ do této třídy.

Uvedená dvě omezení motivovala rozvoj fuzzy verze algoritmu k-NN:

Fuzzy k-NN přiřazuje vstupu x vektor hodnot stupňů příslušnosti $(\mu_{c_1}(x), \mu_{c_2}(x), \dots, \mu_{c_l}(x))$,

kde l je celkový počet tříd. Stupně příslušnosti se počítají pomocí následujícího vzorce:

$$\mu_{c_i}(x) = \frac{\sum_{j=1}^k \mu_{c_j}(x_j) \frac{1}{\|x-x_j\|^{2/(m-1)}}}{\sum_{j=1}^k \frac{1}{\|x-x_j\|^{2/(m-1)}}},$$

kde x_1, x_2, \dots, x_k označuje k nejbližších sousedů x .

Fuzzy k -NN tedy obsahuje dva jednoduché kroky:

1. Najdi k nejbližších sousedů x .
2. Vypočítej stupně příslušnosti do jednotlivých tříd dle výše uvedeného vztahu pro $\mu_{c_i}(x)$.

Parametr m funguje jako určitá váha. Určuje, jak silně je významná vzdálenost při výpočtu stupně příslušnosti. Jak se $m \rightarrow \infty$, člen $\|x-x_j\|^{2/(m-1)} \rightarrow 1$ bez ohledu na vzdálenost, takže sousedé x_j jsou váhováni rovnoměrně. Jestliže $m \rightarrow 1$, bližší sousedé jsou váhováni jako významnější než vzdálenější. Takto se redukuje vliv vzdálenějších sousedů na hodnotu stupně příslušnosti.

Například fuzzy 10-NN se pro $m=0.01$ chová skoro stejně jako fuzzy 1-NN.

Optimalizace fuzzy pravidel pomocí NN (s BP)

Uvažme příklad s ohodnocením portfolia, kde jsou dána následující 3 pravidla:

R_1 : IF $x_1 = L_1$ AND $x_2 = L_2$ AND $x_3 = L_3$ THEN $pv = VB$

R_2 : IF $x_1 = H_1$ AND $x_2 = H_2$ AND $x_3 = L_3$ THEN $pv = B$

R_3 : IF $x_1 = H_1$ AND $x_2 = H_2$ AND $x_3 = H_3$ THEN $pv = S$

kde x_1, x_2, x_3 označují poměr kursů mezi US\$ a DEM, US\$ a SEK, US\$ a FIM.

Nechť jsou dány fuzzy množiny $L_1 = \text{"US\$/DEM je nízký"}$, $H_1 = \text{"US\$/DEM je vysoký"}$, $L_2 = \text{"US\$/SEK je nízký"}$, $H_2 = \text{"US\$/SEK je vysoký"}$, $L_3 = \text{"US\$/FIM je nízký"}$, $H_3 = \text{"US\$/FIM je vysoký"}$:

$$L_1(t) = \frac{1}{1 + e^{b_1(t-c_1)}} \quad H_1(t) = \frac{1}{1 + e^{-b_1(t-c_1)}}$$

$$L_2(t) = \frac{1}{1 + e^{b_2(t-c_2)}} \quad H_2(t) = \frac{1}{1 + e^{-b_2(t-c_2)}}$$

$$L_3(t) = \frac{1}{1 + e^{b_3(t-c_3)}} \quad H_3(t) = \frac{1}{1 + e^{-b_3(t-c_3)}}$$

$$b_1 = 6$$

$$b_2 = 6$$

$$b_3 = 6$$

$$c_1 = 1.5$$

$$c_2 = 7.5$$

$$c_3 = 4.5$$

Fuzzy množiny v konsekvencích jsou dány takto:

B = "hodnota portfolia je vysoká"

S = "hodnota portfolia je malá"

VB a VS jsou modifikace "velmi vysoká" a "velmi malá".

