O S T R A V S K Á U N I V E R Z I T A

P Ř Í R O D O V Ě D E C K Á F A K U L T A

N E U R O N O V É S Í T Ě 1

E V A V O L N Á

OSTRAVA 2002

Cíle předmětu

Seznámit studenta se základy teorie neuronových sítí a dát mu potřebnou motivaci pro pochopení důležitosti teorie pro praxi. Důraz bude kladen nejen na základní teorii, ale i na schopnost ji aplikovat při řešení konkrétních příkladů.
Všechny v textu uvedené popisy algoritmů jsou převzaty z [2].

Po prostudování textu budete znát:

TYTO UČEBNÍ TEXTY JSOU URČENY STUDENTŮM INFORMATIKY PRO PŘEDMĚT *NEURONOVÉ SÍTĚ I*. JSOU V NICH VYSVĚTLENY VŠECHNY ZÁKLADNÍ POJMY Z TEORIE UMĚLÝCH NEURONOVÝCH SÍTÍ. V JEDNOTLIVÝCH KAPITOLÁCH JSOU POSTUPNĚ PODLE OBTÍŽNOSTI UVEDENY ZÁKLADNÍ MODELY NEURONOVÝCH SÍTÍ (TJ. PERCEPTRON, ADALINE, MADALINE, DOPŘEDNÁ VÍCEVRSTVÁ NEURONOVÁ SÍŤ S ADAPTAČNÍ METODOU BACKPROPAGATION, ASOCIATIVNÍ NEURONOVÉ SÍTĚ A NEURONOVÉ SÍTĚ PRACUJÍCÍ NA PRINCIPU SAMOORGANIZACE), A TO JEJICH ARCHITEKTURA, AKTIVNÍ DYNAMIKA A ADAPTIVNÍ DYNAMIKA.

Obsah předmětu

1. Úvod do problematiky neuronových sítí.
2. Hebbovo učení.
3. Neuronová síť.
4. Perceptron.
5. Adaline. Madaline.
6. Backpropagation.
7. Varianty backpropagation.
8. Samoorganizace.
9. Counterpropagation.
10. Asociativní neuronové sítě.
11. Hopfieldova síť.
12. Dvousměrná asociativní paměť.
13. Postavení neuronových sítí v informatice.

Čas potřebný k prostudování učiva předmětu:
jeden semestr

ÚVOD DO PROBLEMATIKY NEURONOVÝCH SÍTÍ.

V této úvodní kapitole se stručně seznámíte s historií neuronových sítí a se základním matematickým modelem biologického neuronu, tj. *formálním neuronem*. Z tohoto modelu budeme dále vycházet, a proto je nutné, abyste jeho pochopení věnovali zvýšenou pozornost.

Klíčová slova této kapitoly:

stav neuronu, bias neuronu, vnitřní potenciál neuronu, synaptické váhy, aktivační(přenosová) funkce.

Historie neuronových sítí [6]

Za počátek vzniku oboru neuronových sítí je považována práce Warrena McCullocha a Waltera Pittse z roku 1943, kteří vytvořili velmi jednoduchý matematický model neuronu, což je základní buňka nervové soustavy. Číselné hodnoty parametru v tomto modelu byly převážně bipolární, tj. z množiny {-1,0,1}. Ukázali, že nejjednodušší typy neuronových sítí mohou v principu počítat libovolnou aritmetickou nebo logickou funkci. Ačkoliv nepočítali s možností bezprostředního praktického využití svého modelu, jejich článek měl velký vliv na ostatní badatele. Například zakladatel kybernetiky Norbert Wiener se jím inspiroval při studiu podobnosti činnosti nervové soustavy a systémů výpočetní techniky. Nebo autor amerického projektu elektronických počítačů John von Neumann napsal práce, ve kterých navrhoval výzkum počítačů, které by byly inspirovány činností mozku. Tyto návrhy, přestože byly hojně citovány, nepřinesly zpočátku očekávané výsledky.

V roce 1949 napsal Donald Hebb knihu „The Organization of Behaviour“, ve které navrhl učící pravidlo pro synapse neuronů (mezineuronové rozhraní). Toto pravidlo bylo inspirováno myšlenkou, že podmíněné reflexy, které jsou pozorovatelné u všech živočichů, jsou vlastnostmi jednotlivých neuronů. Hebb se snažil vysvětlit některé experimentální výsledky psychologie. Také jeho práce ovlivnila ostatní vědce, kteří se začali zabývat podobnými otázkami. Avšak 40. a 50. léta zatím ještě nepřinesla zásadní pokroky v oblasti neurovýpočtů. Typickým příkladem výzkumu v tomto období byla v roce 1951 konstrukce prvního neuropočítače Snark, u jehož zrodu stál Marvin Minsky. Snark byl sice úspěšný z technického hlediska, dokonce již automaticky adaptoval váhy (tj. míra synaptické propustnosti), ale ve skutečnosti nebyl nikdy využit k řešení nějakého zajímavého praktického problému. Nicméně jeho architektura později inspirovala další konstruktéry neuropočítačů.

V roce 1957 Frank Rosenblatt vynalezl tzv. perceptron, který je zobecněním McCullochova a Pittsova modelu neuronu pro reálný číselný obor parametrů. Pro tento model navrhl učící algoritmus, o kterém matematicky dokázal, že pro daná tréninková data nalezne po konečném počtu kroků odpovídající váhový vektor parametrů (pokud existuje) nezávisle na jeho počátečním nastavení. Rosenblatt také napsal jednu z prvních knih o neurovýpočtech „Principles of Neurodynamics“.

Na základě tohoto výzkumu Rosenblatt spolu s Charlesem Wightmanem a dalšími sestrojili během let 1957 a 1958 první úspěšný neuropočítač, který nesl jméno „Mark I Perceptron“. Protože původním odborným zájmem Rosenblatta bylo rozpoznávání obrazců, „Mark I Perceptron“ byl navržen pro rozpoznávání znaků. Znak byl promítán na světelnou tabuli, ze které byl snímán polem 20x20 fotovodičů. Intenzita 400 obrazových bodů byla vstupem do neuronové sítě perceptronů, jejímž úkolem bylo klasifikovat, o jaký znak se jedná (např. „A“, „B“ apod.). „Mark I Perceptron“ měl 512 adaptovatelných váhových parametrů, které byly realizovány polem 8x8x8 potenciometrů. Hodnota odporu u každého potenciometru, která právě odpovídala příslušné váze, byla nastavována automaticky samostatným motorem. Ten byl řízen analogovým obvodem, který implementoval perceptronový učící algoritmus. Jednotlivé perceptrony bylo možné spojit se vstupy libovolným způsobem. Typicky bylo použito náhodné zapojení, aby se ilustrovala schopnost perceptronu učit se požadované vzory bez přesného zapojení drátů v protikladu ke klasickým programovatelným počítačům. Díky úspěšné presentaci

uvedeného neuropočítače se neurovýpočty, které byly alternativou ke klasickým výpočtům realizovaným na von neumannovské architektuře počítače, staly novým předmětem výzkumu. Frank Rossenblatt je proto dodnes některými odborníky považován za zakladatele tohoto nového oboru.

Krátce po objevu perceptronu Bernard Widrow se svými studenty vyvinul další typ neuronového výpočetního prvku, který nazval „ADALINE“ (ADAptive LInear NEuron). Tento model byl vybaven novým výkonným učícím pravidlem, které se dodnes nezměnilo. Widrow se svými studenty demonstroval funkčnost „ADALINE“ na mnoha jednoduchých typových příkladech. Widrow také založil první firmu (Memistor Corporation) orientovanou na hardware neuropočítačů, která v první polovině 60. let vyráběla a prodávala neuropočítače a jejich komponenty.

Na přelomu 50. a 60. let dochází k úspěšnému rozvoji neurovýpočtů v oblasti návrhu nových modelů neuronových sítí a jejich implementací. Například Karel Steinbuch vyvinul model binární asociativní sítě nebo Roger Barron a Lewey Gilstrap založil v roce 1960 první firmu zaměřenou na aplikace neurovýpočtů. Výsledky z uvedeného období jsou shrnuty v knize Nilse Nilssona „Learning Machines“ z roku 1965.

Přes nesporné úspěchy dosažené v tomto období se obor neuronových sítí potýkal se dvěma problémy. Za prvé, většina badatelů přistupovala k neuronovým sítím z experimentálního hlediska a zanedbávala analytický výzkum neuronových modelů. Za druhé, nadšení některých výzkumných pracovníků vedlo k velké publicitě neopodstatněných prohlášení (např. za několik málo let bude vyvinut umělý mozek). Tyto skutečnosti diskreditovaly neuronové sítě v očích odborníků z jiných oblastí a odradily vědce a inženýry, kteří se o neurovýpočty zajímali. Navíc se samostatný obor neuronových sítí vyčerpal a další krok v této oblasti by býval požadoval radikálně nové myšlenky a postupy. Nejlepší odborníci oblast neuronových sítí opouštěli a začali se zabývat příbuznými obory umělé inteligence.

Poslední epizodou tohoto období byla kampaň vedená Marvinem Minským a Seymourem Papertem, kteří využili svůj vliv na to, aby zdiskreditovali výzkum neuronových sítí, nacházející se v krizi, ve snaze přenést finanční zdroje z této oblasti na jiný výzkum v oblasti umělé inteligence. V té době koloval rukopis jejich výzkumné zprávy, která napomáhala tomuto záměru. Uvedený rukopis byl v upravené formě publikován roce 1969 pod názvem „Perceptrons“. V této knize Minsky a Papert využili pro svou argumentaci známého triviálního faktu, že jeden perceptron nemůže počítat jednoduchou logickou funkci, tzv. vylučovací disjunkci (XOR). Tento problém lze sice vyřešit vytvořením dvouvrstvé sítě se třemi neurony, ale pro vícevrstvý perceptron nebyl v této době znám učící algoritmus. Autoři z toho nesprávně vyvodili, že takový algoritmus vzhledem ke komplikovanosti funkce, kterou vícevrstvá síť počítá, snad ani není možný. Jejich tvrzení bylo všeobecně přijato a považováno za matematicky dokázané. Kampaň Minského a Paperta byla úspěšná, výzkum neuronových sítí nebyl již déle dotován a neurovýpočty byly považovány za neperspektivní.

V dalším období od roku 1967 do 1982 probíhal výzkum neuronových sítí ojediněle a izolovaně, převážně mimo území Spojených států, kde kniha „Perceptrons“ měla velký vliv. Většina prací byla publikována např. pod hlavičkou adaptivní zpracování signálů, rozpoznávání obrazců a biologické modelování. Avšak již v počátcích tohoto *tichého* období se neurovýpočty začali zabývat talentovaní badatelé, mezi nimi byli např. Shun-Ichi Amari, James Anderson, Kunihiko Fukushima, Stephen Grossberg, Harry Klopf, Teuvo Kohonen a David Willshaw. Tito vědci přispěli svými objevy k renesanci neuronových sítí.

Počátkem 80. let se badatelé v oblasti neurovýpočtů osmělili a začali podávat vlastní grantové projekty zaměřené na vývoj neuropočítačů a jejich aplikace. Zásluhou programového manažera Ira Skurnicka začala v roce 1983 americká grantová agentura DARPA (Defense Advanced Research Projects Agency) finančně podporovat výzkum neuronových sítí a jejího příkladu v krátké době následovaly i jiné organizace podporující základní i aplikovaný výzkum.

Další zásluhu na renesanci oboru neuronových sítí měl světově uznávaný fyzik John Hopfield, který se v této době zabýval neurovýpočty. Své výsledky publikoval v roce 1982 a 1984. Ukázal souvislost některých modelů neuronových sítí s fyzikálními modely magnetických materiálů. Svými zvanými přednáškami, které měl po celém světě, získal pro neuronové sítě stovky kvalifikovaných vědců, matematiků a technologů.

V roce 1986 publikovali své výsledky badatelé z tzv. „PDP skupiny“ (Parallel Distributed Processing Group). Ve svých pracích popsali učící algoritmus zpětného šíření chyby (backpropagation) pro vícevrstvou neuronovou síť a vyřešili tak problém, který se Minskému a Pappertovi v 60. letech jevil jako nepřekonatelná překážka pro využití a další rozvoj neuronových sítí. Tento algoritmus je doposud nejpoužívanější učící metodou neuronových sítí a jeho publikováním dosáhl zájem o neuronové sítě svého vrcholu.

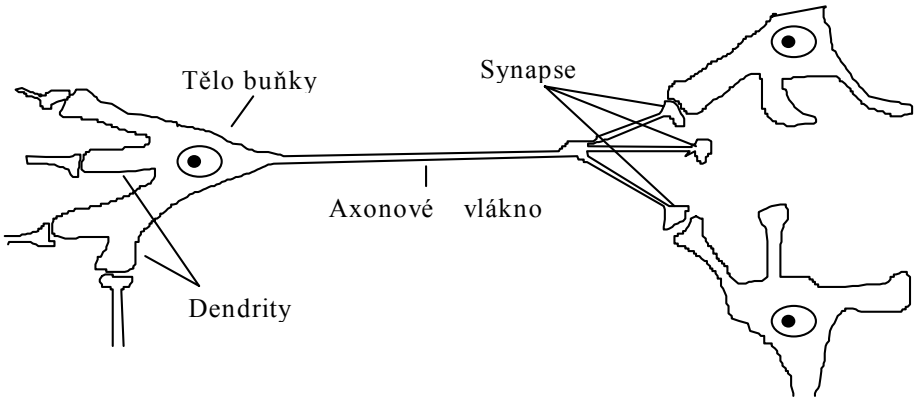
V roce 1987 se v San Diegu konala první větší konference specializovaná na neuronové sítě (IEEE International Conference on Neural Networks), na které byla založena mezinárodní společnost pro výzkum neuronových sítí INNS (International Neural Network Society). O rok později INNS začala vydávat svůj časopis „Neural Networks“. V následujících letech vznikly další specializované časopisy: Neural Computing (1989), IEEE Transactions on Neural Networks (1990) a mnoho jiných (např. v Praze vychází od roku 1991 mezinárodní časopis Neural Network World). Od roku 1987 mnoho renovovaných univerzit založilo nové výzkumné ústavy

zabývající se neuronovými sítěmi a vyhlásilo výukové programy zaměřené na neurovýpočty. Tento trend pokračuje dodnes.

Biologický neuron [6]

Původním cílem výzkumu neuronových sítí byla snaha pochopit a modelovat způsob, jakým myslíme a způsob, jak funguje lidský mozek. Neurofyziologické poznatky umožnily vytvořit zjednodušené matematické modely, které se dají využít pro neurovýpočty při řešení praktických úloh z oblasti umělé inteligence. To znamená, že neurofyziologie zde slouží jen jako zdroj inspirací a navržené modely neuronových sítí jsou již dále rozvíjeny bez ohledu na to, zda modelují lidský mozek. Při vytváření modelů neuronových sítí nám nejde o vytvoření identických kopií lidského mozku, ale chceme napodobit pouze jeho základní funkce.

Základním stavebním funkčním prvkem nervové soustavy je nervová buňka, *neuron*. Neurony jsou samostatné specializované buňky, určené k přenosu, zpracování a uchování informací, které jsou nutné pro realizaci životních funkcí organismu. Struktura neuronu je schématicky znázorněna na obrázku 1.



Obrázek 1: ***Biologický neuron.***

Neuron je přizpůsoben pro přenos signálů tak, že kromě vlastního těla (*somatu*), má i vstupní a výstupní přenosové kanály: *dendrity* a *axon*. Z axonu odbočuje řada větví (*terminálů*), zakončených blánou, která se převážně stýká s výběžky (*trny*), dendritů jiných neuronů. K přenosu informace pak slouží unikátní mezineuronové rozhraní, *synapse*. Míra synaptické propustnosti je nositelem všech význačných informací během celého života organismu. Z funkčního hlediska lze synapse rozdělit na *excitační*, které umožňují rozšíření vzruchu v nervové soustavě a na *inhibiční*, které způsobují jeho útlum. Paměťová stopa v nervové soustavě vzniká pravděpodobně zakódováním synaptických vazeb na cestě mezi receptorem (čidlem orgánu) a efektorem (výkonným orgánem). Šíření informace je umožněno tím, že soma i axon jsou obaleny membránou, která má schopnost za jistých okolností generovat elektrické impulsy. Tyto impulsy jsou z axonu přenášeny na dendrity jiných neuronů synaptickými branami, které svojí propustností určují intenzitu podráždění dalších neuronů. Takto podrážděné neurony při dosažení určité hraniční meze, tzv. *prahu*, samy generují impuls a zajišťují tak šíření příslušné informace. Po každém průchodu signálu se synaptická propustnost mění, což je předpokladem paměťové schopnosti neuronů. Také propojení neuronů prodělává během života organismu svůj vývoj: v průběhu učení se vytváří nové paměťové stopy nebo při zapomínání se synaptické spoje přerušují.

Nervová soustava člověka je velmi složitý systém, který je stále předmětem zkoumání. Uvedené velmi zjednodušené neurofyziologické principy nám však v dostatečné míře stačí k formulaci matematického modelu neuronové sítě.

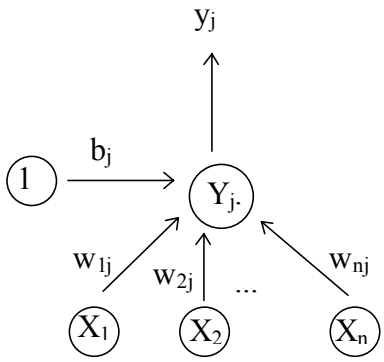
V dalších kapitolách budeme používat následujícího značení:

x_i, y_j	Stav neuronů X_i, Y_j, tj. pro vstupní neurony X_i je x_i vstupní signál; pro ostatní neurony Y_j je $y_j = f(y\_in_j)$.
w_{ij}	Váha přiřazena spojení z neuronu X_i do neuronu Y_j.

b_j	Bias neuronu Y_j.
y_in_j	Vnitřní potenciál neuronu Y_j:
W	Váhová matice: $W = \{w_{ij}\}$.
$\boldsymbol{w}_{.j}$	Vektor vah: $\boldsymbol{w}_{.j} = (w_{1j}, w_{2j}, ..., w_{nj})^T$.. Je to j.sloupec váhové matice.
$\|x\|$	Norma nebo velikost vektoru **x**.
θ_j	Práh pro aktivační funkci neuronu Y_j.
s	Tréninkový vstupní vektor: $\mathbf{s} = (s_1, s_2, ..., s_n)$.
t	Tréninkový výstupní vektor: $\mathbf{t} = (t_1, t_2, ..., t_m)$.
x	Vstupní vektor: $\mathbf{x} = (x_1, x_2, ..., x_n)$.
Δw_{ij}	Změna váhy w_{ij}: $\Delta w_{ij} = [w_{ij}(new) - w_{ij}(old)]$.
α	Koeficient učení.

Formální neuron [6]

Základem matematického modelu neuronové sítě je *formální neuron*. Jeho struktura je schematicky zobrazena na obrázku 2. Formální neuron Y_j (dále jen neuron) má n obecně reálných *vstupů* $x_1,...,x_n$, které modelují dendrity. Vstupy jsou ohodnoceny reálnými *synaptickými váhami* $w_{1j}, \ , w_{nj}$., které určují jejich propustnost.Ve shodě s neurofyziologickou motivací mohou být synaptické váhy i záporné, čímž se vyjadřuje jejich inhibiční charakter.



Obrázek 2: *Formální neuron s biasem.*

Vážená suma vstupních hodnot představuje *vnitřní potenciál j.* neuronu:

$$y\_in_j = \sum_{i=1}^{n} w_{ij} x_i \ .$$

Bias může být do vztahu včleněn přidáním komponent $x_0 = 1$ k vektoru $\mathbf{x}$, tj. $\mathbf{x} = (1, x_1, x_2, ..., x_n)$. Bias je dále zpracováván jako jakákoliv jiná váha, tj. $w_{0j} = b_j$. Vstup do neuronu Y_j je dán následujícím vztahem

$$\begin{aligned} y\_in_j &= \sum_{i=0}^{n} w_{ij} x_i \\ &= w_{0j} + \sum_{i=1}^{n} x_i w_{ij} \\ &= b_j + \sum_{i=1}^{n} x_i w_{ij} \ . \end{aligned}$$

Hodnota vnitřního potenciálu y_in_j po dosažení *prahové* hodnoty b_j indukuje *výstup (stav)* neuronu y_j, který modeluje elektrický impuls axonu. Nelineární nárůst výstupní hodnoty $y_j = f(y\_in_j)$ při dosažení prahové hodnoty potenciálu b_j je dán *aktivační (přenosovou) funkcí f*. Nejjednodušším typem přenosové funkce je *ostrá nelinearita*, která má pro *j.* neuron (Y_j) tvar:

$$f\left(y\_in_j\right) = \begin{cases} 1 & \text{pokud } y\_in_j \geq 0; \\ 0 & \text{pokud } y\_in_j < 0. \end{cases}$$

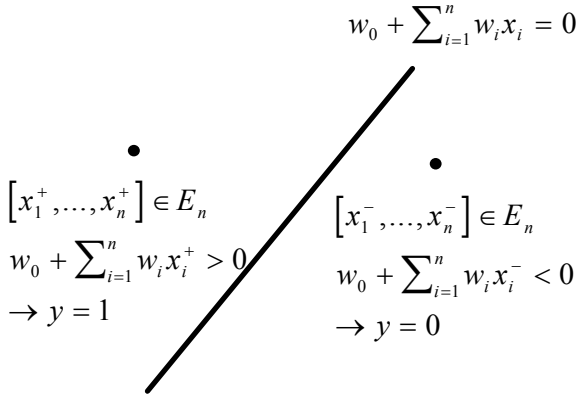
Někteří autoři nepoužívají váhový bias, ale místo toho pracují s fixním prahem θ pro aktivační funkci. V tomto případě má přenosové funkce *ostrá nelinearita* pro *j.* neuron (Y_j) tvar:

$$f\left(y\_in_j\right) = \begin{cases} 1 & \text{pokud } y\_in_j \geq \theta; \\ 0 & \text{pokud } y\_in_j < \theta, \end{cases}$$

kde

$$y\_in_j = \sum_{i=1}^{n} w_{ij} x_i \ .$$

K lepšímu pochopení funkce jednoho neuronu nám pomůže geometrická představa načrtnutá na obrázku 3. Vstupy neuronu budeme chápat jako souřadnice bodu v *n-rozměrném Euklidovském vstupním prostoru* E_n.