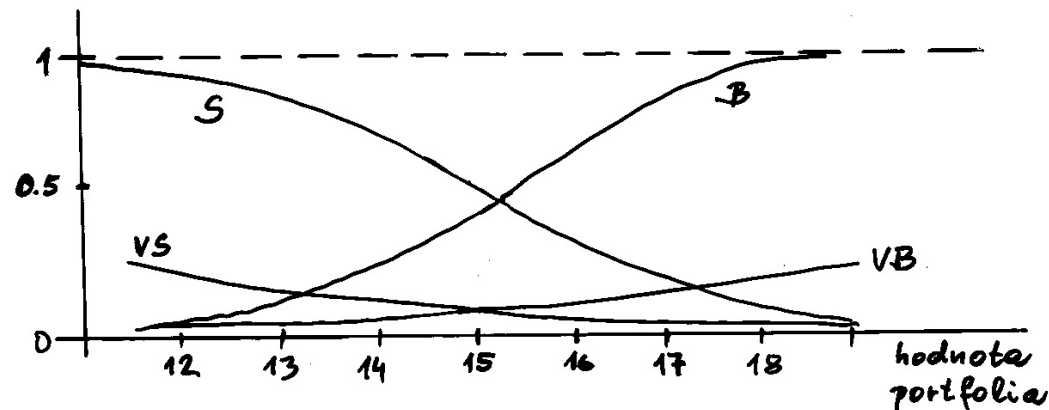
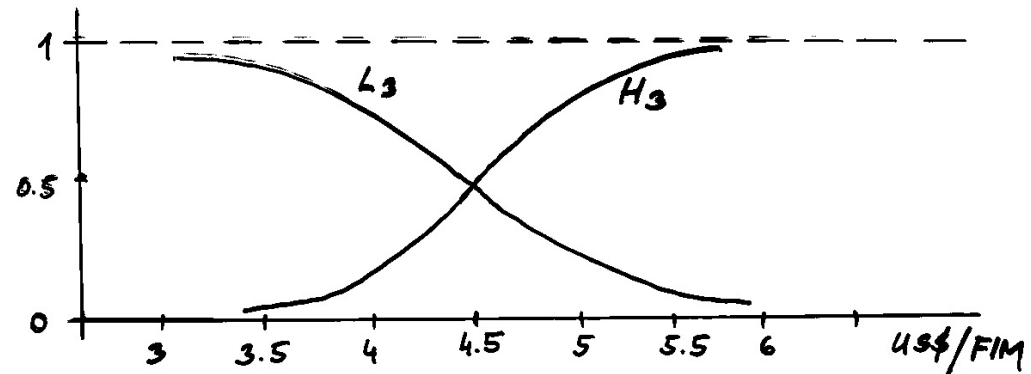
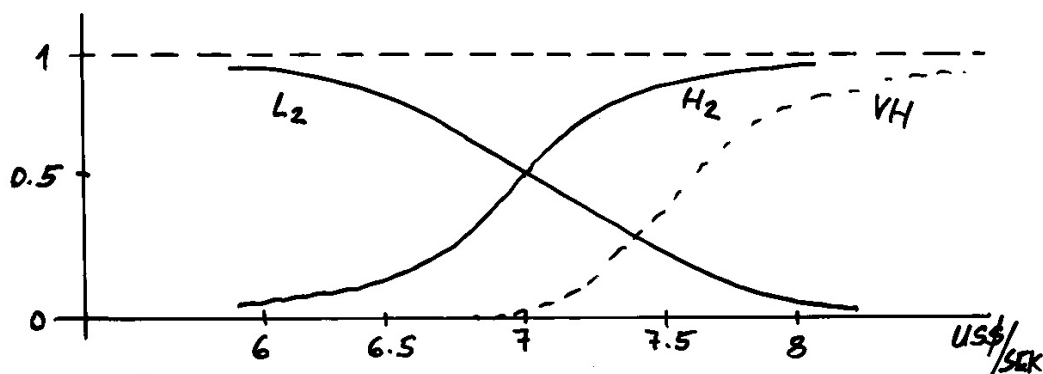
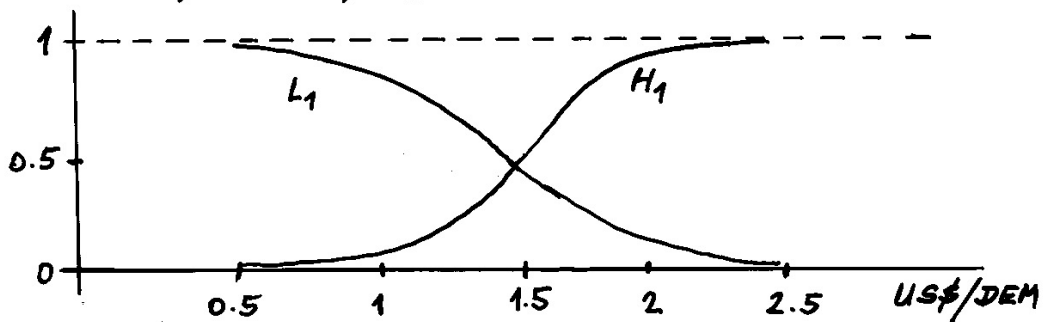
$$B(t) = \frac{1}{1 + e^{-b_4(t-c_4)}} \quad S(t) = \frac{1}{1 + e^{b_4(t-c_4)}}$$

\downarrow sklon \downarrow střed

$$VB(t) = \frac{1}{1 + e^{-b_4(t-c_4-c_5)}} \quad VS(t) = \frac{1}{1 + e^{b_4(t-c_4-c_5)}}$$