Obrázek 3: ***Geometrická interpretace funkce neuronu.***

V tomto prostoru má rovnice nadroviny (v E_2 přímka, v E_3 rovina) tvar:

$$w_0 + \sum_{i=1}^{n} w_i x_i = 0.$$

Tato nadrovina dělí vstupní prostor na dva poloprostory. Souřadnice bodů $[x_1^+,...,x_n^+]$, které leží v jednom poloprostoru, splňují následující nerovnost:

$$w_0 + \sum_{i=1}^{n} w_i x_i^+ > 0.$$

Body $[x_1^-,...,x_n^-]$ z druhého poloprostoru pak vyhovují relaci s opačným relačním znaménkem:

$$w_0 + \sum_{i=1}^{n} w_i x_i^- < 0.$$

Synaptické váhy neuronu $w_0,..., w_n$ (včetně biasu) lze chápat jako koeficienty této nadroviny. Je zřejmé, že neuron klasifikuje, ve kterém z obou poloprostorů určených nadrovinou leží bod, jehož souřadnice jsou na vstupu, tj. neuron realizuje *dichotomii* vstupního prostoru. Neuron je tedy *aktivní*, je-li jeho stav $y = 1$ a *pasivní*, pokud je jeho stav $y = 0$.

Úkoly:

Vytvořte geometrickou interpretaci funkce jednoho neuronu ve 2-rozměrném Euklidovském prostoru. Vstupy neuronu jsou souřadnice bodu v E_2.

HEBBOVO UČENÍ.

Dříve než se pustíte do studia této kapitoly, důkladně se seznamte s problematikou ***formálního neuronu*** a s používaným ***značením*** (viz kapitola „*Úvod do problematiky neuronových sítí*“).

Klíčová slova této kapitoly:

Hebbovo učení, tréninkový vzor, váhový přírustek.

Hebbovo učení

Hebbovo učení je založeno na myšlence, že váhové hodnoty na spojení mezi dvěma neurony, které jsou současně ve stavu „on“, budou narůstat a naopak: váhové hodnoty na spojení mezi dvěma neurony, které jsou současně ve stavu „off“, se budou zmenšovat. Uvažujme jednovrstvou (dopřednou) neuronovou síť, ve které jsou všechny vstupní neurony propojeny s jediným výstupní neuronem, ale ne již navzájem mezi sebou. Pokud jsou data reprezentována v bipolární formě, lze váhové hodnoty aktualizovat následovně:

$$w_i(new) = w_i(old) + x_i y.$$

Popis algoritmu

Krok 0. Inicializace všech vah:

$w_i = 0 \quad (i = 1$ až $n)$

Krok 1. Pro každý vzor - tréninkový pár, tj. vstupní vektor (**s**) a příslušný výstup (**t**), opakovat následující kroky (2 až 4).

Krok 2. Aktivovat vstupní neurony:

$x_i = s_i \quad (i = 1$ až $n)$.

Krok 3. Aktivovat výstupní neuron:

$y = t$.

Krok 4. Aktualizovat váhy podle

$w_i(new) = w_i(old) + x_i y \quad (i = 1$ až $n)$.

Aktualizovat biasy podle

$b(new) = b(old) + y$.

Bias lze zapsat také jako váhovou hodnotu přiřazenou výstupu z neuronu, jehož aktivace má vždy hodnotu 1. Aktualizace váhových hodnot může být také vyjádřena ve vektorové formě jako

$$\mathbf{w}(new) = \mathbf{w}(old) + \mathbf{x}y.$$

Váhový přírůstek lze zapsat ve tvaru

$$\Delta \boldsymbol{w} = \boldsymbol{x}y$$

a potom

$$\boldsymbol{w}(new) = \boldsymbol{w}(old) + \Delta \boldsymbol{w}.$$

Výše uvedený algoritmus je pouze jedním z mnoha způsobu implementace Hebbova pravidla učení. Tento algoritmus vyžaduje pouze jeden průchod tréninkovou množinou. Existují však i jiné ekvivalentní metody nalezení vhodných váhových hodnot, které jsou popsány dále.

Příklad:

Hebbovo pravidlo učení pro logickou funkci „AND“ v bipolární reprezentaci.

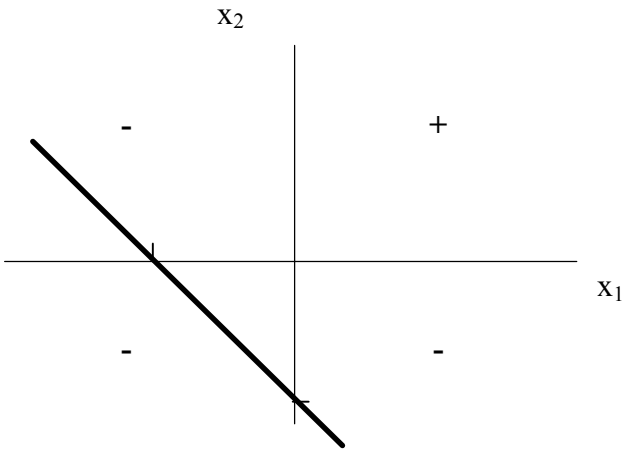
VSTUP			POŽADOVANÝ VÝSTUP
x_1	x_2	b	
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1

Po předložení *prvního* tréninkového vzoru, dostáváme následující:

VSTUP			POŽADOVANÝ VÝSTUP	PŘÍRUSTKY VAH			VÁHOVÉ HODNOTY		
x_1	x_2	b		Δw_1	Δw_2	Δb	w_1	w_2	b
							0	0	0
1	1	1	1	1	1	1	1	1	1

Separující nadrovina je dána rovnicí přímky

$$x_2 = -x_1 - 1.$$



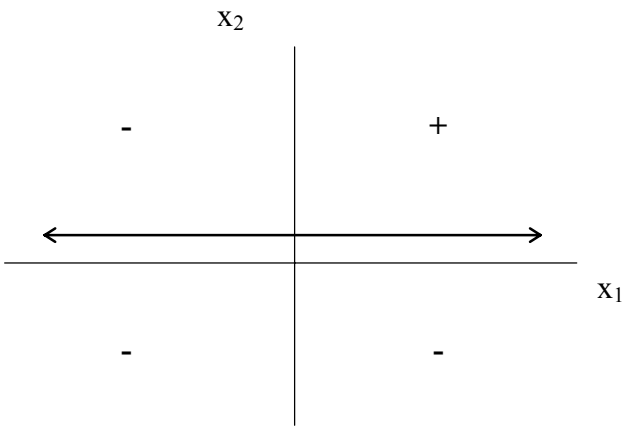
Obrázek 4: *Hebbovo pravidlo učení pro logickou funkci „AND“ v bipolární reprezentaci - první tréninkový vzor.*

Předložíme-li *druhý* tréninkový vzor, dostáváme následující:

VSTUP			POŽADOVANÝ VÝSTUP	PŘÍRUSTKY VAH			VÁHOVÉ HODNOTY		
x_1	x_2	b		Δw_1	Δw_2	Δb	w_1	w_2	b
							1	1	1
1	-1	1	-1	-1	1	-1	0	2	0

Separující nadrovina je pak dána rovnicí přímky

$$x_2 = 0.$$



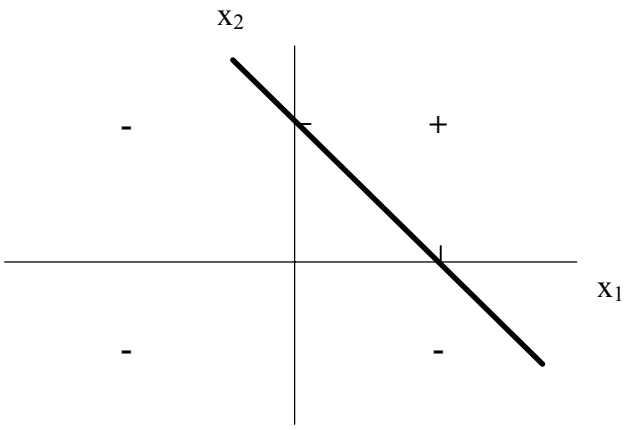
Obrázek 5: *Hebbovo pravidlo učení pro logickou funkci „AND“ v bipolární reprezentaci - druhý tréninkový vzor.*

Po předložení *třetího* tréninkového vzoru, dostáváme:

VSTUP			POŽADOVANÝ VÝSTUP	PŘÍRUSTKY VAH			VÁHOVÉ HODNOTY		
x_1	x_2	b		Δw_1	Δw_2	Δb	w_1	w_2	b
							0	2	0
-1	1	1	-1	1	-1	-1	1	1	-1

Separující nadrovina je dána rovnicí přímky

$$x_2 = -x_1 + 1.$$



Obrázek 6: *Hebbovo pravidlo učení pro logickou funkci „AND“ v bipolární reprezentaci - třetí a čtvrtý tréninkový vzor.*

A nakonec po předložení *čtvrtého* tréninkového vzoru, dostáváme:

VSTUP			POŽADOVANÝ VÝSTUP	PŘÍRUSTKY VAH			VÁHOVÉ HODNOTY		
x_1	x_2	b		Δw_1	Δw_2	Δb	w_1	w_2	b
							1	1	-1
-1	-1	1	-1	1	1	-1	2	2	-2

Tvar separující nadroviny (přímky) se nezměnil, tj.

$$x_2 = -x_1 + 1.$$

Úkoly:

Objasněte Hebbovo pravidlo učení pro logickou funkci „OR“ v bipolární reprezentaci.

Tato kapitola je úvodní kapitolou zabývající se problematikou vzájemného propojení neuronů, tj. ***architekturou*** neuronové sítě. Dále si zde ozřejmíme i způsob, jakým probíhá ***šíření*** a ***zpracování*** informace v neuronové síti.

Všechny zde uvedené pojmy doporučuji pečlivě nastudovat, protože je budeme dále velmi často používat.

Klíčová slova této kapitoly:

architektura (topologie) neuronové sítě, organizační dynamika neuronové sítě, aktivní dynamika neuronové sítě, adaptivní dynamika neuronové sítě, homogenní neuronová síť, učení s učitelem, samoorganizace.

Neuronová síť [6]

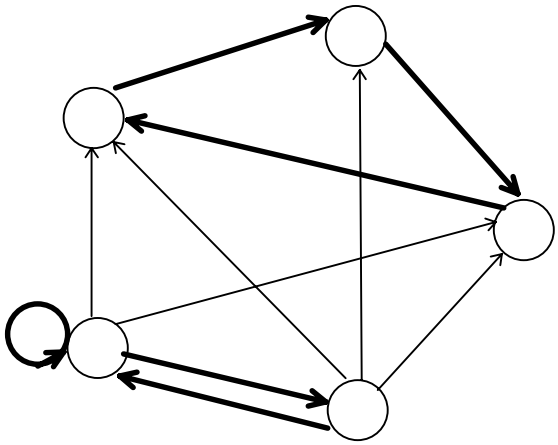
Každá neuronová síť je složena z formálních neuronů, které jsou vzájemně propojeny tak, že výstup jednoho neuronu je vstupem do (obecně i více) neuronů. Obdobně jsou terminály axonu biologického neuronu přes synaptické vazby spojeny s dendrity jiných neuronů. Počet neuronů a jejich vzájemné propojení v síti určuje *architekturu* (*topologii*) neuronové sítě. Z hlediska využití rozlišujeme v síti *vstupní*, *pracovní* (*skryté*, *mezilehlé, vnitřní*) a *výstupní* neurony. Šíření a zpracování informace v síti je umožněno změnou stavů neuronů ležících na cestě mezi vstupními a výstupními neurony. Stavy všech neuronů v síti určují *stav* neuronové sítě a synaptické váhy všech spojů představují *konfiguraci* neuronové sítě.

Neuronová síť se v čase vyvíjí, mění se stav neuronů, adaptují se váhy. V souvislosti se změnou těchto charakteristik v čase je účelné rozdělit celkovou *dynamiku* neuronové sítě do tří dynamik a uvažovat pak tři režimy práce sítě: *organizační* (změna topologie), *aktivní* (změna stavu) a *adaptivní* (změna konfigurace). Uvedené dynamiky neuronové sítě jsou obvykle zadány počátečním stavem a matematickou rovnicí, resp. pravidlem, které určuje vývoj příslušné charakteristiky sítě (topologie, stav, konfigurace) v čase. Změny, které se řídí těmito zákonitostmi probíhají v odpovídajících režimech práce neuronové sítě.

Konkretizací jednotlivých dynamik pak obdržíme různé *modely* neuronových sítí vhodné pro řešení různých tříd úloh.

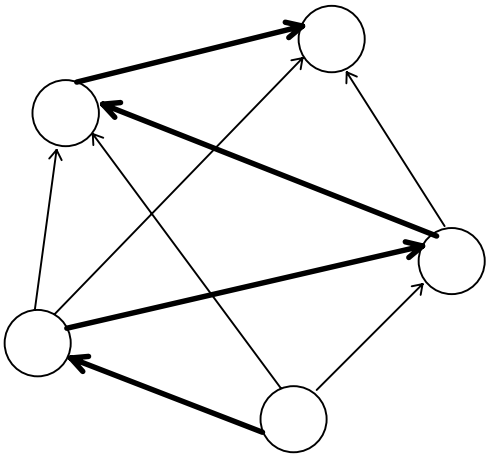
Organizační dynamika [6]

Organizační dynamika specifikuje architekturu neuronové sítě a její případnou změnu. Změna topologie se většinou uplatňuje v rámci adaptivního režimu tak, že síť je v případě potřeby rozšířena o další neurony a příslušné spoje. Avšak organizační dynamika převážně předpokládá pevnou architekturu neuronové sítě (tj. takovou architekturu, která se již v čase nemění). Rozlišujeme dva typy architektury: *cyklická* (*rekurentní*) a *acyklická* (*dopředná*) síť. V případě cyklické topologie existuje v síti skupina neuronů, která je spojena v kruhu (tzv. *cyklus*). To znamená, že v této skupině neuronů je výstup prvního neuronu vstupem druhého neuronu, jehož výstup je opět vstupem třetího neuronu atd., až výstup posledního neuronu v této skupině je vstupem prvního neuronu. Nejjednodušším příkladem cyklu je *zpětná vazba* neuronu, jehož výstup je zároveň jeho vstupem. Nejvíce cyklů je v *úplné topologii* cyklické neuronové sítě, kde výstup libovolného neuronu je vstupem každého neuronu. Příklad obecné cyklické neuronové sítě je uveden na obrázku 7, kde jsou vyznačeny všechny možné cykly.



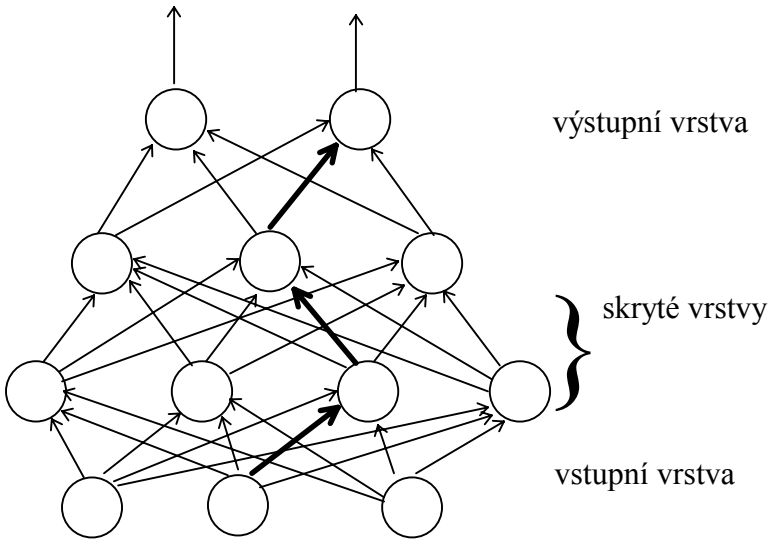
Obrázek 7: ***Příklad cyklické architektury.***

V acyklických sítích naopak cyklus neexistuje a všechny cesty vedou jedním směrem. Příklad acyklické sítě je na obrázku 8, kde je vyznačena nejdelší cesta.



Obrázek 8: ***Příklad acyklické architektury.***

U acyklické neuronové sítě lze neurony vždy (disjunktně) rozdělit do *vrstev*, které jsou uspořádány (např. nad sebou) tak, že spoje mezi neurony vedou jen z nižších vrstev do vrstev vyšších (obecně však mohou přeskočit jednu nebo i více vrstev). Speciálním případem takové architektury je *vícevrstvá neuronová síť*.



Obrázek 9: ***Příklad architektury vícevrstvé neuronové sítě 3-4-3-2.***

V této síti je první (dolní), tzv. *vstupní* vrstva tvořena vstupními neurony a poslední (horní), tzv. *výstupní* vrstva je složena z výstupních neuronů. Ostatní, tzv. *skryté* (*mezilehlé, vnitřní*) vrstvy jsou složeny ze skrytých (*vnitřních*) neuronů. V topologii vícevrstvé sítě jsou neurony jedné vrstvy spojeny se všemi neurony bezprostředně následující vrstvy. Proto architekturu takové sítě lze zadat jen počty neuronů v jednotlivých vrstvách (oddělených pomlčkou), v pořadí od vstupní k výstupní vrstvě. Také cesta v takové síti vede směrem od vstupní vrstvy k výstupní, přičemž obsahuje po jednom neuronu z každé vrstvy. Příklad architektury třívrstvé neuronové sítě 3-4-3-2 s jednou vyznačenou cestou je na obrázku 9, kde kromě vstupní a výstupní vrstvy jsou i dvě skryté vrstvy.

Aktivní dynamika [6]

Aktivní dynamika specifikuje *počáteční stav sítě* a způsob jeho změny v čase při pevné topologii a konfiguraci. V aktivním režimu se na začátku nastaví stavy vstupních neuronů na tzv. *vstup sítě* a zbylé neurony jsou v uvedeném počátečním stavu. Všechny možné vstupy, resp. stavy sítě, tvoří *vstupní prostor*, resp. *stavový prostor*, neuronové sítě. Po inicializaci stavu sítě probíhá vlastní výpočet. Obecně se předpokládá spojitý vývoj stavu neuronové sítě v čase a hovoří se o *spojitém modelu*, kdy stav sítě je spojitou funkcí času, která je obvykle v aktivní dynamice zadána diferenciální rovnicí. Většinou se však předpokládá diskrétní čas, tj. na počátku se síť nachází v čase 0 a stav sítě se mění jen v čase 1, 2, 3, ... V každém takové časové kroku je podle daného pravidla aktivní dynamiky vybrán jeden neuron (tzv. *sekvenční výpočet*) nebo více neuronů (tzv. *paralelní výpočet*), které *aktualizují* (mění) svůj stav na základě svých vstupů, tj. stavů sousedních neuronů, jejichž výstupy jsou vstupy aktualizovaných neuronů. Podle toho, zda neurony mění svůj stav nezávisle na sobě nebo je jejich aktualizace řízena centrálně, rozlišujeme *synchronní* a *asynchronní* modely neuronových sítí. Stav výstupních neuronů, který se obecně mění v čase, je *výstupem* neuronové sítě (tj. výsledkem výpočtu). Obvykle se však uvažuje taková aktivní dynamika, že výstup sítě je po nějakém čase konstantní a neuronová síť tak v aktivním režimu realizuje nějakou funkci na vstupním prostoru, tj. ke každému vstupu sítě vypočítá právě jeden výstup. Tato tzv. *funkce neuronové sítě* je dána aktivní dynamikou, jejíž rovnice parametricky závisí na topologii a konfiguraci, které se v aktivním režimu, jak již bylo uvedeno, nemění. Je zřejmé, že v aktivním režimu se neuronová síť využívá k vlastním výpočtům.

Aktivní dynamika neuronové sítě také určuje funkci jednoho neuronu, jejíž předpis (matematický vzorec) je většinou pro všechny (nevstupní) neurony v síti stejný (tzv. *homogenní* neuronová síť). Můžeme se setkat s následujícími *sigmoidními aktivačními funkcemi*:

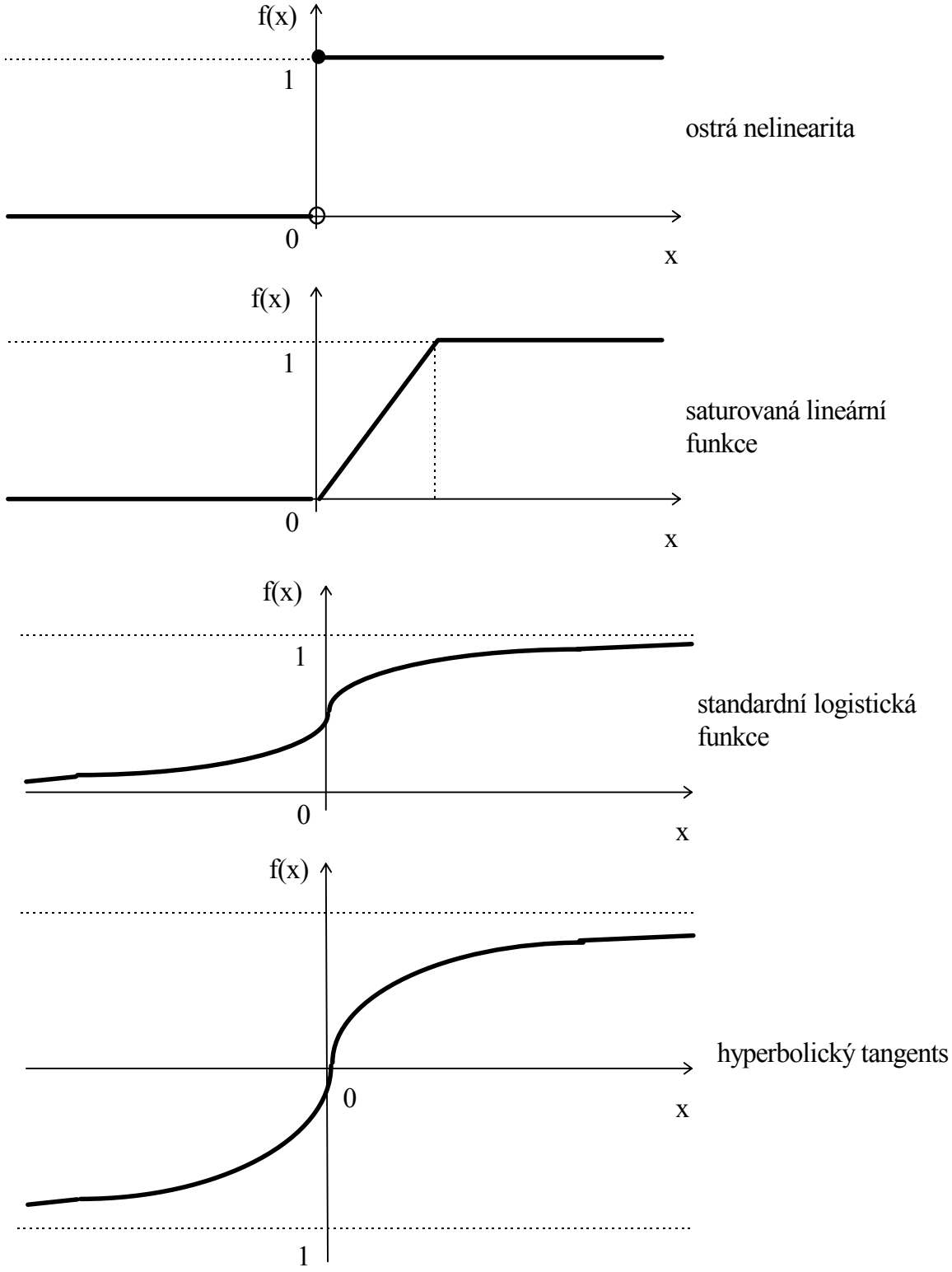
$$f(x) = \begin{cases} 1 & \text{pokud } x \geq 1 \\ & \\ 0 & \text{pokud } x < 0 \end{cases} \quad \text{ostrá nelinearita}$$

$$f(x) = \begin{cases} 1 & x \geq 1 \\ x & 0 \leq x \leq 1 \\ 0 & x < 0 \end{cases} \quad \text{saturovaná lineární funkce}$$

$$f(x) = \frac{1}{1+e^{-x}} \quad \text{standardní (logistická) sigmoida}$$

$$f(x) = \frac{1-e^{-x}}{1+e^{-x}} \quad \text{hyperbolický tangens}$$

Grafy těchto funkcí jsou znázorněny na obrázku 10. Podle toho, zda je funkce neuronu diskrétní nebo spojitá rozlišujeme *diskrétní* a *analogové* modely neuronových sítí.



Obrázek 10: ***Grafy sigmoidních aktivačních funkcí***

Adaptivní dynamika [6]

Adaptivní dynamika neuronové sítě specifikuje *počáteční konfiguraci* sítě a způsob, jakým se mění váhové hodnoty na spojeních mezi jednotlivými neurony v čase. Všechny možné konfigurace sítě tvoří *váhový*

prostor neuronové sítě. V adaptivním režimu se tedy na začátku nastaví váhy všech spojů v síti na počáteční konfiguraci (např. náhodně). Po inicializaci konfigurace sítě probíhá vlastní adaptace. Podobně jako v aktivní dynamice se obecně uvažuje spojitý model se spojitým vývojem konfigurace neuronové sítě v čase, kdy váhy sítě jsou (spojitou) funkcí času, která je obvykle v adaptivní dynamice zadána diferenciální rovnicí. Většinou se však předpokládá diskrétní čas adaptace.

Víme, že funkce sítě v aktivním režimu závisí na konfiguraci. Cílem adaptace je nalézt takovou konfiguraci sítě ve váhovém prostoru, která by v aktivním režimu realizovala předepsanou funkci. Jestliže aktivní režim sítě se využívá k vlastnímu výpočtu funkce sítě pro daný vstup, pak adaptivní režim slouží k *učení* („programování“) této funkce.

Požadovaná funkce sítě je obvykle zadána tzv. *tréninkovou množinou* (*posloupností*) dvojic vstup/výstup sítě (tzv. *tréninkový vzor*). Způsobu adaptace, kdy požadované chování sítě modeluje učitel, který pro vzorové vstupy sítě informuje adaptivní mechanismus o správném výstupu sítě, se říká *učení s učitelem* (supervised learning). Někdy učitel hodnotí kvalitu momentální skutečné odpovědi (výstupu) sítě pro daný vzorový vstup pomocí známky, která je zadána místo požadované hodnoty výstupu sítě (tzv. *klasifikované učení*). Jiným typem adaptace je tzv. *samoorganizace*. V tomto případě tréninková množina obsahuje jen vstupy sítě. To modeluje situaci, kdy není k dispozici učitel, proto se tomuto způsobu adaptace také říká *učení bez učitele*. Neuronová síť v adaptivním režimu sama organizuje tréninkové vzory (např. do shluků) a odhaluje jejich souborné vlastnosti.

Úkoly:

Zopakujte si všechny základní pojmy této kapitoly (viz „KLÍČOVÁ SLOVA“ kapitoly)

P E R C E P T R O N .

Při popisu algoritmu adaptace perceptronu budeme používat ***značení***, které je uvedeno v kapitole „*Úvod do problematiky neuronových sítí*“.

Perceptron je nejjednodušší neuronová síť s jedním pracovním neuronem a na jeho adaptačním algoritmu si vysvětlíme proces učení s učitelem.

Klíčová slova této kapitoly:

perceptron, adaptační pravidlo perceptronu, koeficient učení, práh.

Perceptron

Autorem této nejjednodušší neuronové sítě je Frank Rosenblatt (r. 1957). Za typický perceptron je považována jednoduchá neuronová síť s n vstupy ($x_1, x_2, ..., x_n$) a jedním pracovním neuronem spojeným se všemi svými vstupy. Každému takovému spojení je přiřazena váhová hodnota ($w_1, w_2, ..., w_n$). Signál přenášený vstupními neurony je buď binární (tj. má hodnotu 0 nebo 1), nebo bipolární (tj. má hodnotu -1, 0 nebo 1). Výstupem z perceptronu je pak $y = f(y\_in)$, kde aktivační funkce f má tvar (θ je libovolný, ale pevný práh aktivační funkce f):

$$f(y\_in) = \begin{cases} 1 & pokud\ y\_in > \theta \\ 0 & pokud\ -\theta \le y\_in \le \theta \\ -1 & pokud\ y\_in < -\theta \end{cases}$$

Váhové hodnoty jsou adaptovány podle adaptačního pravidla perceptronu tak, aby diference mezi skutečným a požadovaným výstupem byla co nejmenší. Adaptační pravidlo perceptronu je mnohem silnější než Hebbovo adaptační pravidlo.

Popis algoritmu

Krok 0. Inicializace vah w_i *(i = 1* až *n)* a biasu *b malými náhodnými čísly.*

Přiřazení inicializační hodnoty koeficientu učení α *(0* $< \alpha \le$ *1).*

Krok 1. Dokud není splněna podmínka ukončení výpočtu, opakovat kroky (2 až 6).

Krok 2. Pro každý tréninkový pár **s:t** (tj. vstupní vektor **s** a příslušný výstup **t**), provádět kroky (3 až 5).

Krok 3. Aktivuj vstupní neurony:

$x_i = s_i$.

Krok 4 Vypočítej skutečnou hodnotu na výstupu:

$$y\_in = b + \sum_i x_i w_i ;.$$

$$y = \begin{cases} 1 & pokud\ y\_in > \theta \\ 0 & pokud - \theta \le y\_in \le \theta \\ -1 & pokud\ y\_in < -\theta \end{cases}$$

Krok 5 Aktualizuj váhové hodnoty a bias pro daný vzor

jestliže $y \neq t$,

$w_i(new) = w_i(old) + \alpha\, t\, x_i$ $(i = 1$ až $n)$.

$b(new) = b(old) + \alpha\, t$.

jinak

$w_i(new) = w_i(old)$

$b(new) = b(old)$

Krok 6. Podmínka ukončení:

jestliže ve *2. kroku* již nenastává žádná změna váhových hodnot, stop; jinak, pokračovat.

Aktualizaci podléhají pouze ty váhové hodnoty, které neprodukují požadovaný výstup *y*. To znamená, že čím více tréninkových vzorů má korektní výstupy, tím méně je potřeba času k jejich tréninku. Práh aktivační funkce je pevná nezáporná hodnota θ. Tvar aktivační funkce pracovního neuronu je takový, že umožňuje vznik pásu pevné šířky (určené hodnotou θ) oddělujícího oblast pozitivní odezvy od oblasti negativní odezvy na vstupní signál. Předcházející analýza o zaměnitelnosti prahu a biasu zde nemá uplatnění, protože změna θ mění šířku oblasti, ne však její umístění. Místo jedné separující přímky tedy máme pás určený dvěma rovnoběžnými přímkami:

1. Přímka separující oblast pozitivní odezvy od oblasti nulové odezvy na vstupní signál; tato hraniční přímka má tvar:

$$w_1 x_1 + w_2 x_2 + b > \theta.$$

2. Přímka separující oblast nulové odezvy od oblasti negativní odezvy na vstupní signál;. tato hraniční přímka má tvar:

$$w_1 x_1 + w_2 x_2 + b < -\theta.$$

Příklad:

Adaptační algoritmus perceptronu pro logickou funkci „AND“: binární vstupní hodnoty, bipolární výstupní hodnoty. Pro jednoduchost předpokládejme, že $\theta = 0{,}2$ a $\alpha = 1$.

VSTUP			POŽADOVANÝ VÝSTUP
x_1	x_2	b	
1	1	1	1
1	0	1	-1
0	1	1	-1
0	0	1	-1

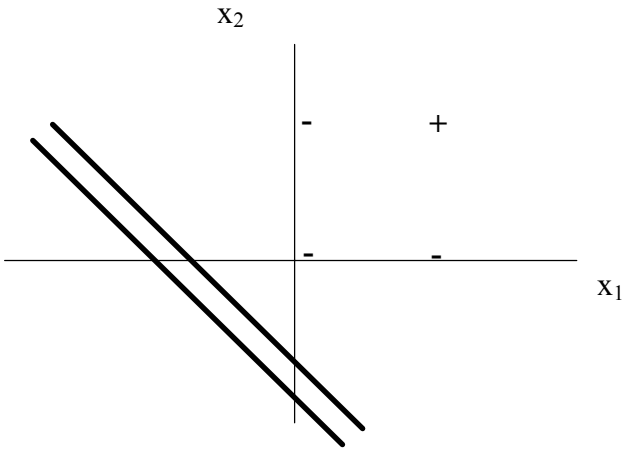
Po předložení *prvního* tréninkového vzoru, dostáváme následující:

VSTUP			VÝSTUP			PŘÍRUSTKY VAH			VÁHOVÉ HODNOTY		
x_1	x_2	b	y_in	y	t	Δw_1	Δw_2	Δb	w_1	w_2	b
									0	0	0
1	1	1	0	0	1	1	1	1	1	1	1

Separující přímky jsou dány rovnicemi

$$x_1 + x_2 + 1 = 0{,}2$$

$$x_1 + x_2 + 1 = -0{,}2$$



Obrázek 11: *Hraniční pás pro logickou funkci „AND“ - první tréninkový vzor.*

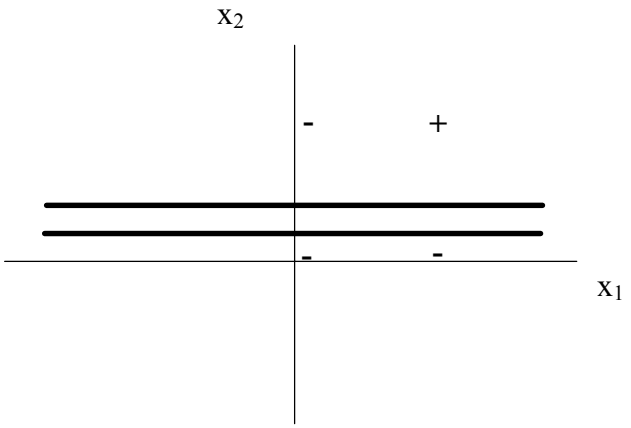
Předložíme-li *druhý* tréninkový vzor, dostáváme následující:

VSTUP			VÝSTUP			PŘÍRUSTKY VAH			VÁHOVÉ HODNOTY		
x_1	x_2	b	y_in	y	t	Δw_1	Δw_2	Δb	w_1	w_2	b
									1	1	1
1	0	1	2	1	-1	-1	0	-1	0	1	0

Separující přímky mají tvar

$$x_2 = 0{,}2$$

$$x_2 = -0{,}2$$



Obrázek 12: *Hraniční pás pro logickou funkci „AND“ - druhý tréninkový vzor*

Po předložení *třetího* tréninkového vzoru, dostáváme:

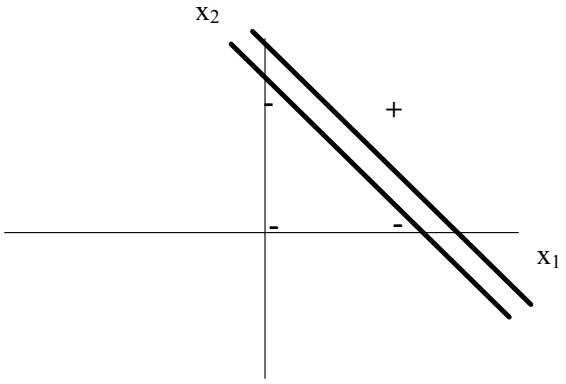
VSTUP			VÝSTUP			PŘÍRUSTKY VAH			VÁHOVÉ HODNOTY		
x_1	x_2	b	y_in	y	t	Δw_1	Δw_2	Δb	w_1	w_2	b
									0	1	0
0	1	1	1	1	-1	0	-1	-1	0	0	-1

Pro úplnost prvního tréninkového cyklu předložíme i *čtvrtý* vzor a dostáváme následující:

VSTUP			VÝSTUP			PŘÍRUSTKY VAH			VÁHOVÉ HODNOTY		
x_1	x_2	b	y_in	y	t	Δw_1	Δw_2	Δb	w_1	w_2	b
									0	0	-1
0	0	1	-1	-1	-1	0	0	0	0	0	-1

Výsledky po desátém tréninkovém cyklu jsou:

1	1	1	1	1	1	0	0	0	2	3	-4
1	0	1	-2	-1	-1	0	0	0	2	3	-4
0	1	1	-1	-1	-1	0	0	0	2	3	-4
0	0	1	-4	-1	-1	0	0	0	2	3	-4



Obrázek 13: *Hraniční pás pro logickou funkci „AND“ po adaptaci algoritmem perceptronu.*

Kladná odezva je dána všemi body, pro které platí

$$2x_1 + 3x_2 - 4 > 0{,}2.$$

Hraniční přímka oblasti má tvar

$$x_2 = -\frac{2}{3}x_1 + \frac{7}{5}.$$

Záporná odezva je dána všemi body, pro které platí

$$2x_1 + 3x_2 - 4 < -0{,}2.$$

Hraniční přímka oblasti má pak tvar

$$x_2 = -\frac{2}{3}x_1 + \frac{19}{15}.$$

Úkoly:

1. Srovnejte Hebbovo adaptační pravidlo a adaptační pravidlo perceptronu.

2. Objasněte adaptační algoritmus perceptronu pro logickou funkci „OR“ v bipolární reprezentaci.

Korespondenční úkoly:

Vytvořte počítačový program pro realizaci adaptačního algoritmu perceptronu.

A D A L I N E .
M A D A L I N E .

Při popisu adaptačního algoritmu pro Adaline a Madaline budeme vycházet ze ***značení***, které je uvedeno v kapitole „*Úvod do problematiky neuronových sítí*“ a které bude v této kapitole rozšířeno.

Adaptační algoritmus neuronu Adaline bude srovnán s adaptačním algoritmem percetronu.

V závěru pak budou uvedeny možnosti klasifikace různých typů neuronových sítí (tj. 1-vrstvé, vrstvé a 3-vrstvé neuronové sítě).

Klíčová slova této kapitoly:

Adaline, Madaline, adaptační algoritmus pro Adaline, delta pravidlo.

Adaline

Adaline, tj. *Adaptive Linear Neuron*. Pro své vstupy obvykle používá bipolární aktivaci (1 nebo -1), výstupní hodnota je nejčastěji také bipolární. Adaline má rovněž bias chovající se jako regulovatelná váha (w_0) přiřazená spojení, které vychází z neuronu, jehož aktivace je vždy 1.

Adaptační algoritmus pro Adaline má následující tvar:

Krok 0. Inicializace vah malými náhodnými hodnotami.

Přiřazení inicializační hodnoty koeficientu učení α *(viz poznámky za algoritmem).*

Krok 1. Dokud není splněna podmínka ukončení výpočtu, opakovat kroky (2 až 6).

Krok 2. Pro každý bipolární tréninkový pár **s**:t (tj. vstupní vektor **s** a příslušný výstup t), provádět kroky (3 až 5).

Krok 3. Aktivovat vstupní neurony:

$x_i = s_i$.

Krok 4 Vypočítat skutečnou hodnotu na výstupu:

$$y\_in = b + \sum_i x_i w_i ;.$$

$y = y\_in$.

Krok 5 Aktualizovat váhové hodnoty a $i = 1, ..., n$:

$w_i(new) = w_i(old) + \alpha\ (t - y\_in)\ x_i$.

$b(new) = b(old) + \alpha\ (t - y\_in)$.

Krok 6. Podmínka ukončení:

jestliže největší změna váhových hodnot, která se vyskytuje v kroku 2 je menší než maximální povolená chyba, stop; jinak, pokračovat.

Nastavení vhodné hodnoty koeficientu učení α se děje následovně:

Podle Hecht-Nielsena lze za jeho horní hraniční hodnotu považovat největší vlastní číslo korelační matice *R* vstupu (řádku) vektoru ***x(p)***,

$$R = \frac{1}{P}\sum_{p=1}^{P} \mathbf{x}(p)^T \mathbf{x}(p),$$

tedy

α < *jedna polovina největší hodnoty vlastního čísla R.*

Jelikož hodnota *R* není během výpočtu měněna, obvykle se volí i α jako $0.1 < n\alpha < 1.0$, kde *n* je počet vstupů. Pokud dosadíme za α příliš velkou hodnotu, adaptační algoritmus nebude konvergovat. Pokud dosadíme za α příliš malou hodnotu, proces učení bude extrémně pomalý.

Důkaz konvergence adaptačního pravidla pro Adaline je obsažen v derivaci *delta pravidla*. *Delta pravidlo* mění váhové hodnoty na spojeních mezi jednotlivými neurony tak, aby byl minimalizován rozdíl mezi vstupním signálem *y_in* výstupního neuronu a požadovaným výstupem *t*. Cílem adaptace je minimalizovat tuto chybu přes všechny tréninkové vzory. Příslušné váhové korekce jsou akumulovány a po každém tréninkovém cyklu jsou všechny váhové hodnoty aktualizovány najednou.