\rightarrow posun

$$b_4 = 0.5, \quad c_4 = 15, \quad c_5 = 5$$



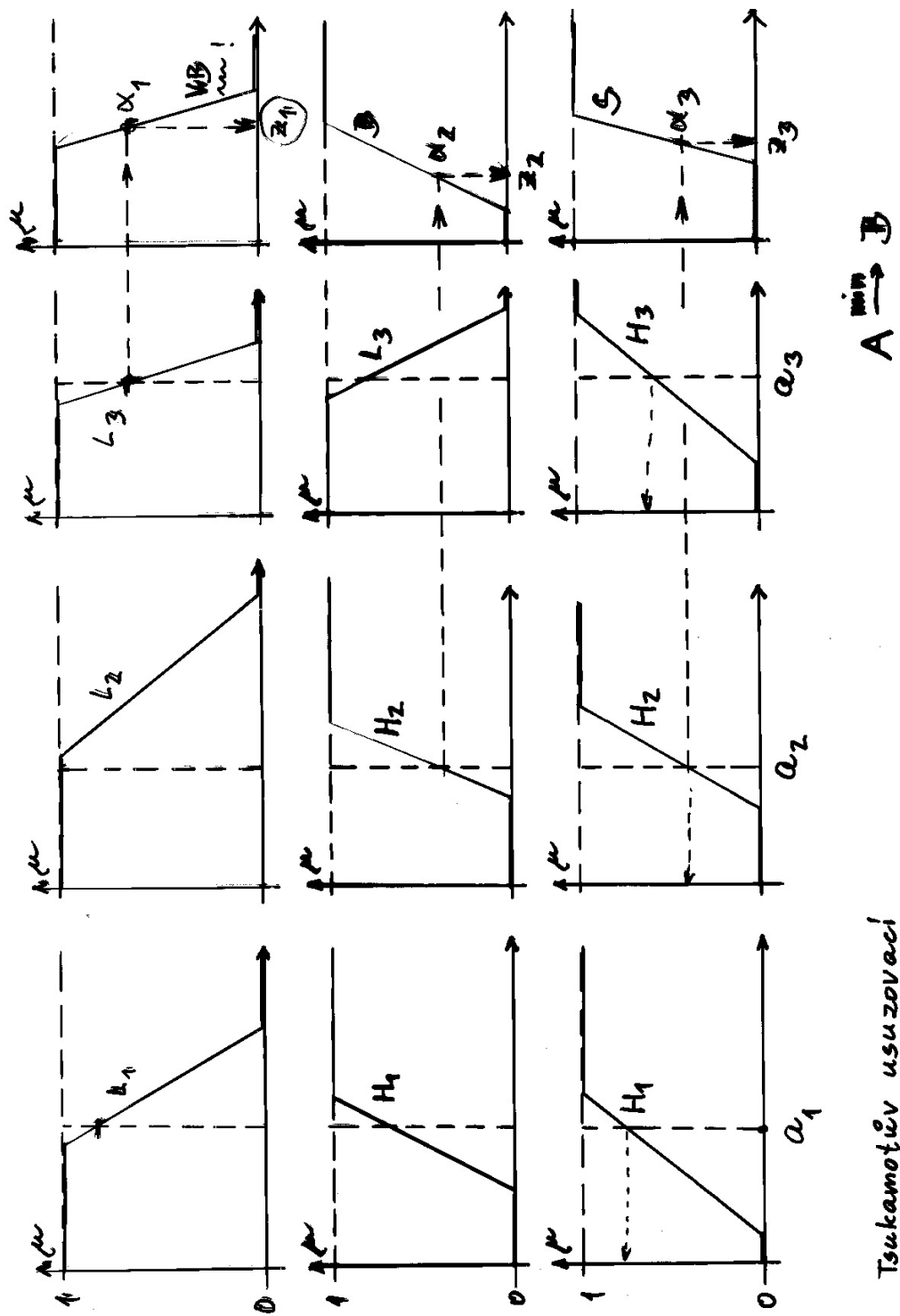
Vyhodnocení portfolia (inference) pomocí Tsukamotova usuzovacího mechanismu:

- výpočet aktivacních úrovní pravidel:

$$\alpha_1 = L_1(a_1) \wedge L_2(a_2) \wedge L_3(a_3)$$

$$\alpha_2 = H_1(a_1) \wedge H_2(a_2) \wedge L_3(a_3)$$

$$\alpha_3 = H_1(a_1) \wedge H_2(a_2) \wedge H_3(a_3)$$



Konsekventy jednotlivých pravidel:

$$z_1 = VB^{-1}(\alpha_1) = c_4 + c_5 + \frac{1}{b_4} \ln \frac{1 - \alpha_1}{\alpha_1}$$

$$z_2 = B^{-1}(\alpha_2) = c_4 + \frac{1}{b_4} \ln \frac{1 - \alpha_2}{\alpha_2}$$

$$z_3 = S^{-1}(\alpha_3) = c_4 - \frac{1}{b_4} \ln \frac{1 - \alpha_3}{\alpha_3}$$

Celkový výstup systému:

$$z_0 = \frac{\alpha_1 z_1 + \alpha_2 z_2 + \alpha_3 z_3}{\alpha_1 + \alpha_2 + \alpha_3}$$

Pozn.: Aktuální hodnoty vstupů jsou značeny jako a_1, a_2, a_3 (tj. právě platné poměry kursů).

Takagi-Sugeno usuzovací mechanismus má svůj ekvivalent ve struktuře ANFIS, tj. umělá neuronová síť s architekturou ANFIS realizuje soubor fuzzy pravidel IF - THEN. (hybridní systém)

(1) Výstup neuronů této vrstvy je stupeň, do něhož se shoduje vstup s lingvistickou hodnotou přiřazenou tomuto uzlu.

(2) Ve druhé vrstvě počítá každý uzel míru aktivace odpovídajícího pravidla:

$$\alpha_1 = L_1(a_1) \underline{t} L_2(a_2) \underline{t} L_3(a_3)$$

$$\alpha_2 = H_1(a_1) \underline{t} H_2(a_2) \underline{t} L_3(a_3)$$

$$\alpha_3 = H_1(a_1) \underline{t} H_2(a_2) \underline{t} H_3(a_3)$$

kde \underline{t} označuje t-normu realizující logickou spojku AND (např. minimum \wedge).

(3) Uzly třetí vrstvy realizují normalizaci aktivacních úrovní:

$$\beta_1 = \frac{\alpha_1}{\alpha_1 + \alpha_2 + \alpha_3}$$

$$\beta_2 = \frac{\alpha_2}{\alpha_1 + \alpha_2 + \alpha_3}$$

$$\beta_3 = \frac{\alpha_3}{\alpha_1 + \alpha_2 + \alpha_3}$$

(4) Výstupem neuronů čtvrté vrstvy je součet normalizované aktivacní úrovně a pravé strany jednotlivých pravidel:

$$\beta_1 z_1 = \beta_1 \cdot V \bar{B}^1(\alpha_1)$$

$$\beta_2 z_2 = \beta_2 \cdot \bar{B}^1(\alpha_2)$$

$$\beta_3 z_3 = \beta_3 \cdot \bar{S}^1(\alpha_3)$$

(5) Jediný uzel páté vrstvy počítá celkový výstup systému jako součet vstupních signálů:

$$z_0 = \beta_1 z_1 + \beta_2 z_2 + \beta_3 z_3$$

Předpokládejme, že je dána ostrá tréninková množina:

$$\{ (\underbrace{x_1}_{\text{směnné kursy}}, \underbrace{y_1}_{\text{hodnoty portfolia}}), \dots, (\underbrace{x_K}_{\text{směnné kursy}}, \underbrace{y_K}_{\text{hodnoty portfolia}}) \}$$

kde x_k je vektor hodnot skutečných směnných kursů DEM, SEK, FIM vůči US\$ a y_k je reálná hodnota portfolia v čase $t=k$. Míru chyby k-tého tréninkového páru definujeme obvyklým vztahem:

$$E_k = \frac{1}{2} (y_k - \sigma_k)^2$$

kde σ_k je vypočítaný výstup fuzzy systému pravidel R pro vstup x_k a y_k je skutečná hodnota ($k=1, 2, \dots, K$)

K naučení parametrů antecedentů a konsekventů pravidel lze použít gradientní metodu. K „vyladění“ fuzzy množin je nutno určit optimalizovanou hodnotu parametrů \underline{b} a \underline{c} , např. pro b_4, c_4, c_5 (sklon, střed, posun):

$$b_4(t+1) = b_4(t) - \overset{\text{učicí konst.}}{\eta} \frac{\partial E_k}{\partial b_4} = b_4(t) - \frac{\eta}{b_4^2} \delta_k \frac{\alpha_1 + \alpha_2 - \alpha_3}{\alpha_1 + \alpha_2 + \alpha_3}$$

$$c_4(t+1) = c_4(t) - \eta \frac{\partial E_k}{\partial c_4} = c_4(t) + \eta \delta_k \frac{\alpha_1 + \alpha_2 + \alpha_3}{\alpha_1 + \alpha_2 + \alpha_3}$$

$$= c_4(t) + \eta (\delta_k) \rightarrow \text{chyba na výstupu}$$

$$c_5(t+1) = c_5(t) - \eta \frac{\partial E_k}{\partial c_5} = c_5(t) + \eta \delta_k \frac{\alpha_1}{\alpha_1 + \alpha_2 + \alpha_3}$$

kde $\delta_k = y_k - \sigma_k$ označuje chybu, $\eta > 0$ je učicí konstanta, t označuje krok.