Delta pravidlo příslušející *I.* váhové hodnotě je pro každý vzor zapsáno následovně:

$$\Delta w_I = \alpha\left(t - y\_in\right) x_I$$

Dále budeme používat toto označení:

x	Vektor aktivací vstupních neuronů, má *n* složek.
y_in	Hodnota vstupního signálu výstupního neuronu *Y* je $y\_in = \sum_{i=1}^{n} x_i w_i$.
t	Požadovaný výstup.

Derivace:

Pro každý tréninkový vzor je dána chybová funkce *E = E(**w**)*, tj. funkce všech váhových hodnot w_i, *i = 1, ..., n* vztahem

$$E = (t - y\_in)^2$$

Gradient *E* je vektor, jehož složky jsou parciální derivace *E* podle všech složek vektoru ***w***. Gradient udává směr největšího růstu (chyby *E*); pokud však má opačný směr způsobuje její nejrychlejší zmenšování. Chybová funkce *E* je minimalizována prostřednictvím úprav váhových hodnot w_I ve směru $-\frac{\partial E}{\partial w_I}$.

Protože $y\_in = \sum_{i=1}^{n} x_i w_i$,

$$\frac{\partial E}{\partial w_I} = -2\left(t - y\_in\right)\frac{\partial\, y\_in}{\partial\, w_I}$$

$$= -2\left(t - y\_in\right) x_I .$$

Chyba (E) bude tedy redukována rychleji, pokud budou příslušné váhové hodnoty upravovány podle delta pravidla,

$$\Delta w_I = \alpha\left(t - y\_in\right) x_I .$$

Příklad:

Adaptační algoritmus Adaline pro logickou funkci „OR“ (bipolární vstupní i výstupní hodnoty) je zapsán následovně:

VSTUP		POŽADOVANÝ VÝSTUP
x_1	x_2	
1	1	1
1	-1	1
-1	1	1
-1	-1	-1

Jak již bylo výše naznačeno, je adaptační algoritmus Adaline navržen k nalezení takových váhových hodnot w_i, aby minimalizovaly celkovou chybu

$$E = \sum_{p=1}^{4}\left(x_1(p)w_1 + x_2(p)w_2 + w_0 - t(p)\right)^2 ,$$

kde

$$x_1(p)w_1 + x_2(p)w_2 + w_0$$

je vstupní signál vedoucí do výstupního neuronu pro vzor p a $t(p)$ je požadovaný výstup příslušející vzoru p.

Váhové hodnoty, které minimalizují chybovou funkci, mají v tomto příkladě tvar:

$$w_1 = \frac{1}{2},$$

$$w_2 = \frac{1}{2},$$

$$\text{bias} \quad w_0 = \frac{1}{2}.$$

Separující přímka je tedy určená rovnicí

$$\frac{1}{2}x_1 + \frac{1}{2}x_2 + \frac{1}{2} = 0.$$

Geometrický význam funkce Adaline se nepatrně liší od perceptronu. Uvažujme vstup $\boldsymbol{x}=(x_1, ..., x_n)$, tj. bod $[x_1, ..., x_n]$ v n-rozměrném vstupním prostoru. Nadrovina s koeficienty $\boldsymbol{w}$ pro daný neuron Adaline určená rovnicí

$$w_0 + \sum_{i=1}^{n} w_i x_i = 0$$

rozděluje tento prostor na dva poloprostory, ve kterých má hodnota výstupu y zapsaného rovnicí

$$y = \sum_{i=1}^{n} w_i x_i$$

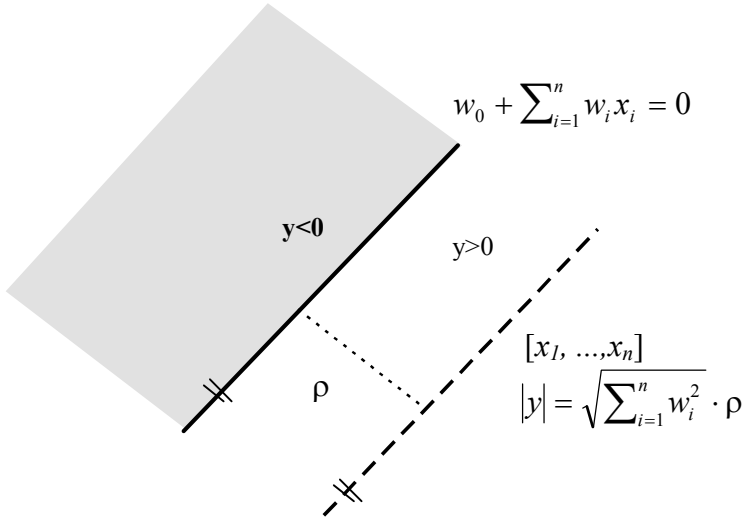
odlišné znaménko (tj. je buď kladná, nebo záporná). Pro body ležící na této nadrovině je hodnota výstupu nulová. Vzdálenost ρ bodu $[x_1, ..., x_n]$ od této nadroviny je dána rovnicí:

$$\rho = \frac{\left| w_0 + \sum_{i=1}^{n} w_i x_i \right|}{\sqrt{\sum_{i=1}^{n} w_i^2}} = \frac{\left| y_j \right|}{\sqrt{\sum_{i=1}^{n} w_i^2}}.$$

Tedy absolutní hodnota $|y|$ výstupu z neuronu Adaline závisí lineárně na vzdálenosti bodu od nadroviny ve vstupním prostoru:

$$\left| y_j \right| = \sqrt{\sum_{i=1}^{n} w_i^2} \cdot \rho.$$

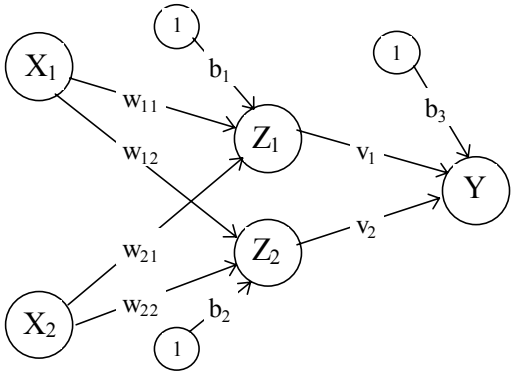
Body ze vstupního prostoru, které mají stejný výstup, leží na jedné nadrovině rovnoběžné s nadrovinou $w_0 + \sum_{i=1}^{n} w_i x_i = 0$, která je od ní ve vzdálenosti ρ ve směru daném znaménkem *y*. Uvedená situace je načrtnuta na obrázku 14, kde nadrovina určená stejným výstupem je znázorněna přerušovanou čarou.



Obrázek 14: ***Geometrická interpretace funkce neuronu Adaline.***

Madaline

Madaline, tj. *Many Adaptive Linear Neurons*. Základním prvkem v tomto modelu je neuron Adaline, který je velmi podobný perceptronu (viz předcházející kapitola). Jednoduchá architektura neuronové sítě Madaline je zobrazena na obrázku 15. Výstupy (z_1 a z_2) z obou skrytých neuronů typu Adaline (Z_1 a Z_2), jsou určeny stejnými signály (x_1 a x_2) vycházejícími z neuronů X_1 a X_2, které samozřejmě závisí na příslušné prahové funkci. Pak i skutečný výstup y je nelineární funkcí vstupního vektoru (x_1, x_2) a příslušné prahové funkce. Použití skrytých neuronů Z_1 a Z_2 sice dává síti větší výpočtové možnosti, ale naproti tomu komplikuje adaptační proces.



Obrázek 15: ***Madaline se dvěma skrytými neurony Adaline a jedním výstupním neuronem Adaline.***

Původní adaptační algoritmus MRI (z roku 1960) adaptuje pouze váhové hodnoty příslušející oběma skrytým neuronům, zatímco váhové hodnoty příslušející výstupnímu neuronu jsou fixní. Adaptační algoritmus MRII (z roku 1987) upravuje všechny váhové hodnoty. Dále budeme pracovat pouze s adaptačním algoritmem MRI: Váhové hodnoty v_1 a v_2 a bias b_3 , příslušející výstupnímu neuronu Y, jsou určeny tak, že výstupní signál z Y je roven 1, pokud je alespoň jedna hodnota signálu vycházejícího ze skrytých neuronů (tj. Z_1 a Z_2 nebo obou z nich) rovna jedné. Pokud jsou oba signály vysílané ze Z_1 i Z_2 rovny -1, má výstupní signál z Y hodnotu -1. Jinými slovy, výstupní neuron Y provádí logickou funkci „OR“ na signálech vysílaných z neuronů Z_1 a Z_2 . Můžeme tedy přiřadit

$$v_1 = \frac{1}{2},$$
$$v_2 = \frac{1}{2},$$
$$b_3 = \frac{1}{2}.$$

Váhové hodnoty příslušející prvnímu skrytému neuronu Adaline (w_{11} a w_{21}) a váhové hodnoty příslušející druhému skrytému neuronu Adaline (w_{12} a w_{22}) jsou adaptovány podle algoritmu MRI takto:

Aktivační funkce pro Z_1, Z_2 a Y je dána následovně:

$$f(x) = \begin{cases} 1 & \text{pokud } x \geq 0; \\ -1 & \text{pokud } x < 0. \end{cases}$$

Adaptační algoritmus MRI

Krok 0. Váhové hodnoty v_1 a v_2 a bias b_3 jsou inicializovány výše uvedeným způsobem.
Inicializace zbývajících vah malými náhodnými hodnotami.
Přiřazení inicializační hodnoty koeficientu učení α. stejným způsobem jako v adaptačním algoritmu pro neuron Adaline.

Krok 1. Dokud není splněna podmínka ukončení výpočtu, opakovat kroky (2 až 8).

Krok 2. Pro každý bipolární tréninkový pár **s:t** provádět kroky (3 až 7).

Krok 3. Aktivovat vstupní neurony:

$x_i = s_i$.

Krok 4 Vypočítat vstupní hodnoty skrytých neuronů:

$$z\_in_1 = b_1 + x_1 w_{11} + x_2 w_{21},$$
$$z\_in_2 = b_2 + x_1 w_{12} + x_2 w_{22}.$$

Krok 5 Stanovení výstupních hodnot skrytých neuronů:

$$z_1 = f\left(z\_in_1\right),$$
$$z_2 = f\left(z\_in_2\right).$$

Krok 6 Stanovení skutečné výstupní hodnoty signálu neuronové sítě Madaline:

$$y\_in = b_3 + z_1 v_1 + z_2 v_2;$$
$$y = f\left(y\_in\right).$$

Krok 7 Aktualizovat váhové hodnoty*:*

Pokud je $y = t$, nenastávají žádné změny.

Jinak (pro $y \neq t$)*:*

Je-li $t = 1$, potom pro váhové hodnoty na spojeních vedoucích k Z_J ($J=1,2$) platí:

$w_{iJ}(new) = w_{iJ}(old) + \alpha\ (1 - z\_in_J)\ x_i$.

$b_J(new) = b_J(old) + \alpha\ (1 - z\_in_J)$.

Je-li $t = -1$, potom pro váhové hodnoty na spojeních vedoucích k Z_K ($K=1,2$) platí:

$w_{iK}(new) = w_{iK}(old) + \alpha\ (-1 - z\_in_K)\ x_i$.

$b_K(new) = b_K(old) + \alpha\ (-1 - z\_in_K)$.

Krok 8. Podmínka ukončení:

pokud již nenastávají žádné změny váhových hodnot nebo pokud již bylo vykonáno maximálně definované množství váhových změn, stop; jinak, pokračovat.

Příklad:

Adaptační algoritmus MRI pro logickou funkci „XOR“ (bipolární vstupní i výstupní hodnoty) je zapsán následovně:

VSTUP		POŽADOVANÝ VÝSTUP
x_1	x_2	
1	1	-1
1	-1	1
-1	1	1
-1	-1	-1

Krok 0. $\alpha = 0.5$;

Inicializace váhových hodnot:

váhy vedoucí do Z_1			**váhy vedoucí do Z_2**			**váhy vedoucí do Y**		
w_{11}	w_{21}	b_1	w_{12}	w_{22}	b_2	v_1	v_2	b_3
0.05	0.2	0.3	0.1	0.2	0.15	0.5	0.5	0.5

Krok 1. Adaptace:

Krok 2. Pro první tréninkový pár; (1,1):-1

Krok 3. $x_1 = 1$,

$x_2 = 1$.

Krok 4 $z\_in_1 = 0.3 + 0.05 + 0.2 = 0.55$,

$z\_in_2 = 0.15 + 0.1 + 0.2 = 0.45$

Krok 5 $z_1 = 1$,

$z_2 = 1$.

Krok 6 $y\_in = 0.5 + 0.5 + 0.5$;

$y = 1$.

Krok 7 $t - y = -1-1 = -2 \neq 0$,

Pokud je $t = -1$, potom aktualizovat váhové hodnoty na spojeních vedoucích k Z_1:

$$b_1(new) = b_1(old) + \alpha(-1 - z\_in_1)$$
$$= 0.3 + (0.5)(-1.55)$$
$$= -0.475$$
$$w_{11}(new) = w_{11}(old) + \alpha(-1 - z\_in_1)x_1$$
$$= 0.05 + (0.5)(-1.55)$$
$$= -0.725$$
$$w_{21}(new) = w_{21}(old) + \alpha(-1 - z\_in_1)x_2$$
$$= 0.2 + (0.5)(-1.55)$$
$$= -0.575$$

a aktualizovat váhové hodnoty na spojeních vedoucích k Z_2:

$$
\begin{aligned}
b_2(new) &= b_2(old) + \alpha(-1 - z\_in_2) \\
&= 0.15 + (0.5)(-1.45) \\
&= -0.575 \\
w_{12}(new) &= w_{12}(old) + \alpha(-1 - z\_in_2)x_1 \\
&= 0.1 + (0.5)(-1.45) \\
&= -0.625 \\
w_{22}(new) &= w_{22}(old) + \alpha(-1 - z\_in_2)x_2 \\
&= 0.2 + (0.5)(-1.45) \\
&= -0.525
\end{aligned}
$$

Po čtyřech tréninkových cyklech, byly nalezeny tyto váhové hodnoty:

w_{11} = - 0.73	w_{12} = 1.27
w_{21} =1.53	w_{22} = - 1.33
b_1 = - 0.99	b_2 = - 1.09

Geometrická interpretace nalezených váhových hodnot:

Oblast kladné odezvy vznikne sjednocením obou oblastí pozitivní odezvy skrytých neuronů Z_1 a Z_2.

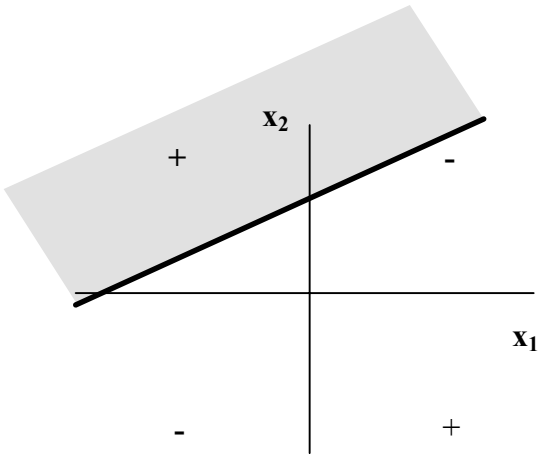
Pro skrytý neuron Z_1 má hraniční přímka tvar

$$
\begin{aligned}
x_2 &= -\frac{w_{11}}{w_{21}}x_1 - \frac{b_1}{w_{21}} \\
&= \frac{0.73}{1.53}x_1 + \frac{0.99}{1.53} \\
&= 0.48x_1 + 0.65.
\end{aligned}
$$

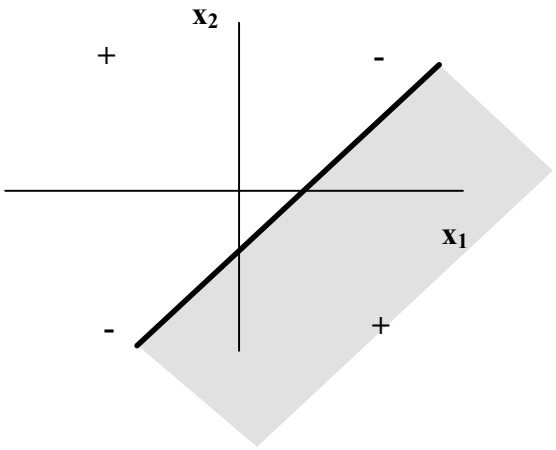
Pro skrytý neuron Z_2 má hraniční přímka tvar

$$
\begin{aligned}
x_2 &= -\frac{w_{12}}{w_{221}}x_1 - \frac{b_2}{w_{22}} \\
&= \frac{1.27}{1.33}x_1 + \frac{1.09}{1.33} \\
&= 0.96x_1 - 0.82.
\end{aligned}
$$

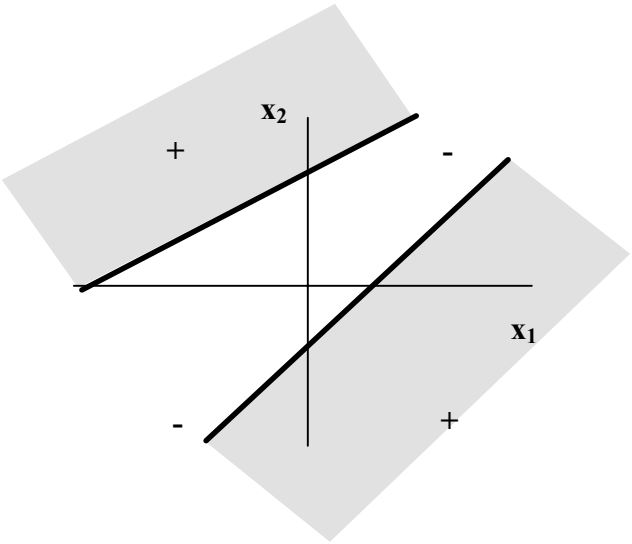
Vypočítané oblasti kladné a záporné odezvy na vstupní signál jsou znázorněny na následujících obrázcích.



Obrázek 16: ***Oblast kladné odezvy pro Z_1.***



Obrázek 17: ***Oblast kladné odezvy pro Z_2.***



Obrázek 18: ***Oblast kladné odezvy pro Madaline pro „XOR“ funkci.***

Úkoly:

1. Srovnejte adaptační algoritmus neuronu Adaline a perceptronu.

2. Srovnejte geometrickou interpretaci funkce neuronu Adaline a perceptronu.

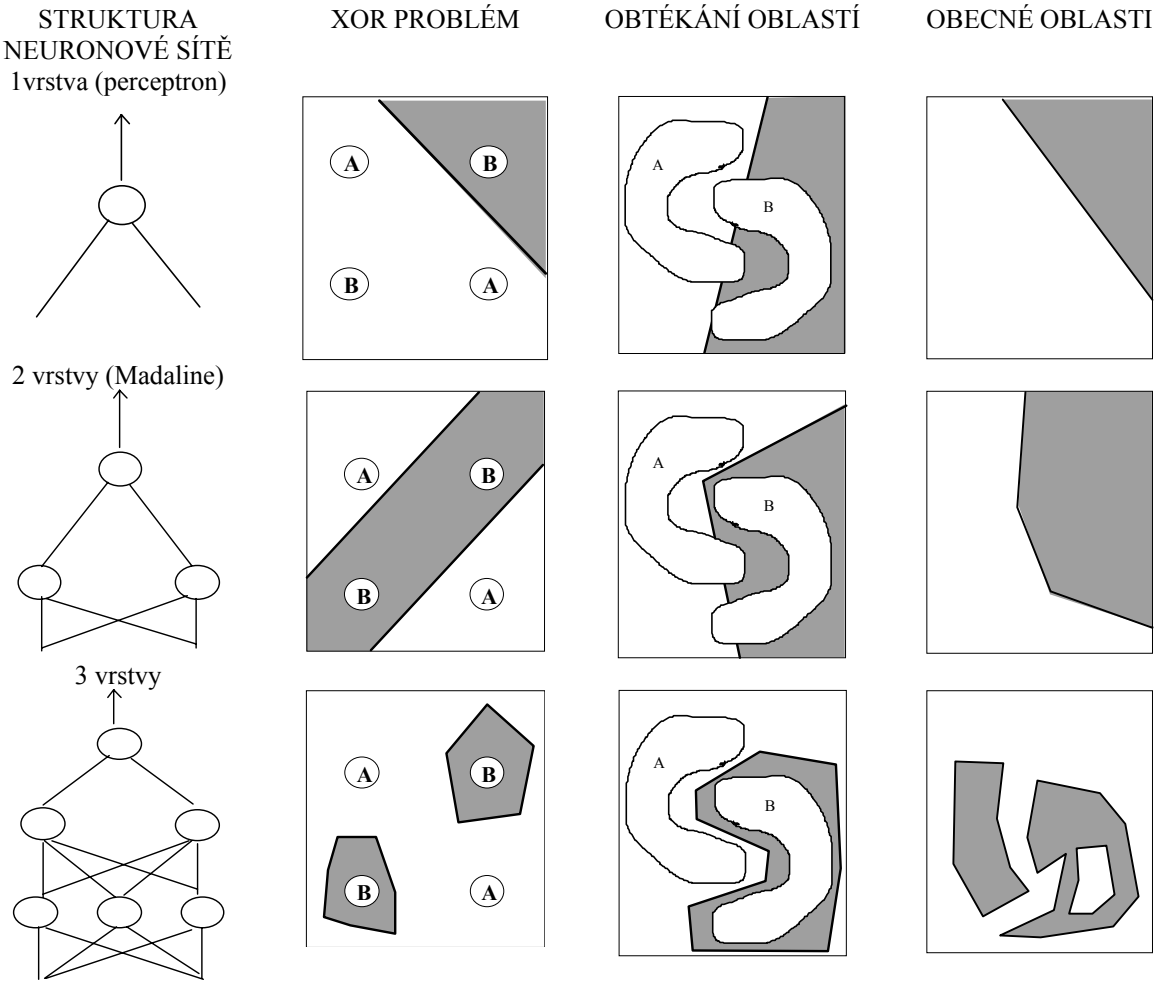
3. Objasněte adaptační algoritmus MRI pro vybranou logickou funkci v bipolární reprezentaci.