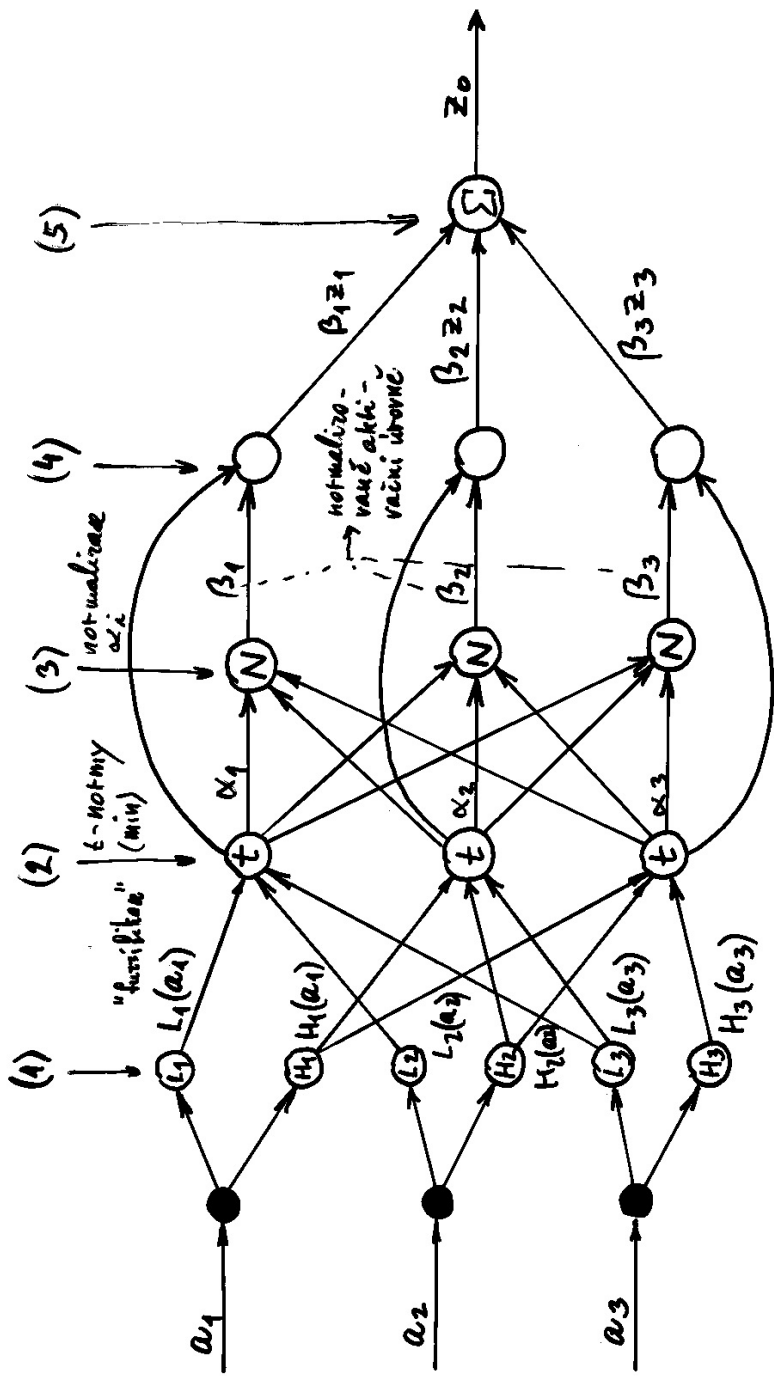
Ostatní parametry lze upravit na základě obdobného odvození.

$$z_i(t+1) = z_i(t) - \eta \frac{\partial E_k}{\partial z_i} =$$

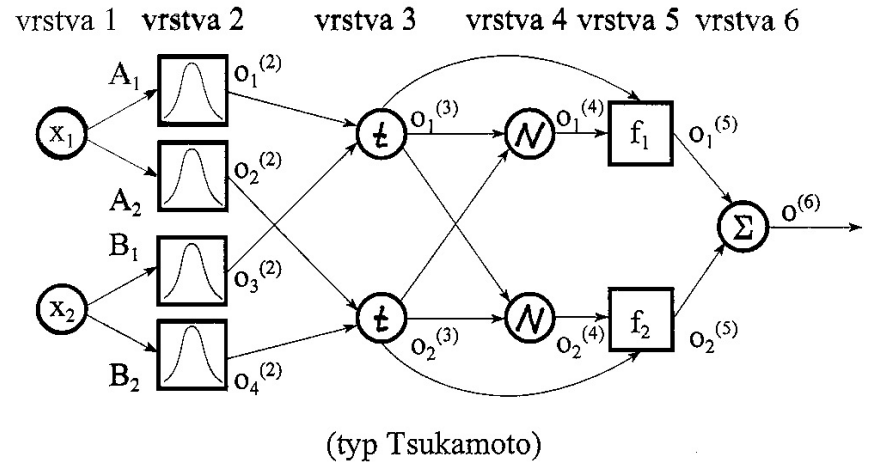
$$= z_i(t) - \eta (\sigma_k - y_k) \frac{\alpha_i}{\sum_{i=1}^m \alpha_i}$$

$$E_k = \frac{(\sigma_k - y_k)^2}{2}$$

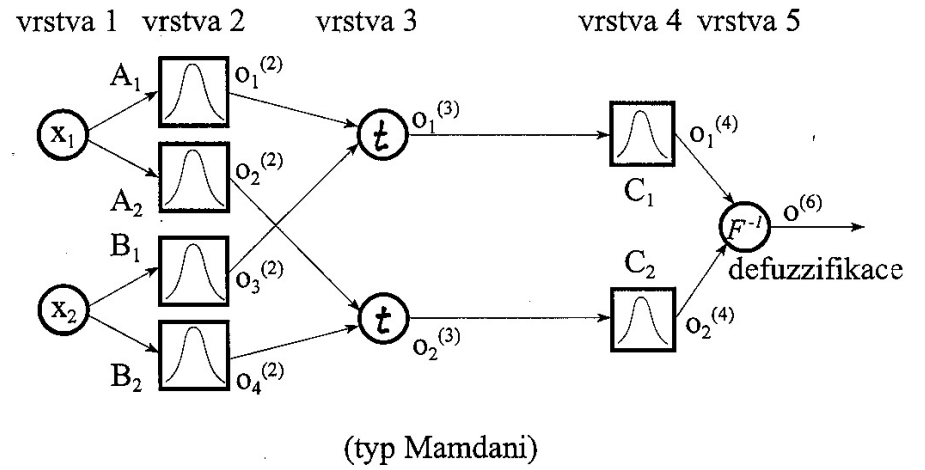
| datum | US\$/DEM | US\$/SEK | US\$/FIM | PV _{real} | PV _{comp} | PV _{comp} po tréninku |
|---------|----------|----------|----------|--------------------|--------------------|--------------------------------|
| 11.1.95 | 1.534 | 7.530 | 4.779 | 14.88 | 14.88 | 18.92 |
| 19.5.95 | 1.445 | 7.393 | 4.398 | 19.4 | 17.55 | 19.37 |
| 11.8.95 | 1.429 | 7.146 | 4.229 | 22.6 | 19.25 | 22.64 |
| 28.8.95 | 1.471 | 7.325 | 4.369 | 20 | 17.71 | 19.9 |