Korespondenční úkoly (vybraný úkol vykonejte):

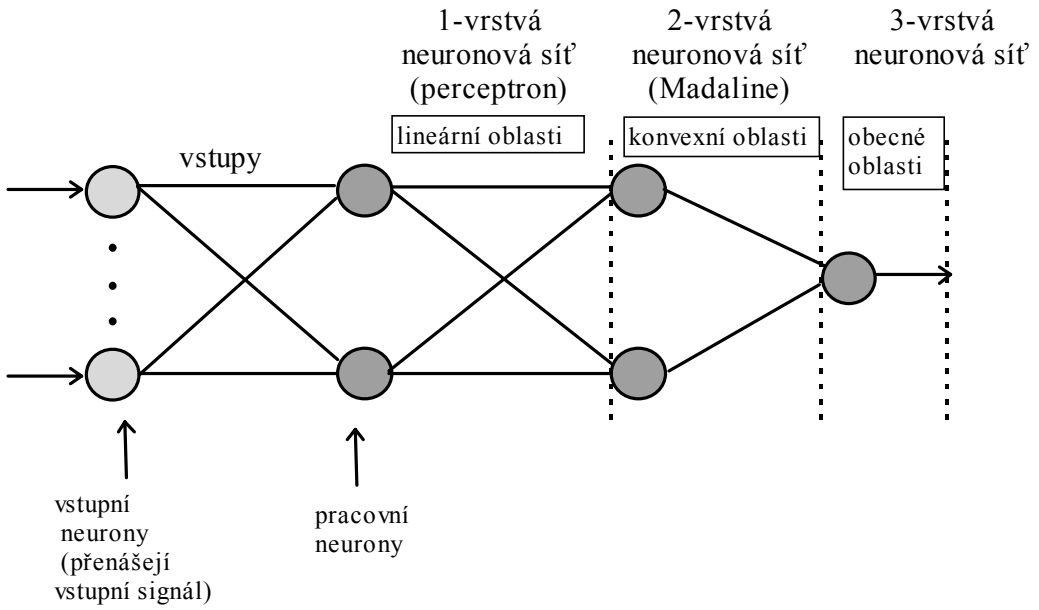
1. Vytvořte počítačový program pro realizaci adaptačního algoritmu neuronu Adaline.

2. Vytvořte počítačový program pro realizaci adaptačního algoritmu MRI.

Shrnutí: Na následujících dvou obrázcích jsou souhrnně zobrazeny různé tyty neuronových sítí (tj. neuronové sítě s různým počtem vnitřních vrstev) a jejich možnosti klasifikace.



Obrázek 19: ***Neuronové sítě s různým počtem vnitřních vrstev a jejich možnosti klasifikace.***



Obrázek 20: Mezní oblasti rozpoznávané neuronovou sítí s různým počtem vnitřních vrstev.

B A C K P R O P A G A T I O N .

V této kapitole se podrobně seznámíte s adaptačním algoritmem zpětného šíření chyby (*backpropagation*), jež je používán v přibližně 80% všech aplikací neuronových (tj. je nejrozšířenějším adaptačním algoritmem vícevrstvých neuronových sítí).

Zavedeme si zde i další ***značení***, které budeme používat i v následných kapitolách.

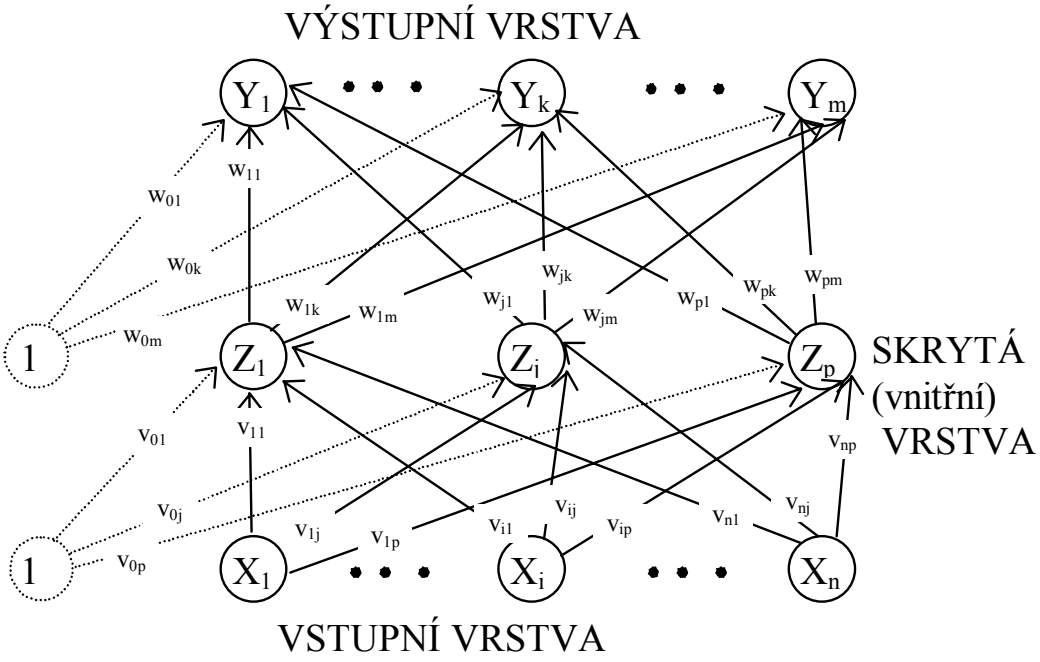
Klíčová slova této kapitoly:

backpropagation (adaptační algoritmus zpětného šíření chyby), generalizace, trénovací množina, dopředné (feedforward) šíření signálu.

V této kapitole budeme používat následující značení:

x	Vstupní vektor: **x** = (x_1, ..., x_i, ..., x_n).
t	Výstupní tréninkový vektor: **t** = (t_1, ..., t_k, ..., t_m).
δ_k	Částečné váhové korekce pro w_{jk} příslušející chybě na spojeních vedoucích k neuronu Y_k ve výstupní vrstvě.
δ_j	Částečné váhové korekce pro v_{ij} příslušející chybě na spojeních vedoucích k neuronu Z_j ve skryté vrstvě.
α	Koeficient učení.
X_i	*i.* neuron ve vstupní vrstvě: Pro neurony ve vstupní vrstvě je hodnota vstupního i výstupního signálu stejná, x_i .
v_{0j}	Bias *j.* neuronu ve skryté vrstvě.
Z_j	*j.* neuron ve skryté vrstvě: Hodnota vstupního signálu pro Z_j je z_in_j: $z\_in_j = v_{0j} + \sum_i x_i v_{ij}$. Hodnota vstupního signálu pro Z_j je z_j: $z_j = f(z\_in_j)$.
w_{0k}	Bias *k.* neuronu ve výstupní vrstvě.
Y_k	*k.* neuron ve výstupní vrstvě: Hodnota vstupního signálu pro Y_k je y_in_k: $y\_in_k = w_{0k} + \sum_j z_j w_{jk}$. Hodnota vstupního signálu pro Z_j je z_j: $y_k = f(y\_in_k)$.

Pravděpodobně nejrozšířenější způsob propojení neuronů se sigmoidní aktivační funkcí jsou vícevrstvé sítě. Vícevrstvá neuronová síť s jednou vnitřní vrstvou neuronů (neurony jsou označeny Z_j, $j = 1,..., p$) je zobrazena na obrázku 21. Výstupní neurony (neurony jsou označeny Y_k, $k = 1,..., m$). Neurony ve výstupní a vnitřní vrstvě musí mít definovaný bias. Typické označení pro bias *k.* neuronu (Y_k) ve výstupní vrstvě je w_{0k}, a typické označení pro bias *j.* neuronu (Z_j) ve vnitřní vrstvě je v_{0j}. Bias (např. *j.* neuronu) odpovídá, jak již bylo dříve uvedeno, váhové hodnotě přiřazené spojení mezi daným neuronem a fiktivním neuronem, jehož aktivace je vždy *1*. Z uvedeného obrázku tedy vyplývá, že vícevrstvá neuronová síť je tvořena minimálně třemi vrstvami neuronů: vstupní, výstupní a alespoň jednou vnitřní vrstvou. Vždy mezi dvěmi sousedními vrstvami se pak nachází tzv. *úplné propojení neuronů*, tedy každý neuron nižší vrstvy je spojen se všemi neurony vrstvy vyšší.



Obrázek 21: ***Neuronová síť s jednou vnitřní vrstvou neuronů.***

Adaptační algoritmus zpětného šíření chyby (*backpropagation*) je používán v přibližně 80% všech aplikací neuronových sítí. Samotný algoritmus obsahuje tři etapy: dopředné (*feedforward*) šíření vstupního signálu tréninkového vzoru, zpětné šíření chyby a aktualizace váhových hodnot na spojeních.

Během dopředného šíření signálu obdrží každý neuron ve vstupní vrstvě (X_i, $i = 1,..., n$) vstupní signál (x_i) a zprostředkuje jeho přenos ke všem neuronům vnitřní vrstvy ($Z_1, ..., Z_p$). Každý neuron ve vnitřní vrstvě vypočítá svou aktivaci (z_j) a pošle tento signál všem neuronům ve výstupní vrstvě. Každý neuron ve výstupní vrstvě vypočítá svou aktivaci (y_k), která odpovídá jeho skutečnému výstupu (*k.* neuronu) po předložení vstupního vzoru.

V podstatě tímto způsobem získáme odezvu neuronové sítě na vstupní podnět daný excitací neuronů vstupní vrstvy. Takovým způsobem probíhá šíření signálů i v biologickém systému, kde vstupní vrstva může být tvořena např. zrakovými buňkami a ve výstupní vrstvě mozku jsou pak identifikovány jednotlivé objekty sledování. Otázkou pak zůstává to nejdůležitější, jakým způsobem jsou stanoveny synaptické váhy vedoucí ke korektní odezvě na vstupní signál. Proces stanovení synaptických vah je opět spjat s pojmem učení - adaptace - neuronové sítě.

Další otázkou je schopnost *generalizace* (zobecnění*)* nad naučeným materiálem, jinými slovy jak je neuronová síť schopna na základě naučeného usuzovat na jevy, které nebyly součástí učení, které však lze nějakým způsobem z naučeného odvodit. I tady je cítit jakási analogie s lidským učením daná rozdílem mezi bezduchým biflováním a učením spjatým se schopností porozumět problematice tak, aby mohlo být nové odvozeno z předchozího.

Co je nutné k naučení neuronové sítě? Je to jednak tzv. *trénovací množina* obsahující prvky popisující řešenou problematiku a dále pak metoda, která dokáže tyto vzorky zafixovat v neuronové síti formou hodnot

synaptických vah pokud možno včetně již uvedené schopnosti generalizovat. Zastavme se nejdříve u trénovací množiny. Každý vzor trénovací množiny popisuje jakým způsobem jsou excitovány neurony vstupní a výstupní vrstvy.

Formálně můžeme za trénovací množinu ***T*** považovat množinu prvků (vzorů), které jsou definovány uspořádanými dvojicemi následujícím způsobem:

$$T = \left\{ \left\{ S_1, T_1 \right\} \quad \left\{ S_2, T_2 \right\} \quad \ldots \quad \left\{ S_q, T_q \right\} \right\}$$

$$S_i = \left[s_1 \quad s_2 \quad \ldots \quad s_n \right] \qquad s_j \in \langle 0,1 \rangle$$

$$T_i = \left[t_1 \quad t_2 \quad \ldots \quad t_m \right] \qquad t_j \in \langle 0,1 \rangle$$

kde q počet vzorů trénovací množiny
S_i vektor excitací vstupní vrstvy tvořené n neurony
T_i vektor excitací výstupní vrstvy tvořené m neurony
s_j, t_j excitace *j-tého* neuronu vstupní resp. výstupní vrstvy.

Metoda, která umožňuje adaptaci neuronové sítě nad danou trénovací množinou se nazývá *backpropagation*, což v překladu znamená metodu zpětného šíření. Na rozdíl od už popsaného dopředného chodu při šíření signálu neuronové sítě spočívá tato metoda adaptace v opačném šíření informace směrem od vrstev vyšších k vrstvám nižším.

Během adaptace neuronové sítě metodou backpropagation jsou srovnávány vypočítané aktivace y_k s definovanými výstupními hodnotami t_k pro každý neuron ve výstupní vrstvě a pro každý tréninkový vzor. Na základě tohoto srovnání je definována chyba neuronové sítě, pro kterou je vypočítán faktor δ_k ($k = 1, ..., m$). δ_k je, jak již bylo uvedeno, části chyby, která se šíří zpětně z neuronu Y_k ke všem neuronům předcházející vrstvy, jež mají s tímto neuronem definované spojení. Podobně lze definovat i faktor δ_j (j = *1, ..., p*), který je části chyby šířené zpětně z neuronu Z_j ke všem neuronům vstupní vrstvy, jež mají s tímto neuronem definované spojení.

Úprava váhových hodnot w_{jk} na spojeních mezi neurony vnitřní a výstupní vrstvy závisí na faktoru δ_k a aktivacích z_j neuronů Z_j ve vnitřní vrstvě. Úprava váhových hodnot v_{ij} na spojeních mezi neurony vstupní a vnitřní vrstvy závisí na faktoru δ_j a aktivacích x_i neuronů X_i ve vstupní vrstvě.

Aktivační funkce pro neuronové sítě s adaptační metodou backpropagation musí mít následující vlastnosti: musí být spojitá, diferencovatelná a monotónně neklesající. Nejčastěji používanou aktivační funkcí je proto standardní (logická) sigmoida a hyperbolický tangens.

Chyba sítě $E(\mathbf{w})$ je vzhledem k tréninkové množině definována jako součet parciálních chyb sítě $E_l(\mathbf{w})$ vzhledem k jednotlivým tréninkovým vzorům a závisí na konfiguraci sítě $\mathbf{w}$:

$$E(\mathbf{w}) = \sum_{l=1}^{q} E_l(\mathbf{w}).$$

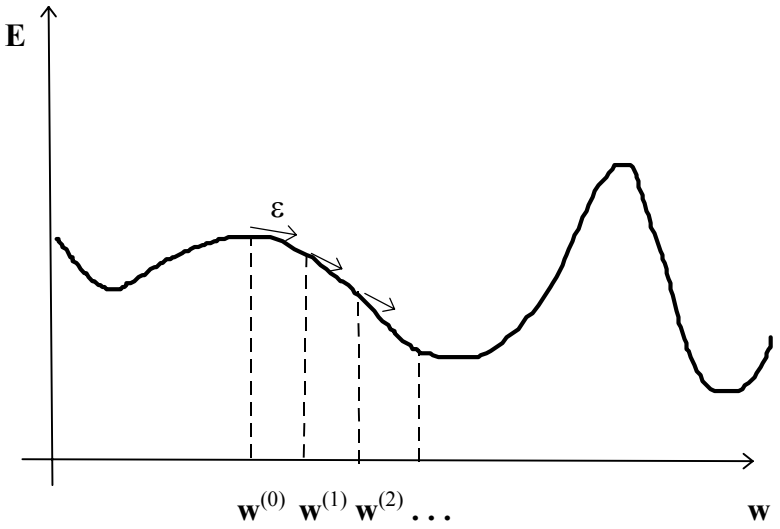
Parciální chyba $E_l(\mathbf{w})$ sítě pro *l.* tréninkový vzor (l = 1, ...,q) je úměrná součtu mocnin odchylek skutečných hodnot výstupu sítě pro vstup *l*-tréninkového vzoru od požadovaných hodnot výstupů u tohoto vzoru:

$$E_l(\mathbf{w}) = \frac{1}{2} \sum_{k \in Y} \left(y_k - t_k \right)^2 .$$

Cílem adaptace je minimalizace chyby sítě ve váhovém prostoru. Vzhledem k tomu, že chyba sítě přímo závisí na komplikované nelineární složené funkci vícevrstvé sítě, představuje tento cíl netriviální optimalizační problém. Pro jeho řešení se v základním modelu používá nejjednodušší varianta gradientní metody, která vyžaduje diferencovatelnost chybové funkce. K lepšímu pochopení nám pomůže geometrická představa.

Na obrázku 22 je schematicky znázorněna chybová funkce $E(\mathbf{w})$ tak, že konfigurace, která představuje mnohorozměrný vektor vah $\mathbf{w}$, se promítá na osu x. Chybová funkce určuje chybu sítě vzhledem k pevné tréninkové množině v závislosti na konfiguraci sítě. Při adaptaci sítě hledáme takovou konfiguraci, pro kterou je chybová funkce minimální. Začneme s náhodně zvolenou konfigurací $\mathbf{w}^{(0)}$, kdy odpovídající chyba sítě od požadované funkce bude pravděpodobně velká. V analogii s lidským učením to odpovídá počátečnímu nastavení

synaptických vah u novorozence, který místo požadovaného chování jako chůze, řeč apod. provádí náhodné pohyby a vydává neurčité zvuky. Při adaptaci sestrojíme v tomto bodě $\mathbf{w}^{(0)}$ ke grafu chybové funkce tečný vektor (*gradient*) $\frac{\partial E}{\partial \mathbf{w}}\left(\mathbf{w}^{(0)}\right)$ a posuneme se ve směru tohoto vektoru dolů o ε. Pro dostatečně malé ε tak získáme novou konfiguraci $\mathbf{w}^{(1)} = \mathbf{w}^{(0)} + \Delta\mathbf{w}^{(1)}$, pro kterou je chybová funkce menší než pro původní konfiguraci $\mathbf{w}^{(0)}$, tj. $\boldsymbol{E}(\mathbf{w}^{(0)}) \geq \boldsymbol{E}(\mathbf{w}^{(1)})$. Celý proces konstrukce tečného vektoru opakujeme pro $\mathbf{w}^{(1)}$ a získáme tak $\mathbf{w}^{(2)}$ takové, že $\boldsymbol{E}(\mathbf{w}^{(1)}) \geq \boldsymbol{E}(\mathbf{w}^{(2)})$ atd., až se limitně dostaneme do lokálního minima chybové funkce. Ve vícerozměrném váhovém prostoru tento postup přesahuje naši představivost. I když při vhodné volbě koeficientu učení (α) tato metoda vždy konverguje k nějakému lokálnímu minimu z libovolné počáteční konfigurace, není vůbec zaručeno, že se tak stane v reálném čase. Obvykle je tento proces časově velmi náročný (několik dnů výpočtu PC) i pro malé topologie vícevrstvé sítě (desítky neuronů).



Obrázek 22: ***Gradientní metoda.***

Hlavním problémem gradientní metody je, že pokud již nalezne lokální minimum, pak toto minimum nemusí být globální (viz obr.22). Uvedený postup adaptace se v takovém minimu zastaví (nulový gradient) a chyba sítě se již dále nesnižuje. To lze v naší analogii s učením člověka interpretovat tak, že počáteční nastavení konfigurace v okolí nějakého minima chybové funkce určuje možnosti jedince učit se. Inteligentnější lidé začínají svou adaptaci v blízkosti hlubších minim. I zde je však chybová funkce definovaná relativně vzhledem k požadovanému „inteligentnímu“ chování (tréninková množina), které však nemusí být univerzálně platné. Hodnotu člověka nelze měřit žádnou chybovou funkcí. Elektrické šoky aplikované v psychiatrických léčebnách připomínají některé metody adaptace neuronových sítí, které v případě, že se učení zastavilo v mělkém lokálním minimu chybové funkce, náhodně vnáší šum do konfigurace sítě, aby se síť dostala z oblasti abstrakce tohoto lokálního minima a mohla popř. konvergovat k hlubšímu minimu.

Popis algoritmu backpropagation

Krok 0. Váhové hodnoty a bias jsou inicializovány malými náhodnými čísly.

Přiřazení inicializační hodnoty koeficientu učení α.

Krok 1. Dokud není splněna podmínka ukončení výpočtu, opakovat kroky (2 až 9).

Krok 2. Pro každý (bipolární) tréninkový pár **s:t** provádět kroky (3 až 8).

Feedforward:

Krok 3. Aktivovat vstupní neurony (X_i, *i=1, ...n)*

$x_i = s_i$..