ANFIS-1

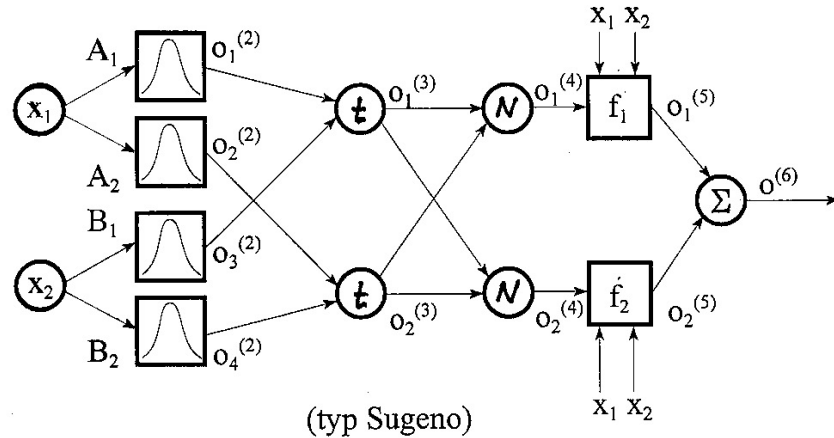


ANFIS-2



ANFIS-3

vrstva 1 vrstva 2 vrstva 3 vrstva 4 vrstva 5 vrstva 6



ANFIS-1

R_1 : IF $x_1 \in A_1$ AND $x_2 \in B_1$ THEN $o_1^{(5)} = f_1(o_1^{(3)}, o_1^{(4)})$

R_2 : IF $x_1 \in A_2$ AND $x_2 \in B_2$ THEN $o_2^{(5)} = f_2(o_2^{(3)}, o_2^{(4)})$

ANFIS-2

R_1 : IF $x_1 \in A_1$ AND $x_2 \in B_1$ THEN $o_1^{(4)} = C_1$

R_2 : IF $x_1 \in A_2$ AND $x_2 \in B_2$ THEN $o_2^{(4)} = C_2$

ANFIS-3

R_1 : IF $x_1 \in A_1$ AND $x_2 \in B_1$ THEN $o_1^{(5)} = f_1(x_1, x_2) = p_1x_1 + q_1x_2 + r_1$

R_2 : IF $x_1 \in A_2$ AND $x_2 \in B_2$ THEN $o_2^{(5)} = f_2(x_1, x_2) = p_2x_1 + q_2x_2 + r_2$

ANFIS architektura

(Adaptive Network Fuzzy Inference System)

Znalostní báze s fuzzy pravidly je přímo mapována do síťové struktury NN. Pomocí hybridního učícího algoritmu se systém adaptuje na své okolí využitím příkladů - optimalizují se pravidla. Strukturovaná síťová architektura rovněž umožňuje extrahovat optimalizovaná fuzzy pravidla z natrénované sítě.

ANFIS kombinuje fuzzy logiku s umělými neuronovými sítěmi. NN mohou být trénovány standardním BP nebo novým hybridním algoritmem.

Počáteční pravidla lze specifikovat jen zhruba. To umožňuje rychlý inkrementální návrh systémů, kdy se napřed existující znalost zakóduje do fuzzy pravidel, která jsou pak adaptována na konkrétní proces.

Existují 3 základní inferenční typy, určující bližší architekturu ANFIS:

ANFIS-1: Tsukamotova inference

ANFIS-2: Max-Min inference

ANFIS-3: Takagiho-Sugenova inference

Pravidla (fuzzy) typu IF-THEN mohou existovat v nějaké znalostní bázi. Tato pravidla jsou pak přímo mapována do struktury neuronové sítě (trénovatelné).

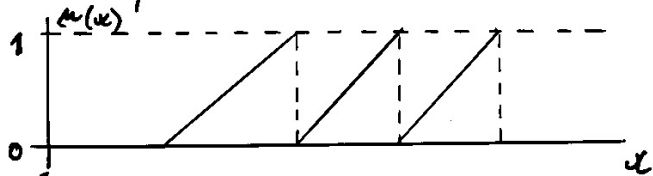
Díky speciální architektuře neuronové sítě je možno po tréninku (tj. optimalizaci či adaptaci) pravidla zpětně ze sítě extrahovat. Původní pravidla byla tedy použita k inicializaci NN jako výchozí znalost, která je učením prostřednictvím NN z kvalitněna.

Architektura ANFIS je speciální vícevrstvá dopředná NN skládající se z adaptivních a neadaptivních jednotek komunikujících pomocí přímých spojů. Adaptivní jednotky mají parametry, které jsou během učení optimalizovány.

Na obrázku je zřejmé, že adaptivní jednotky tvoří druhou vrstvu, přičemž obsahují řadu parametrů premisy pro definici funkce příslušnosti. Jednotky předcházející poslední vrstvu jsou rovněž adaptivní - parametry definují množinu parametrů konsekventů.

ANFIS-1

Konsekventní část (vrstva 5) používá pro zjednodušení lineární aproximaci pomocí monotónně uarrájených funkcí příslušnosti.



ANFIS-2

Realizuje fuzzy inferenci pomocí metody max-min. Oproti ANFIS-1 je jedna vrstva vynechána. Adaptivní jednotky ve čtvrté vrstvě provádějí inferenci. Výstupní jednotka slouží pro defuzzifikaci. Tento typ architektury je ve skutečnosti komplikovanější než typy 1 a 3. Učení je zde podstatně pomalejší, protože nastavování parametrů je složitější. Zde lze parametry optimalizovat pouze pomocí sestupné gradientní metody učení.

ANFIS-3

Realizuje tzv. Takagi-Sugenovu inferenci, která je ve srovnání s Tsukamotoovou lepší. Jednotky páté vrstvy obsahují parametry konsekventu $\{ \varphi_i, \sigma_i, \tau_i \}$.

Učení pro architekturu ANFIS

Učení je založeno na sestupné gradientní metodě, tj. v podstatě BP. Protože učení pomocí BP je pomalé a konvergence je omezena existencí lokálních minim, využívá se hybridní učicí algoritmus, který lze uplatnit na struktury obdobné ANFIS.

Existují dva algoritmy učení pro nastavení množiny parametrů $S = S_1 \cup S_2$ (S_1 jsou parametry premisy, S_2 parametry závěru).

Učicí cyklus se skládá z dopředného a zpětného chodu:

| | dopředný krok | zpětný krok |
|-------|-----------------------------|-------------------|
| S_1 | nemodifikován | gradientní sestup |
| S_2 | LSE (Least Square Error) | nemodifikován |

Dopředný krok

Předpokl., že ANFIS má jen 1 výstupní jednotku.
(Algoritmus lze snadno zobecnit pro více výstupů.)

Výstup out je generován následovně:

- (1) $out = F(\vec{I}, S)$ \vec{I} ... vektor všech vstupních prom.
 S ... množina param. jednotek
- (2) $H(out) = HoF(\vec{I}, S)$
- (3) $B = AX$
- (4) $(A^T A)^{-1} A^T B = X^*$

Existuje funkce H taková, že HoF je lineární v premise (parametry S_2). P příkladů změni (2) na maticovou rovnici (3), kde neznámý vektor X obsahuje elementy $\in S_2$. Počet elementů S_2 (tj. M elementů) je obvykle mnohem menší než počet trénovacích příkladů (P), takže (3) nemá exaktní řešení.

Pomocí algoritmu LSE se minimalizuje chyba $\|AX - B\|^2$ aproximací X na X^* (4). Přímý výpočet X^* zahrnuje časově náročné výpočty inverzní matice $(A^T A)^{-1}$ a vyžaduje, aby matice $(A^T A)$ nebyla singularní a byla dobře podmíněná. K výpočtu X^* se proto používá iterační metoda:

$$S_{i+1} = S_i - \frac{S_i a_{i+1}^T a_{i+1} S_i}{1 + a_{i+1}^T S_i a_{i+1}} \quad (5)$$

$$X_{i+1} = X_i + S_{i+1} a_{i+1} (b_{i+1}^T - a_{i+1}^T X_i) \quad (6)$$

$i = 0, 1, \dots, P-1$ (P je počet trénovacích příkladů),

$X_0 = \vec{0}$, $S_0 = \gamma I$ (γ je velké číslo), a_i^T je i -tý řádek matice A , b_i^T je i -tý prvek vektoru B , $X^* = X_p$.

Pozn.: je-li b_i^T i -tý řádek matice B , pak se jedná o zobecnění na více výstupů.

Zpětný krok

Chyba E_p je zpětně šířena tak, aby byly vhodně modifikovány parametry premisy S_1 . Pro každou dvojici vzorů p se počítají derivace $\frac{\partial E_p}{\partial \sigma}$ pro každý element (výstup) σ . Pro výstupní vrstvu platí: $\frac{\partial E_p}{\partial \sigma} = -2(t_p - \sigma)$

Pro vnitřní jednotku i vrstvy k se derivace chyby počítá pomocí následujícího řetězeného pravidla:

$$\frac{\partial E_p}{\partial \sigma_{ip}^{(k)}} = \sum_{m=1}^{n^{(k+1)}} \frac{\partial E_p}{\partial \sigma_{mp}^{(k+1)}} \frac{\partial \sigma_{mp}^{(k+1)}}{\partial \sigma_{ip}^{(k)}}, \quad 1 \leq k \leq L-1$$

↑
počet vrstev

Použití zjednodušených pravidel

ANFIS-1 neumožňuje modelovat konvexní funkce příslušnosti. ANFIS-2 vyžaduje defuzifikaci.