Krok 4 Vypočítat vstupní hodnoty vnitřních neuronů: (Z_j, *j=1,...,*

p):

$$z\_in_j = v_{0j} + \sum_{i=1}^{n} x_i v_{ij}.$$

Stanovení výstupních hodnot vnitřních neuronů

$$z_j = f\left(z\_in_j\right).$$

Krok 5 Stanovení skutečných výstupních hodnoty signálu neuronové sítě (Y_k, *k=1, ..., m)*:

$$y\_in_k = w_{0k} + \sum_{j=1}^{p} z_j\, w_{jk},$$

$$y_k = f\left(y\_in_k\right).$$

Backpropagation:

Krok 6 Ke každému neuronu ve výstupní vrstvě (Y_k, *k=1, ..., m)* je přiřazena hodnota očekávaného výstupu pro vstupní tréninkový vzor. Dále je vypočteno $\delta_k = \left(t_k - y_k\right) f'\left(y\_in_k\right)$, které je součástí váhové korekce $\Delta w_{jk} = \alpha\, \delta_k z_j$ i korekce biasu $\Delta w_{0k} = \alpha\, \delta_k$.

Krok 7 Ke každému neuronu ve vnitřní vrstvě (*Zj, j=1, ..., p)* je přiřazena sumace jeho delta vstupů (tj. z neuronů, které se nacházejí v následující vrstvě), $\delta\_in_j = \sum_{k=1}^{m} \delta_k w_{jk}$.

Vynásobením získaných hodnot derivací jejich aktivační funkce obdržíme $\delta_j = \delta\_in_j f'\left(z\_in_j\right)$, které je součástí váhové korekce $\Delta v_{ij} = \alpha\, \delta_j x_i$ i korekce biasu $\Delta v_{0j} = \alpha\, \delta_j$.

Aktualizace vah a prahů:

Krok 8 Každý neuron ve výstupní vrstvě (Y_k, *k=1, ..., m)* aktualizuje na svých spojeních váhové hodnoty včetně svého biasu (*j=0, ..., p)*:

$$w_{jk}\left(new\right) = w_{jk}\left(old\right) + \Delta w_{jk}.$$

Každý neuron ve vnitřní vrstvě (Z_j, *j=1, ..., p)* aktualizuje na svých spojeních váhové hodnoty včetně svého biasu (*i=0, ..., n)*:

$$v_{ij}\left(new\right) = v_{ij}\left(old\right) + \Delta v_{ij}.$$

Krok 9. Podmínka ukončení:
pokud již nenastávají žádné změny váhových hodnot nebo pokud již bylo vykonáno maximálně definované množství váhových změn, stop; jinak, pokračovat.

Ačkoliv vlastní popis učícího algoritmu backpropagation je formulován pro klasický von neumannovský model počítače, přesto je zřejmé, že jej lze implementovat distribuovaně. Pro každý tréninkový vzor probíhá nejprve aktivní režim pro jeho vstup tak, že informace se v neuronové síti šíří od vstupu k jejímu výstupu. Potom na základě externí informace učitele o požadovaném výstupu, tj. o chybě u jednotlivých vstupů, se počítají parciální derivace chybové funkce tak, že signál se šíří zpět od výstupu ke vstupu. Výpočet sítě při zpětném chodu probíhá sekvenčně po vrstvách, přitom v rámci jedné vrstvy může probíhat paralelně.

Odvození adaptačního pravidla standardní backpropagation

Symbolem w_{JK} označíme váhovou hodnotu na spojení mezi vnitřním neuronem Z_J a neuronem ve výstupní vrstvě Y_K; indexy IJ jsou použity analogicky pro váhové spojení mezi neuronem ve vstupní vrstvě X_I a vnitřním neuronem Z_J. Indexy uvedené malými písmeny se vyskytují pouze v sumacích. Symbolem *f(x)* označujeme aktivační funkci libovolného typu. Derivace této aktivační funkce je pak označena symbolem *f'*. Závislost aktivační funkce na váhových hodnotách je vyjádřena vztahem:

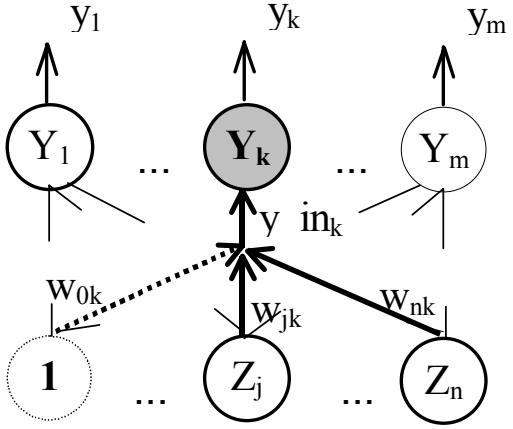
$$y\_in_K = \sum_j z_j w_{jK} ,$$

který musíme vyčíslit, abychom nalezli $f\left(y\_in_K\right)$, tj. aktivační hodnotu Y_K (*K*. neuronu ve výstupní vrstvě).

Chybovou funkci (tj. funkci váhových hodnot), která má být minimalizována, lze zapsat takto:

$$E = 0.5\sum_k \left[t_k - y_k\right]^2 .$$

Dále následuje odvození váhového přírůstku nejprve pro spojení mezi neurony vnitřní a výstupní vrstvy, tj. Δw_{JK} a potom mezi neurony ve vstupní a vnitřními vrstvě, tj. Δv_{IJ}.



Obrázek 23: Adaptace vah neuronu výstupní vrstvy.

$$\frac{\partial E}{\partial w_{J\,K}} = \frac{\partial}{\partial w_{J\,K}} 0.5 \sum_k \left[t_k - y_k \right]^2$$

$$= \frac{\partial}{\partial w_{J\,K}} 0.5 \sum_k \left[t_k - f\left(y\_in_K \right) \right]^2$$

$$= -\left[t_K - y_K \right] \frac{\partial}{\partial w_{J\,K}} f\left(y\_in_K \right)$$

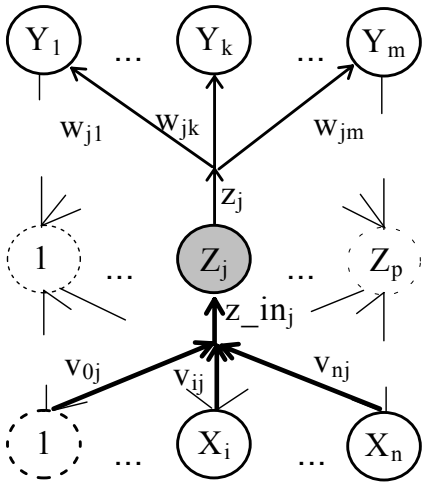
$$= -\left[t_K - y_K \right] f'\left(y\_in_K \right) \frac{\partial}{\partial w_{J\,K}} f\left(y\_in_K \right)$$

$$= -\left[t_K - y_K \right] f'\left(y\_in_K \right) z_J .$$

Pro přehlednější zápis výsledných hodnot je výhodné definovat δ_K :

$$\delta_K = \left[t_K - y_K \right] f'\left(y\_in_K \right).$$

Pro váhové hodnoty na spojeních vedoucích od neuronů vstupní vrstvy k neuronům ve vnitřní vrstvě platí:



Obrázek 24: ***Adaptace vah neuronu vnitřní vrstvy.***

$$\frac{\partial E}{\partial v_{I\,J}} = -\sum_k \left[t_k - y_k \right] \frac{\partial}{\partial v_{I\,J}} y_k$$

$$= -\sum_k \left[t_k - y_k \right] f'\left(y\_in_k \right) \frac{\partial}{\partial v_{I\,J}} y\_in_k$$

$$= -\sum_k \delta_k \frac{\partial}{\partial v_{I\,J}} y\_in_k$$

$$= -\sum_k \delta_k w_{J\,k} \frac{\partial}{\partial v_{I\,J}} z_J$$

$$= -\sum_k \delta_k w_{J\,k} f'\left(z\_in_J \right) \left[x_I \right].$$

I zde pro přehlednost následujícího zápisu definujme

$$\delta_J = -\sum_k \delta_k w_{J\,k} f'\left(z\_in_J\right).$$

Vrátíme se opět k indexaci malými písmeny. Váhové přírůstky pak lze zapsat následujícími způsoby: pro váhové hodnoty na spojeních mezi neurony ve vnitřní a výstupní vrstvě platí

$$\begin{aligned}\Delta w_{j\,k} &= -\alpha \frac{\partial E}{\partial w_{j\,k}} \\ &= \alpha\left[t_k - y_k\right] f'\left(y\_in_k\right) z_j \\ &= \alpha\delta_k z_j ;\end{aligned}$$

a pro váhové hodnoty na spojeních mezi neurony ve vstupní a vnitřní vrstvě platí

$$\begin{aligned}\Delta v_{i\,j} &= -\alpha \frac{\partial E}{\partial v_{i\,j}} \\ &= \alpha f'\left(z\_in_j\right) x_i \sum_k \delta_k w_{j\,k} , \\ &= \alpha\delta_j x_i .\end{aligned}$$

Uvedené vztahy vyjadřují podstatu adaptace neuronové sítě metodou backpropagation . Pokusme se o jejich bližší vysvětlení [7], tj. o přiblížení výrazu daného součinem koeficientu učení α a parciální derivace chyby *E* podle příslušné synaptické váhy. Pokud je hodnota této derivace velká a kladná, znamená to, že i minimální nárůst hodnoty synaptické váhy vede k velké chybě odezvy neuronové sítě. Je proto nutné "ubrat" z aktuální hodnoty synaptické váhy, abychom chybu zmenšili. Pro velkou, ale zápornou hodnotu derivace analogicky platí, že je naopak nutné hodnotu synaptické váhy zvětšit, pokud by měla být chyba odezvy v následujícím kroku nižší. Velikosti úprav synaptických dat jsou logicky dány nejen hodnotami těchto derivací, ale i koeficientem učení (α). Čím větší bude tento koeficient, tím razantnější budou změny v neuronové síti a naopak, pokud se bude jeho hodnota blížit nule pak změny budou jen velmi nepatrné. Na tomto místě se opět nabízí analogie s lidským chováním. V prvém případě se jedná o člověka, který s každou novou informací výrazně přebuduje své názory či znalosti. V druhém případě se jedná o člověka, který s každou novou informací vyžaduje dlouhé přesvědčování a působení, než akceptuje něco nového. Již z této analogie je patrné jak je tento koeficient důležitý pro efektivní adaptaci neuronové sítě, nicméně jeho stanovení je věc experimentu a hledání. Prakticky neexistuje exaktní pravidlo, které by tento problém mohlo vyřešit.

Úkoly:

Použijte adaptační algoritmus backpropagation pro logickou funkci „XOR“. Získané výsledky srovnejte s řešením téhož problému při použití adaptačního algoritmu MRI.

Korespondenční úkoly:

Vytvořte počítačový program pro realizaci adaptačního algoritmu backpropagation.

VARIANTY BACKPROPAGATION.

V této kapitole se seznámíte s možnými variantami adaptačního pravidla backpropagation, tj. do standardního algoritmu zavedeme parametr *momentu* a modifikovatelný parametr *strmosti*.

V závěru rozebereme problematiku vhodné volby topologie vícevrstvé neuronové sítě, která by měla odpovídat složitosti řešeného problému.

V celé kapitole budeme používat značení zavedené v kapitole „*Backpropagation*“.

Klíčová slova této kapitoly:

parametr momentu, parametr strmosti, heterogenní síť, overfitting (přeučení).

Popsaná standardní metoda backpropagation se vzhledem ke své jednoduchosti často používá, i když není příliš efektní. Její jednoduchá a celkem frekventovaná modifikace, která se snaží tento nedostatek částečně odstranit, zohledňuje při výpočtu nejen změny vah ve směru gradientu chybové funkce, ale navíc i předešlou změnu vah, tzv. *moment* (μ). Přírůstky váhových hodnot odvozené standardní metodou backpropagation pak můžeme přepsat do následujících tvarů:

$$w_{jk}(t+1) = w_{jk}(t) + \alpha\delta_k z_j + \mu\left[w_{jk}(t) - w_{jk}(t-1)\right],$$

nebo-li

$$\Delta w_{jk}(t+1) = \alpha\delta_k z_j + \mu\,\Delta w_{jk}(t)$$

a

$$v_{ij}(t+1) = v_{ij}(t) + \alpha\delta_j x_i + \mu\left[v_{ij}(t) - v_{ij}(t-1)\right],$$

nebo-li

$$\Delta v_{ij}(t+1) = \alpha\delta_j x_i + \mu\,\Delta v_{ij}(t),$$

kde $0 < \mu < 1$ je parametr momentu, který určuje míru vlivu předchozí změny (obvykle se volí $\mu = 0.9$). Pomocí momentu gradientní metoda lépe opisuje tvar chybové funkce $E(\boldsymbol{w})$, protože bere do úvahy předchozí gradient.

Doposud jsme se v našem výkladu zabývali pouze adaptací synaptických vah na spojeních mezi neurony, protože jsme pracovali pouze s neurony, které mají stejnou aktivační funkcí, přesněji: aktivační funkci se stejnou strmostí sigmoidu σ. Nicméně nic nebrání tomu, abychom adaptaci podrobili nejen synaptické váhy, ale i výše zmíněné strmosti sigmoidů jednotlivých neuronů. Konfigurace sítě je pak dána vektorem všech vah $\boldsymbol{w}$ a vektorem všech strmostí σ. Při učení adaptujeme tuto konfiguraci tak, že chybu sítě minimalizujeme gradientní metodou v prostoru vah a strmostí. Tím zvyšujeme stupeň volnosti adaptace, kdy tvar aktivační funkce (tj. míra rozhodnosti jednotlivých neuronů) se může přizpůsobit tréninkové množině a snáze nalezne globální minimum chybové funkce sítě. Na druhou stranu při zvýšení počtu adaptovaných parametrů roste počet numerických operací a učení se zpomaluje.

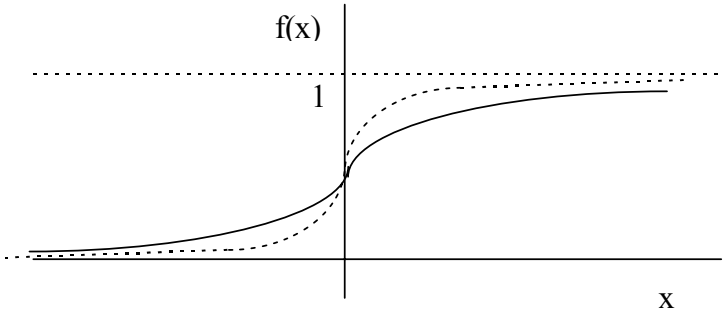
Sigmoidální aktivační funkce *standardní (logická) sigmoida,* je potom přepsána do následujícího tvaru:

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

Derivace této funkce je pak zapsána takto

$$f'(x) = \sigma\, f(x)\left[1 - f(x)\right]$$

a její graf je uveden na obrázku 25.



Obrázek 25: ***Binární sigmoida s modifikovanou strmostí:*** $\sigma = 1$ ***a*** $\sigma = 3$.

Tímto způsobem lze získat tzv. *heterogenní* síť, kde obecně každý neuron může mít svou aktivační dynamiku. Tato vlastnost ve většině případů zvyšuje schopnost sítě konvergovat k naučenému stavu. Adaptační metoda využívající této možnosti je popsána v následující kapitole.

Backpropagation s adaptivní strmostí sigmoidů

Odvození adaptačního pravidla backpropagation s adaptivní strmostí sigmoidu je velmi podobné odvození adaptačního pravidla standardní backpropagation. Budeme používat i stejnou indexaci. Rovněž i volba aktivační funkce *f(x)* je libovolná.
Uvažujme vstupní signál *x* pro neurony ve výstupní vrstvě, Y_K

$$x = \sigma_K y\_in_K$$

a stejným způsobem označený vstupní signál pro neurony ve vnitřní vrstvě, Z_J

$$x = \sigma_J z\_in_J .$$

Aktivační funkce potom závisí nejen na váhových hodnotách

$$y\_in_K = \sum_j z_j w_{jK} ,$$

ale i na hodnotě parametru σ_K, který je přiřazen každému neuronu. Stejné úvahy provedeme i pro neurony vnitřní vrstvy. I když má každý neuron přiřazen svůj parametr σ_K, tvar aktivační funkce je pro všechny tyto neurony identický. Za těchto předpokladů budeme odvozovat hodnoty váhových přírůstků parametrické backpropagation, tj. budeme minimalizovat i v tomto případě chybovou funkci, kterou zapíšeme následujícím způsobem:

$$E = 0.5\sum_k \left[t_k - y_k\right]^2 .$$

Odvození váhového přírůstku provedeme rovněž nejprve pro spojení mezi neurony vnitřní a výstupní vrstvy, tj. Δw_{JK} a potom mezi neurony ve vstupní a vnitřními vrstvě, tj. Δv_{IJ}.

$$\frac{\partial E}{\partial w_{J\,K}} = \frac{\partial}{\partial w_{J\,K}} 0.5 \sum_k \left[t_k - y_k\right]^2$$
$$= \frac{\partial}{\partial w_{J\,K}} 0.5 \sum_k \left[t_k - f\left(\sigma_K y\_in_K\right)\right]^2$$
$$= -\left[t_K - y_K\right] \frac{\partial}{\partial w_{J\,K}} f\left(\sigma_K y\_in_K\right)$$
$$= -\left[t_K - y_K\right] f'\left(\sigma_K y\_in_K\right) \frac{\partial}{\partial w_{J\,K}} f\left(\sigma_K y\_in_K\right)$$
$$= -\left[t_K - y_K\right] f'\left(\sigma_K y\_in_K\right) \sigma_K z_J .$$

Stejně jako váhové hodnoty, musíme také pro každý neuron adaptovat hodnoty parametru σ_K. Toto odvození rovněž provedeme nejprve neurony výstupní vrstvy, tj. $\Delta\sigma_k$ a potom pro neurony ve vnitřními vrstvě, tj. $\Delta\sigma_j$.

$$\frac{\partial E}{\partial \sigma_K} = -\left[t_K - y_K\right] f'\left(\sigma_K y\_in_K\right) \frac{\partial}{\partial_K} \left(\sigma_K y\_in_K\right)$$
$$= -\left[t_K - y_K\right] f'\left(\sigma_K y\_in_K\right) y\_in_K .$$

Pro přehlednost i zde použijeme následujícího zápisu:

$$\delta_K = \left[t_K - y_K\right] f'\left(\sigma_K y\_in_K\right).$$

Pro váhové hodnoty na spojeních vedoucích od neuronů vstupní vrstvy k neuronům ve vnitřní vrstvě platí:

$$\frac{\partial E}{\partial v_{I\,J}} = -\sum_k \left[t_k - y_k\right] \frac{\partial}{\partial v_{I\,J}} y_k$$
$$= -\sum_k \left[t_k - y_k\right] f'\left(\sigma_k y\_in_k\right) \frac{\partial}{\partial v_{I\,J}} \sigma_k y\_in_k$$
$$= -\sum_k \delta_k \sigma_k \frac{\partial}{\partial v_{I\,J}} y\_in_k$$
$$= -\sum_k \delta_k \sigma_k w_{J\,k} \frac{\partial}{\partial v_{I\,J}} z_J$$
$$= -\sum_k \delta_k \sigma_k w_{J\,k} f'\left(\sigma_J z\_in_J\right) \sigma_J \left[x_I\right].$$

a pro parametry σ_j vnitřních neuronů platí:

$$\begin{aligned}\frac{\partial E}{\partial \sigma_J} &= -\sum_k \left[t_k - y_k\right] \frac{\partial}{\partial \sigma_J} y_k \\ &= -\sum_k \left[t_k - y_k\right] f'\left(\sigma_k y\_in_k\right) \frac{\partial}{\partial \sigma_J} \sigma_k y\_in_k \\ &= -\sum_k \delta_k \sigma_k \frac{\partial}{\partial \sigma_J} y\_in_k \\ &= -\sum_k \delta_k \sigma_k w_{Jk} \frac{\partial}{\partial \sigma_J} z_J \\ &= -\sum_k \delta_k \sigma_k w_{Jk} f'\left(\sigma_J z\_in_J\right) z\_in_J .\end{aligned}$$

Také zde pro lepší přehlednost dalších zápisům definujme

$$\delta_J = -\sum_k \delta_k \sigma_k w_{Jk} f'\left(\sigma_J z\_in_J\right).$$

Nyní se vrátíme opět k indexaci malými písmeny a váhové přírůstky resp. přírůstky parametru strmosti sigmoidu pak zapíšeme následujícími způsoby:
pro váhové hodnoty na spojeních mezi neurony vnitřní a výstupní vrstvy platí

$$\begin{aligned}\Delta w_{jk} &= -\alpha \frac{\partial E}{\partial w_{jk}} \\ &= \alpha\left[t_k - y_k\right] f'\left(\sigma_k y\_in_k\right) \sigma_k z_j \\ &= \alpha \delta_k \sigma_k z_j ;\end{aligned}$$

pro váhové hodnoty na spojeních mezi neurony vstupní a vnitřní vrstvy platí

$$\begin{aligned}\Delta v_{ij} &= -\alpha \frac{\partial E}{\partial v_{ij}} \\ &= \alpha \sigma_j f'\left(\sigma_j z\_in_j\right) x_i \sum_k \delta_k \sigma_k w_{jk}, \\ &= \alpha \delta_j \sigma_j x_i ;\end{aligned}$$

resp. pro strmosti sigmoidu neuronů výstupní vrstvy platí

$$\begin{aligned}\Delta \sigma_k &= -\alpha \frac{\partial E}{\partial \sigma_k} \\ &= \alpha\left[t_k - y_k\right] f'\left(\sigma_k y\_in_k\right) y\_in_k \\ &= \alpha \delta_k y\_in_k ;\end{aligned}$$

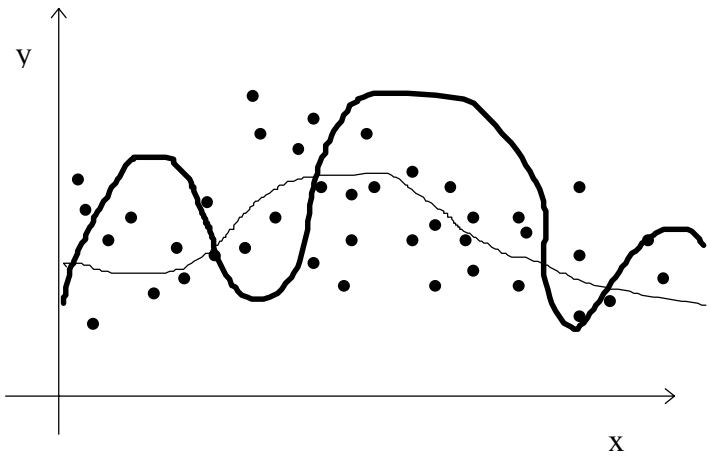
a pro strmosti sigmoidu neuronů vnitřní vrstvy platí

$$\Delta\sigma_j = -\alpha \frac{\partial E}{\partial \sigma_j}$$
$$= -\alpha \sum_k \delta_k \sigma_k w_{jk} f'\left(\sigma_j z\_in_j\right) z\_in_j,$$
$$= \alpha \delta_j z\_in_j.$$

Volba topologie vícevrstvé neuronové sítě [6]

Velkým problémem modelu vícevrstvé neuronové sítě s adaptačním algoritmem backpropagation je (kromě minimalizace chybové funkce) volba vhodné topologie pro řešení konkrétního praktického problému. Zřídkakdy jsou podrobněji známy vztahy mezi vstupy a výstupy, které by se daly využít při návrhu speciální architektury. Většinou se používá vícevrstvá topologie s jednou nebo dvěmi vnitřními vrstvami a očekává se, že učící algoritmus backpropagation zobecní příslušné vztahy z tréninkové množiny ve vahách jednotlivých spojů mezi neurony. I v tomto případě je však potřeba vhodně volit počty neuronů ve vnitřních vrstvách. Je zřejmé, že tento problém organizační dynamiky úzce souvisí s adaptací a generalizací neuronové sítě.