ANFIS-3 nedovoluje ~~připojení~~ asociaci elementů vrstvy 5 s lingvistickými hodnotami.

Možnost vyhnout se těmto nedostatkům dovoluje zjednodušená modifikace pravidel:

$$\text{IF } x_1 = A \text{ AND } x_2 = B \text{ THEN } y = d$$

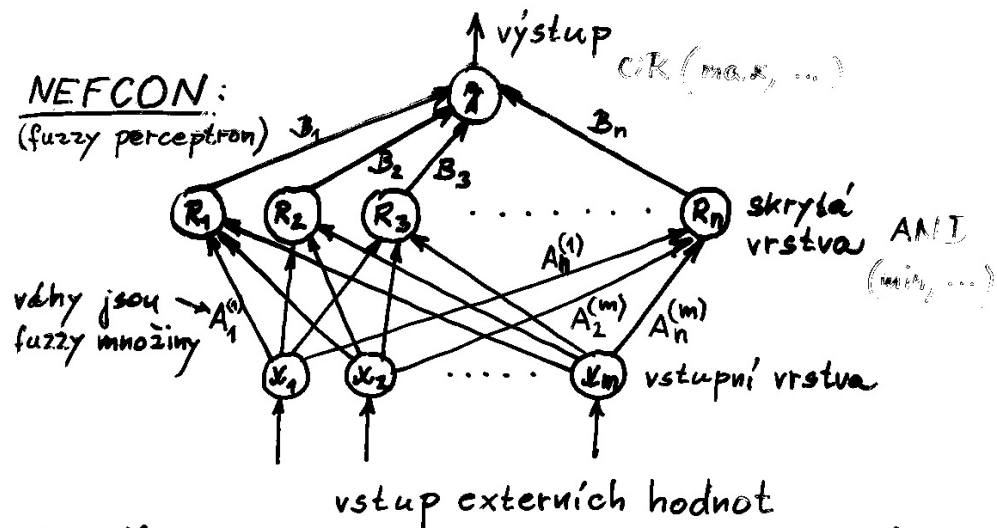
kde d je konstanta. Tato třída pravidel může být realizována kteroukoliv z architektur ANFIS, přičemž je zachována schopnost aproximace

nelineárních funkcí pomocí příkladů (datové body). (Důkaz plyne z aplikace Stone-Weierstrassova lemmatu.)

Pozn.: zjednodušená architektura ANFIS umožňuje typ inference, který je funkčně ekvivalentní RBF (Radial Basis Functions), pokud jsou funkce příslušnosti modelovány "gaussovskými" funkcemi.

NEFCON:

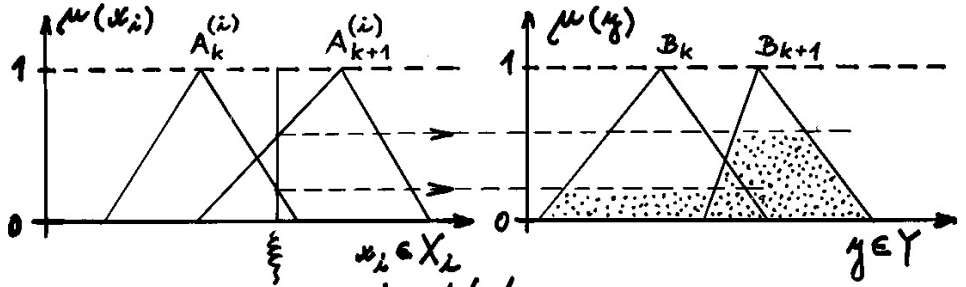
(fuzzy perceptron)



vstup externích hodnot

Pravidla: je $x_1 \in A_1^{(1)}$ AND $x_2 \in A_1^{(2)}$ AND ... THEN $y \in B_1$
 R_1 : IF $x_1 \in A_1^{(1)}$ AND $x_2 \in A_1^{(2)}$ AND ... THEN $y \in B_1$
 R_2 : IF $x_1 \in A_2^{(1)}$ AND $x_2 \in A_2^{(2)}$ AND ... THEN $y \in B_2$
 ...

Výstupní neuron poskytuje ostrou hodnotu (defuzifikovanou).



Přenosová funkce ~~skrytých~~ ^{skrytých} jednotek určuje $\mu(\xi)$, tj. stupeň shody ξ a $A_j^{(i)}$. Míru shody v antecedentu určuje aplikace příslušné t-normy. Jednotky skryté vrstvy šíří kombinaci míry shody antecedentu ~~do výstupní jednotky~~ do výstupní jednotky, která kombinuje míru shody v antecedentu s fuzzy množinou B_j . Výstupy R_j jsou akumulovány a tvoří celkový výstup pravidel.