Architektura vícevrstvé neuronové sítě (tj. určení vhodného počtu vnitřních neuronů a jejich spojení), by měla odpovídat složitosti řešeného problému, tj. počtu tréninkových vzorů, jejich vstupů a výstupů a struktuře vztahů, které popisují. Je zřejmé, že malá síť nemůže řešit komplikovaný problém. Při učení pomocí algoritmu backpropagation se příliš malá síť obvykle zastaví v nějakém mělkém lokálním minimu a je potřeba topologii doplnit o další vnitřní neurony, aby adaptace měla větší stupeň volnosti. Na druhou stranu bohatá architektura sice při učení mnohdy umožní nalézt globální minimum chybové funkce, i když s větším počtem vah roste výpočetní náročnost adaptace. Avšak nalezená konfigurace sítě obvykle příliš zobecňuje tréninkové vzory včetně jejich nepřesností a chyb a pro nenaučené vzory dává chybné výsledky, tj. špatně generalizuje. Tomuto přesnému zapamatování tréninkové množiny bez zobecnění zákonitostí v ní obsažených se říká *přeučení* (*overfitting*). Na obrázku 26 jsou graficky znázorněny dvě funkce sítě spolu s tréninkovými vzory (body), ze kterých byly naučeny. Silná čára představuje přeučenou síť, jejíž funkce se přizpůsobila nepřesným tréninkovým vzorům, zatímco tenká čára představuje funkci sítě, která „správně“ generalizovala zákonitosti v tréninkové množině. Zdá se tedy, že existuje optimální topologie, která je na jednu stranu dostatečně bohatá, aby byla schopna řešit daný problém, a na druhou stranu ne moc velká, aby správně zobecnila potřebné vztahy mezi vstupy a výstupy.



Obrázek 26: ***Graf funkce přeučené sítě (tučně) se „správnou“ generalizací.***

Existují teoretické výsledky ohledně horního odhadu počtu vnitřních neuronů postačujících pro realizaci libovolné funkce z určité třídy, avšak pro praktické potřeby jsou příliš nadhodnocené, a tedy nepoužitelné. V praxi se obvykle topologie volí heuristicky, např. v první vnitřní vrstvě o něco více neuronů, než je vstupů a v druhé vrstvě aritmetický průmět mezi počtem výstupů a neuronů v první vnitřní vrstvě. Po adaptaci se v případě velké chyby sítě případně přidá, respektive při chudé generalizaci odebere několik neuronů a adaptivní režim se

celý opakuje pro novou architekturu. Pro test kvality generalizace neuronové sítě se počítá chyba sítě vzhledem k tzv. *testovací množině*, což je část tréninkové množiny, která se záměrně nevyužila k adaptaci.

Úkoly:

1. Řešte logickou funkci „XOR" standardním adaptačním algoritmem zpětného šíření chyby i algoritmem backpropagation s adaptivní strmostí sigmoidů. Oba výsledky řešení porovnejte.

2. Řešte vybranou logickou funkci standardním adaptačním algoritmem zpětného šíření chyby při stanovení různého počtu neuronů ve vnitřní vrstvě. Získané výsledky řešení srovnejte.

S A M O O R G A N I Z A C E .

V této kapitole se budeme věnovat modelům neuronových sítí, které využívají *soutěžní* strategie učení (*competitive learning*). Společným principem těchto modelů je, že výstupní neurony sítě spolu soutěží o to, který z nich bude aktivní. Na rozdíl od jiných učících principů (např. Hebbovo učení) je tedy v určitém čase aktivní vždy jen jeden neuron.

Klíčová slova této kapitoly:

adaptace bez učitele, *samoorganizace, soutěžní strategie učení (competitive learning), laterální inhibice, sousedství, proces shlukování, kvantování vektorů učením (LVQ).*

Kohonenovy samoorganizační mapy

Tato neuronová síť (angl. *Self-Organizing Map*) byla poprvé popsána v roce 1982. Je nejdůležitější architekturou vycházející ze strategie soutěžního učení (tj. *učení bez učitele*). Základním principem učícího procesu je vytvoření množiny reprezentantů mající stejné pravděpodobnosti výběru. Přesněji, hledáme takové reprezentanty, pro které platí: vybereme-li náhodný vstupní vektor z rozdělení pravděpodobnosti odpovídající rozdělení tréninkové množiny, bude mít každý takový reprezentant přiřazenu pravděpodobnost, která je mu nejblíže. Algoritmus tedy nemá informace o požadovaných aktivitách výstupních neuronů v průběhu adaptace, ale adaptace vah odráží statistické vlastnosti trénovací množiny. Jsou-li si tedy dva libovolné vzory blízké ve vstupním prostoru způsobují v síti odezvu na neuronech, které jsou si fyzicky blízké ve výstupním prostoru. Hlavní ideou těchto neuronových sítí je nalézt prostorovou reprezentaci složitých datových struktur. Mnohodimenzionální data se tímto způsobem zobrazují v daleko jednodušším prostoru. Uvedená vlastnost je typická i pro skutečný mozek, kde například jeden konec sluchové části mozkové kůry reaguje na nízké frekvence, zatímco opačný konec reaguje na frekvence vysoké.

Organizační dynamika sítě:

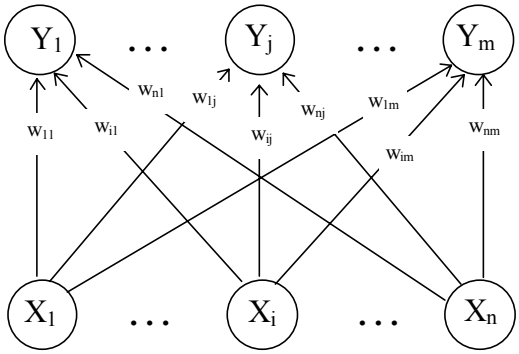
Jedná se o dvouvrstvou síť s úplným propojením neuronů mezi vrstvami. Výstupní neurony jsou navíc uspořádány do nějaké topologické struktury, nejčastěji to bývá dvojrozměrná mřížka nebo jednorozměrná řada jednotek. Tato topologická struktura určuje, které neurony spolu v síti sousedí (pro adaptační proces je to nezbytné). Pro adaptační proces je rovněž důležité zavést pojem *okolí* J výstupního neuronu (j^*) o *poloměru* (velikosti) R, což je množina všech neuronů ($j \in J$), jejichž vzdálenost v síti je od daného neuronu (j^*) menší nebo rovna R:

$$J = \{j;\ d(j,j^*) \leq R\}.$$

To, jak měříme vzdálenost $d(j,j^*)$, je závislé na topologické struktuře výstupních neuronů. Např. pro lineární oblast obsahující m neuronů ve výstupní vrstvě platí pro všechny $j \in J$:

$$\max(1, J - R) \leq j \leq \min(J+R, m).$$

Obecná architektura Kohonenovy samoorganizační mapy obsahující *m* neuronů ve výstupní vrstvě (tj. $Y_1,..., Y_m$) a *n* neuronů ve vstupní vrstvě (tj. $X_1,..., X_n$) je zobrazena na obrázku 27.



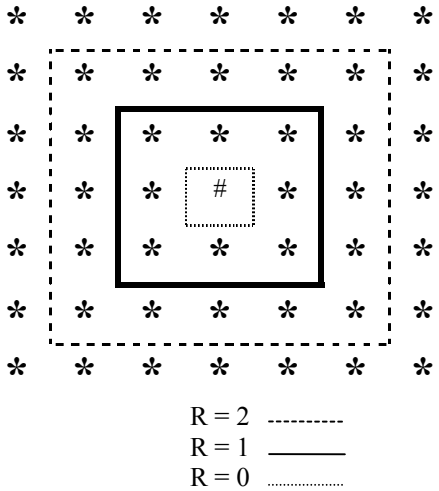
Obrázek 27: ***Kohonenova samoorganizační mapa.***

Sousedství neuronu označeného # je pro ***R =2*** **{ }**, ***R =1*** **( )**, ***R = 0*** **[ ]** v jednorozměrné výstupní oblasti zobrazeno na obrázku 28 (*m = 10* je počet neuronů ve výstupní vrstvě)

* * { * (* [#] * *) * } * *

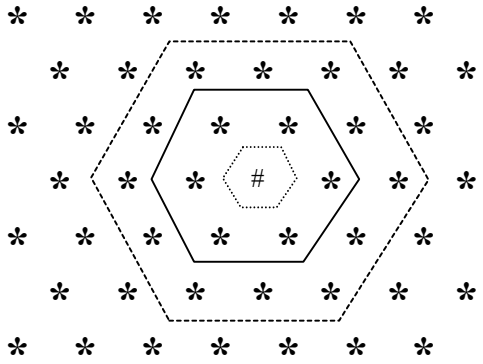
Obrázek 28: ***Sousedství definovaná v lineární výstupní oblasti pro různé hodnoty parametru R.***

Sousedství neuronu označeného # je pro ***R =2, 1, 0*** ve dvourozměrné pravoúhlé výstupní oblasti zobrazeno na obrázku 29.



Obrázek 29: ***Sousedství v pravoúhlé dvojrozměrné oblasti.***

Sousedství neuronu označeného # je pro ***R =2, 1, 0*** ve dvourozměrné hexagonální výstupní oblasti zobrazeno na obrázku 30.



R = 2 ----------
R = 1 ———
R = 0 .....

Obrázek 30: ***Sousedství v hexagonální dvojrozměrné oblasti.***

Princip ***adaptivní dynamiky*** je jednoduchý: Procházíme celou tréninkovou množinu a po předložení jednoho tréninkového vzoru proběhne mezi neurony sítě kompetice. Její vítěz pak spolu s neurony, které jsou v jeho okolí, změní své váhové hodnoty. Reálný parametr učení $0 < \alpha \leq 1$ určuje míru změny vah. Na počátku učení je obvykle blízký jedné a postupně se zmenšuje až na nulovou hodnotu, což zabezpečuje ukončení procesu adaptace. Rovněž i velikost okolí R není konstantní: na začátku adaptace je okolí obvykle velké (např. polovina velikosti sítě) a na konci učení potom zahrnuje jen jeden samotný vítězný neuron (tj. $R = 0$).

Popis algoritmu

Krok 0. Inicializace všech váhových hodnot w_{ij}:

Inicializace poloměru sousedství; tj okolí (R).

Inicializace parametru učení (α).

Krok 1. Pokud není splněna podmínka ukončení, provádět kroky (2 až 8).

Krok 2. Pro každý vstupní vektor $\boldsymbol{x} = (x_1,...,x_n)$ opakovat kroky 3 až 5.

Krok 3. Pro každé j $(j = 1,..., m)$ vypočítat:

$$D(j) = \sum_i \left(w_{ij} - x_i \right)^2 .$$

Krok 4. Najít index J takový, že $D(J)$ je minimum.

Krok 5. Aktualizace váhových hodnot všech neuronů $(j \in J)$ tvořících topologické sousedství charakterizované indexem J, tj. pro všechna i $(i = 1,..., n)$ platí:

$$w_{ij}(new) = w_{ij}(old) + \alpha\left[x_i - w_{ij}(old) \right].$$

Krok 6. Aktualizace parametru učení.

Krok 7. Zmenšení poloměru R topologického sousedství.

Krok 8. Test podmínky ukončení.

Geometrický význam popsaného algoritmu je takový, že vítězný neuron i všichni jeho sousedé v síti, kteří by od něj neměli být příliš vzdáleni ani ve vstupním prostoru, posunou svůj váhový vektor o určitou

poměrnou vzdálenost směrem k aktuálnímu vstupu. Motivací tohoto přístupu je snaha, aby vítězný neuron, který nejlépe reprezentuje předložený vstup (je mu nejblíže), ještě více zlepšil svou relativní pozici vůči němu.

Problémem vzniklým při adaptaci může být nevhodná náhodná inicializace vah, která vede k blízkým počátečním neuronům ve výstupní vrstvě a tudíž pouze jeden z nich vyhrává kompetici zatímco ostatní zůstávají nevyužity. Jedna z možností jak je možné tuto situaci vyřešit, je princip založený na "svědomí" každého z neuronů tak, že v případě příliš častých vítězství jednoho z nich, je tento neuron z procesu soutěže na chvíli vyjmut, aby dostali šanci i ostatní neurony výstupní vrstvy.

V ***aktivním režimu*** se pak sousedství neuronů neprojevuje: předložíme-li síti vstupní vektor, soutěží výstupní neurony o to, kdo je mu nejblíže, a tento neuron se pak excituje na hodnotu rovnu jedné, zatímco výstupy ostatních neuronů jsou rovny nule. Každý neuron tak reprezentuje nějaký objekt, či třídu objektů ze vstupního prostoru: tj. pouze jeden neuron horní vrstvy, jehož potenciál $(\Sigma w.x)$ je maximální odpovídá vstupnímu vektoru $\boldsymbol{x}$. Tento neuron je navíc schopen rozpoznat celou třídu takových, podobných si vektorů

Princip „vítěz bere vše“ se realizuje tzv. *laterální inhibicí*; všechny výstupní neurony jsou navzájem propojeny laterálními vazbami, které mezi nimi přenášejí inhibiční signály. Každý výstupní neuron se pak snaží v kompetici zeslabit ostatní neurony silou úměrnou jeho potenciálu, který je tím větší, čím je neuron blíže vstupu. Výsledkem tedy je, že výstupní neuron s největším potenciálem utlumí ostatní výstupní neurony a sám zůstane aktivním.

Příklad:

Mějme 4 vektory: (1, 1, 0, 0); (0, 0, 0, 1); (1, 0, 0, ,0); (0, 0, 1, 1).
Maximální počet shluků je: $m = 2$.
Předpokládejme, že parametr učení je definován vztahy: $\alpha(0) = 0{,}6$;

$$\alpha(t+1) = 0{,}5\ \alpha(t).$$

Protože jsou k dispozici pouze dva shluky, okolí bodu J (*krok 4*) je nastaveno tak, že v každém kroku aktualizuje své váhové hodnoty pouze jeden neuron výstupní vrstvy, tj. $R = 0$.

Krok 0. Inicializace váhové matice:

$$\begin{bmatrix} 0.2 & 0.8 \\ 0.6 & 0.4 \\ 0.5 & 0.7 \\ 0.9 & 0.3 \end{bmatrix}.$$

Inicializace poloměru sousedství:

$R = 0$.

Inicializace parametru učení:

$\alpha(0) = 0.6$.

Krok 1. Adaptace:

Krok 2. Pro první vektor (1, 1, 0, 0) opakovat kroky 3-5.

Krok 3.

$$D(1) = (0.2-1)^2 + (0.6-1)^2 + (0.5-0)^2 + (0.9-0)^2 = 1.86;$$

$$D(2) = (0.8-1)^2 + (0.4-1)^2 + (0.7-0)^2 + (0.3-0)^2 = 0.98.$$

Krok 4. Vstupní vektor je blíže uzlu 2, tak $J = 2$.

Krok 5. Aktualizace váhových hodnot vítězného neuronu:

$$w_{i2}(new) = w_{i2}(old) + 0.6\left[x_i - w_{i2}(old)\right]$$
$$= 0.4 w_{i2}(old) + 0.6 x_i.$$

Aktualizace druhého sloupce váhové matice

$$\begin{bmatrix} 0.2 & 0.92 \\ 0.6 & 0.76 \\ 0.5 & 0.28 \\ 0.9 & 0.12 \end{bmatrix}.$$

Krok 2. Pro druhý vektor (*0, 0, 0, 1*) opakovat kroky 3-5.

Krok 3. $D(1) = (0.2-0)^2 + (0.6-0)^2 + (0.5-0)^2 + (0.9-1)^2 = 0.66;$

$D(2) = (0.92-0)^2 + (0.76-0)^2 + (0.28-0)^2 + (0.12-1)^2 = 2.2768.$

Krok 4. Vstupní vektor je blíže uzlu 1, tak *J = 1.*

Krok 5. Aktualizace prvního sloupce váhové matice

$$\begin{bmatrix} 0.08 & 0.92 \\ 0.24 & 0.76 \\ 0.20 & 0.28 \\ 0.96 & 0.12 \end{bmatrix}.$$

Krok 2. Pro třetí vektor (*1, 0, 0, 0*) opakovat kroky 3-5.

Krok 3. $D(1) = (0.08-1)^2 + (0.24-0)^2 + (0.2-0)^2 + (0.96-0)^2 = 1.8656;$

$D(2) = (0.92-1)^2 + (0.76-0)^2 + (0.28-0)^2 + (0.12-0)^2 = 0.6768.$

Krok 4. Vstupní vektor je blíže uzlu 2, tak *J = 2.*

Krok 5. Aktualizace druhého sloupce váhové matice

$$\begin{bmatrix} 0.08 & 0.968 \\ 0.24 & 0.304 \\ 0.20 & 0.112 \\ 0.96 & 0.048 \end{bmatrix}.$$

Krok 2. Pro čtvrtý vektor (*0, 0, 1, 1*) opakovat kroky 3-5.

Krok 3. $D(1) = (0.08-0)^2 + (0.24-0)^2 + (0.2-1)^2 + (0.96-1)^2 = 0.7056;$

$D(2) = (0.968-0)^2 + (0.304-0)^2 + (0.112-1)^2 + (0.048-1)^2 = 2.724.$

Krok 4. Vstupní vektor je blíže uzlu 1, tak *J = 1.*

Krok 5. Aktualizace prvního sloupce váhové matice

$$\begin{bmatrix} 0.032 & 0.968 \\ 0.096 & 0.304 \\ 0.680 & 0.112 \\ 0.984 & 0.048 \end{bmatrix}.$$

Krok 6. Zmenšení parametru učení:

$\alpha = 0.5\ (0.6) = 0.3.$

Aktualizace váhových hodnot vítězného neuronu *j* (*j = 1, 2*) ve druhém cyklu bude prováděna podle vztahu:

$$w_{ij}(new) = w_{ij}(old) + 0.3\left[x_i - w_{ij}(old)\right]$$
$$= 0.7w_{ij}(old) + 0.3x_i .$$

Váhová matice má po druhém tréninkovém cyklu tvar:

$$\begin{bmatrix} 0.016 & 0.980 \\ 0.047 & 0.360 \\ 0.630 & 0.055 \\ 0.999 & 0.024 \end{bmatrix}.$$

Parametr učení zmenšil svou hodnotu během 100 iterací (cyklů) z 0.6 na 0.01 a váhová matice nabývala během adaptací následujících hodnot:

Iterace 0: Váhová matice: $\begin{bmatrix} 0.2 & 0.8 \\ 0.6 & 0.4 \\ 0.5 & 0.7 \\ 0.9 & 0.3 \end{bmatrix}$.

Iterace 1: Váhová matice: $\begin{bmatrix} 0.032 & 0.970 \\ 0.096 & 0.300 \\ 0.680 & 0.110 \\ 0.980 & 0.048 \end{bmatrix}$.

Iterace 2: Váhová matice: $\begin{bmatrix} 0.0053 & 0.9900 \\ -0.1700 & 0.3000 \\ 0.7000 & 0.0200 \\ 1.0000 & 0.0086 \end{bmatrix}$.

Iterace 10: Váhová matice: $\begin{bmatrix} 1.5e-7 & 1.0000 \\ 4.6e-7 & 0.3700 \\ 0.6300 & 5.4e-7 \\ 1.0000 & 2.3e-7 \end{bmatrix}$.

Iterace 50: Váhová matice: $\begin{bmatrix} 1.9e-19 & 1.0000 \\ 5.7e-15 & 0.4700 \\ 0.5300 & 6.6e-15 \\ 1.0000 & 2.8e-15 \end{bmatrix}$.

Iterace 100: Váhová matice: $\begin{bmatrix} 6.7e-17 & 1.0000 \\ 2.0e-16 & 0.4900 \\ 0.5100 & 2.3e-16 \\ 1.0000 & 1.0e-16 \end{bmatrix}$.

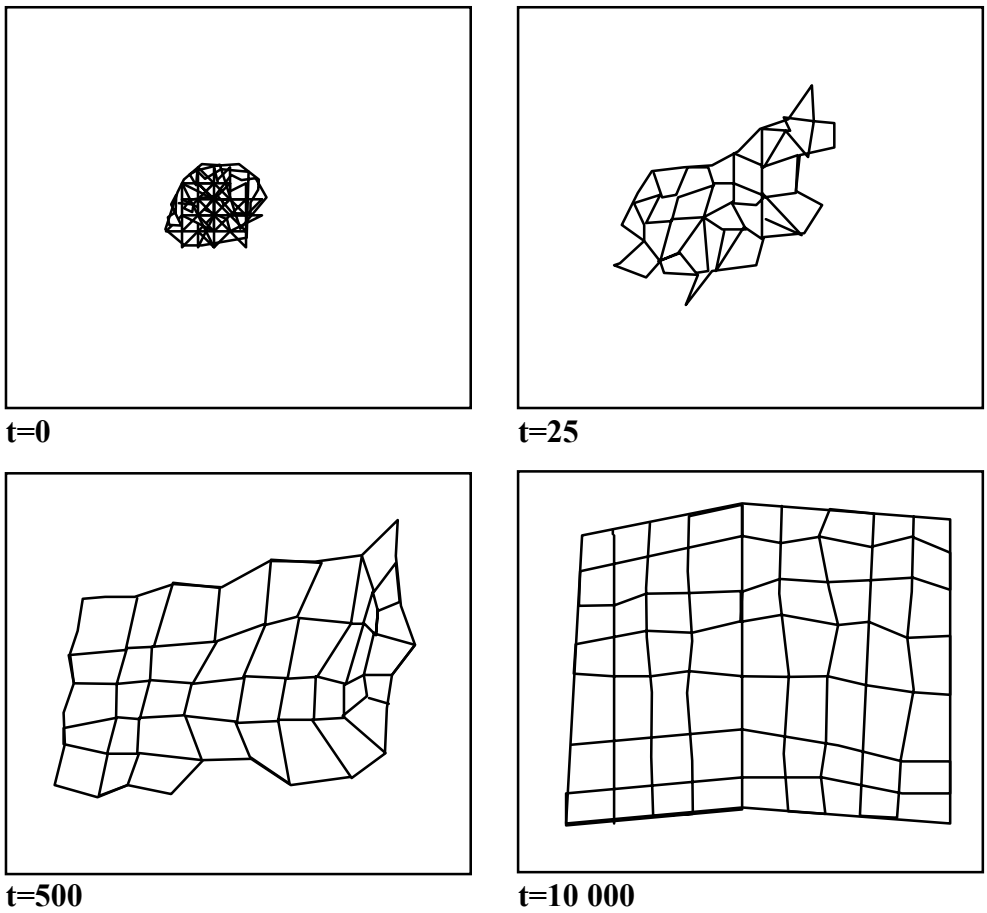
Tato matice konverguje k matici:

$$\begin{bmatrix} 0.0 & 1.0 \\ 0.0 & 0.5 \\ 0.5 & 0.0 \\ 1.0 & 0.0 \end{bmatrix}.$$

Její první sloupec nabývá hodnot, které odpovídají průměrným hodnotám složek obou vektorů přiřazeným prvnímu neuronu výstupní vrstvy (tj. vektoru 2: (0, 0, 0, 1) a vektoru 4: (0, 0, 1, 1)). Její druhý sloupec nabývá hodnot, které odpovídají průměrným hodnotám složek obou vektorů přiřazeným druhému neuronu výstupní vrstvy (tj. vektoru 1: (1, 1, 0, 0) a vektoru 3: (1, 0, 0, ,0)).

Proces shlukování ještě jednou vysvětlíme prostřednictvím funkce hustoty pravděpodobnosti. Tato funkce reprezentuje statistický nástroj popisující rozložení dat v prostoru. Pro daný bod prostoru lze tedy stanovit pravděpodobnost, že vektor bude v daném bodu nalezen. Je-li dán vstupní prostor a funkce hustoty pravděpodobnosti, pak je možné dosáhnout takové organizace mapy, která se této funkci přibližuje (za předpokladu, že je k dispozici reprezentativní vzorek dat). Jinými slovy řečeno, pokud jsou vzory ve vstupním prostoru rozloženy podle nějaké distribuční funkce, budou váhové vektory rozloženy analogicky.

Pokusme se výše uvedené demonstrovat na příkladu, kdy vstupní data jsou rovnoměrně rozložena v dvojdimenzionálním prostoru, konkrétně ve čtvercové oblasti. Váhové vektory budou tedy také dvojdimenzionální a budou zobrazovány formou bodu v prostoru vah. Dále budou v témže prostoru vykreslovány přímky spojující body (váhy) sousedících neuronů. Toto zobrazení pak vyjadřuje prostorové vztahy mezi neurony v prostoru vah. Vývoj prostorového uspořádání váhových vektorů lze demonstrovat na následujících diagramech.



O b r á z e k 3 1 : ***P r o c e s a d a p t a c e m a p y .***

Z obrázku 31 je patrné, že neurony byly optimálně rozloženy tak, aby pokryly vstupní datový prostor.

DP verze Kohonenova algoritmu

(**DP** angl. *Dot Product*); V základní verzi Kohonenova algoritmu (někdy označované **ED** angl. *Euclidean Distance*) hledáme neuron ve výstupní vrstvě, jehož váhový vektor je nejbližší aktuálnímu vstupu ve smyslu Euklidovské vzdálenosti. Vítězný neuron však můžeme hledat i na základě skalárního součinu vektorů vah jednotlivých neuronů a vstupního vektoru. Vítězem soutěže se v důsledku laterální inhibice stává ten neuron, jehož vstupní potenciál je největší a tedy předložený vstup spadá do kategorie vstupních vektorů reprezentovaných vítězným neuronem. Pokud se pokusíme vyjádřit tuto situaci prostřednictvím vektorového počtu, pak jednotlivé potenciály neuronů vyjadřují skalární součiny vektorů vah jednotlivých neuronů

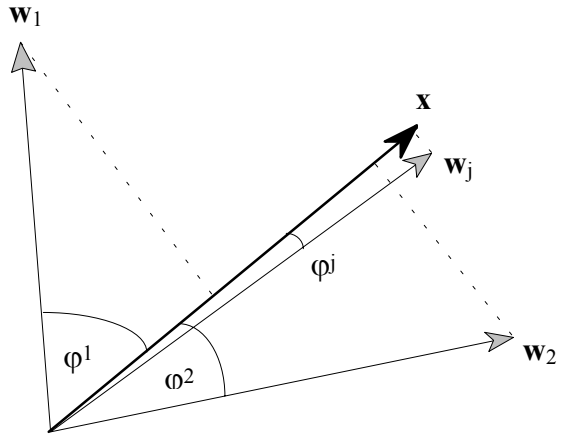
($\boldsymbol{w_j}$, $j=1,..., m$; m je počet neuronů ve výstupní vrstvě) a vstupního vektoru ($\boldsymbol{x}$) a tudíž tyto součiny můžeme chápat jako projekce vektorů vah na vstupní vektor:

$$\mathbf{w}_j^T \mathbf{x} = \left\|\mathbf{w}_j\right\| \cdot \|\mathbf{x}\| \cdot \cos\varphi$$

kde

$$\left\|w_j\right\| = \|x\| = 1.$$

Čím menší je sevřený úhel mezi oběma vektory (φ), tím delší je i projekce váhy na vstupní vektor (viz. obrázek 32). Adaptační pravidlo je založeno na částečné rotaci váhového vektoru vítězného neuronu a jeho topologických sousedů směrem ke vstupnímu vektoru $\boldsymbol{x}$. Jedná se v podstatě o učení Hebbova typu s následným normováním vektorů vah. Normování vah je nutné proto, aby se zamezilo jejich nekontrolovatelnému růstu během procesu adaptace. Současně s tím dosáhneme i efektu nalezení vítěze kompetice na základě jediného parametru, kterým je úhel mezi váhovým vektorem a aktuálním vstupním vektorem.



Obrázek 32: ***Skalární součiny vektorů vah a vstupu***

Maximální potenciál neuronu rovný 1 je zřejmě dosažitelný v případě že $\boldsymbol{w_j} = \boldsymbol{x}$. Zda-li tento j. neuron bude zastupovat i ostatní vstupní vektory závisí na tom, jak jsou tyto vektory podobné vektoru $\boldsymbol{x}$. Z obrázku je patrné, že vektory blízké původnímu vstupu téměř zachovávají i normalitu nových vah. V případě, že nový vektor je příliš vzdálen původnímu tak dochází k porušení této podmínky. Tento jev by ale měl vést ke stavu, že tento vstup bude excitovat jiný neuron výstupní vrstvy, kolem kterého by se měla vytvořit další třída, či populace vstupních vektorů. Postupně se tak vytvoří shluky vstupních neuronů odpovídajících svému neuronu výstupní vrstvy. Velmi podstatný je i fakt, že k vytvoření těchto shluků došlo (jak již bylo dříve uvedeno) prostřednictvím *adaptace bez učitele:* síť je tak schopna *samoorganizace*.

Shrnutí [4]:

Obě popsané verze Kohonenova adaptačního algoritmu spolu navzájem souvisí: tj. hledání $\max\left(\mathbf{w_j}^T \mathbf{x}\right)$ odpovídá hledání $\min\left\|\mathbf{x} - \mathbf{w}_j\right\|$ za předpokladu, že v prvním případě jsou váhové vektory $\boldsymbol{w_j}$ normované (leží na povrchu hyperkoule). Je to patrné z rovnosti

$$\left\|\mathbf{x} - \mathbf{w}_j\right\|^2 = \|\mathbf{x}\|^2 - 2\mathbf{w}_j^T \mathbf{x} + \left\|\mathbf{w}_j\right\|^2.$$

Pokud je aktuální vstup nezávislý na j a pokud je $\left\|\mathbf{w}_j\right\|^2$ konstantní díky normování, tak neuron, jehož váhový vektor je nejbližší vstupu $\boldsymbol{x}$, je současně neuronem, jehož skalární součin $\mathbf{w_j}^T \mathbf{x}$ je největší.

Kvantování vektorů učením

(LVG angl. *Learning Vector Quantization*); Prozatím jsme využívali neuronovou síť Kohonenovy mapy pro učení bez učitele. Nyní se budeme zabývat tím, jak lze tuto síť použít pro řešení problému *klasifikace* dat do několika kategorií. Ukážeme si způsob, kterým označíme výstupní neurony sítě kategoriemi a uvedeme algoritmy, které se používají pro doučení sítě, jež chceme použít k těmto účelům. Kvantování vektorů učením vychází z uvedených principů Kohonenova učení s jediným rozdílem, že místo již výše zmíněné samoorganizace chceme zajistit aby pro každou kategorii si podobných vektorů existoval jí odpovídající a námi definovaný neuron ve výstupní vrstvě sítě. Nejprve tedy musíme určit kolik takových kategorií či tříd budeme požadovat. Každé této třídě pak přiřadíme jeden neuron výstupní vrstvy. Následuje proces postupného předkládání vektorů vstupního prostoru a adaptace sítě, tentokráte s učitelem, který rozhoduje o správnosti odezvy. Vlastní odezva je realizována stejným způsobem jako v případě Kohonenových map, tj. postavená na základě kompetice. Klíčový rozdíl spočívá ve způsobu úpravy vah neuronové sítě.

Obecná architektura LVQ sítě je totožná s architekturou Kohonenovy mapy zobrazené na obrázku 27 (bez topologické struktury neuronů ve výstupní vrstvě). Navíc však má, jak už bylo uvedeno, každý výstupní neuron přiřazenou známou třídu vstupů, které reprezentuje.

Cílem *adaptačního algoritmu LVQ* sítě je nalezení takového neuronu ve výstupní vrstvě (charakterizovaného váhovými hodnotami $\boldsymbol{w}_c$), který je nejbližší pro zadaný vstupní vektor ($\boldsymbol{x}$). Algoritmus končí, pokud $\boldsymbol{x}$ i $\boldsymbol{w}_c$ patří do téže třídy klasifikace. Pokud $\boldsymbol{x}$ i $\boldsymbol{w}_c$ náleží do různých třídy klasifikace, hodnoty váhového vektoru $\boldsymbol{w}_c$ adaptujeme tak dlouho, aby byl tento nedostatek odstraněn.

Dále budeme používat následující označení:

$\mathbf{x}$	Vstupní tréninkový vektor: $\mathbf{x} = (x_1, \ldots, x_i, \ldots, x_n)$.
T	Korektní třída přiřazená tréninkovému vektoru.
$\boldsymbol{w}_j$	Vektor vah pro j. neuron ve výstupní vrstvě: $\boldsymbol{w}_j = (w_{1j}, w_{2j}, \ldots, w_{nj})^T$.
C_j	Třída reprezentující j. neuron ve výstupní vrstvě.
$\left\|\mathbf{x} - \mathbf{w}_j\right\|$	Euklidovská vzdálenost mezi vstupním vektorem $\mathbf{x}$ a váhovým vektorem j. neuronu ve výstupní vrstvě $\boldsymbol{w}_j$.

Popis algoritmu

Krok 0. Přiřazení tříd vstupním tréninkovým vektorům.

Inicializace referenčních vektorů (viz příklady).

Inicializace parametru učení (α).

Krok 1. Pokud není splněna podmínka ukončení, provádět kroky (2 až 6).

Krok 2. Pro každý vstupní vektor $\boldsymbol{x} = (x_1,\ldots,x_n)$ opakovat kroky 3 až 4.

Krok 3. Nalezení takového J, že $\left\|\mathbf{x} - \mathbf{w}_J\right\|$ je minimum.

Krok 4. Aktualizace váhových hodnot $\boldsymbol{w}_J$:

pokud $T = C_J$,pak

$$w_J(new) = w_J(old) + \alpha\left[x - w_{iJ}(old)\right]$$

pokud $T \neq C_J$, pak

$$w_J(new) = w_J(old) - \alpha\left[x - w_{iJ}(old)\right]$$

Krok 6. Aktualizace parametru učení (zmenšení jeho hodnoty).

Krok 7. Test podmínky ukončení.

V případě, že se jedná o *správnou* odezvu, adaptace probíhá podle známého vztahu:

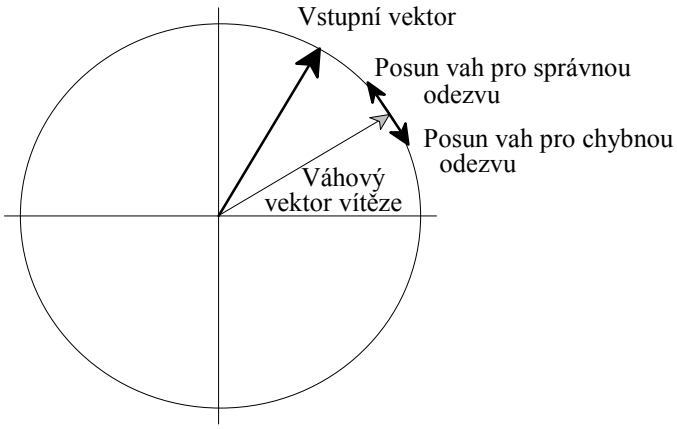
$$w_J(new) = w_J(old) + \alpha\left[x - w_{iJ}(old)\right].$$

Tímto dochází k přiblížení vah neuronu směrem ke vstupnímu vektoru.

V případě *chybné* odezvy bude našim cílem váhy chybného vítěze spíše oddálit od vstupu, což vede k následujícímu předpisu pro adaptaci jeho vah

$$w_J(new) = w_J(old) - \alpha\left[x - w_{iJ}(old)\right].$$

Celá situace je znázorněna na následujícím obrázku 33.



Obrázek 33: ***Adaptace vah pro neuronovou síť LVQ.***

Tento přístup však lze ještě dále zdokonalit. Předpokládejme, že pro předložený vstup se stal vítězem opět *j-tý* neuron namísto *k-tého*. Až doposud jsme uvažovali pouze o adaptaci vah u tohoto vítěze. V případě chybné odezvy by tedy došlo k jeho odsunutí od vstupu, zatímco váhy požadovaného *k-tého* neuronu zůstaly nezměněny. Proč tedy v rámci této adaptace neadaptovat váhy požadovaného vítěze tak, aby se přiblížil vstupnímu vektoru? Znamená to, že budeme pro všechny vstupní vektory adaptovat váhy požadovaných neuronů nezávisle na tom, zda jsou či nejsou vítězi soutěže. V případě, že vítězem se stal nežádoucí neuron, bude následovat jeho odsun od vstupu.

Úkoly:

Mějme pět vektorů: (1,1,0,0), (0,0,0,1), (0,0,1,1), (1,0,0,0), (0,1,1,0). Maximální počet shluků je: m=2. Řešte příklad a) algoritmem adaptace Kohonenovy samoorganizační mapy (vhodně si definujte vztah proparametr učení); b) adaptačním algoritmem LVQ (vhodně si rozdělte vstupní vektory do dvou kategorií). Obě řešení porovnejte.

Korespondenční úkoly:

Vytvořte počítačový program pro realizaci adaptačního algoritmu pracujícího na principu soutěžní strategie učení.

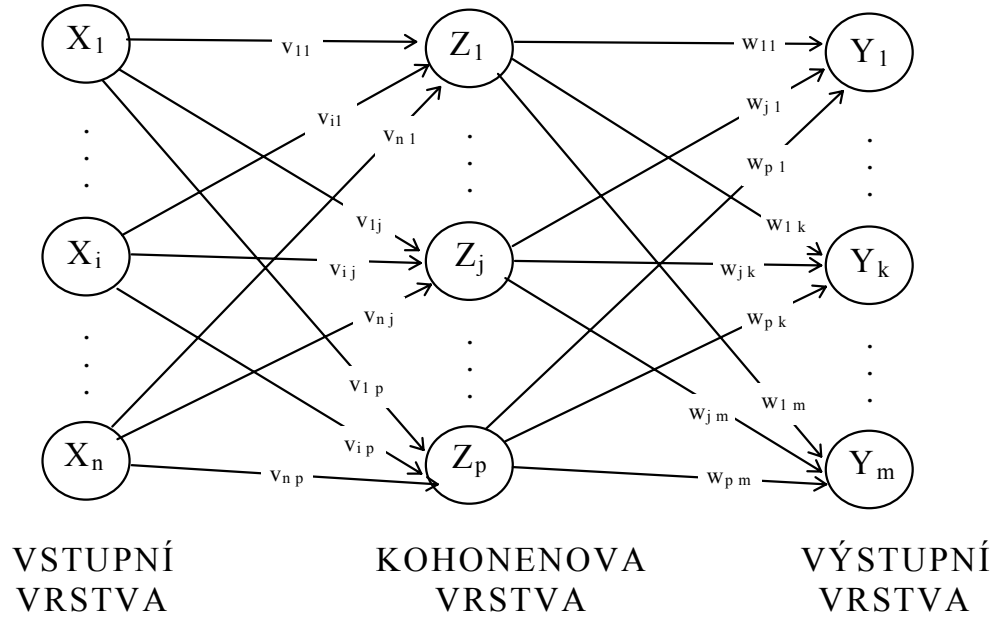
Counterpropagation je model umělé neuronové sítě (navržené Hecht-Nielsenem v r. 1986), která se snaží využít samoorganizační síť v kombinaci s dalším přídavným mechanismem k řešení problémů učení s učitelem. Síť, kterou dále popíšeme je pouze jednou z možných variant této neuronové sítě. Síť counterpropagation pracuje jako vyhledávací tabulka (lookup table), která k danému vstupu najde nejbližšího reprezentanta a odpoví výstupní hodnotou, která je s tímto reprezentantem spojena.

Klíčová slova této kapitoly:

Grossbergovy jednotky „instar“, Grossbergovo adaptační pravidlo.

Dopředná síť typu counterpropagation

Dopředná síť typu counterpropagation (angl. *Forward-Only Counterpropagation*) je tvořena třemi vrstvami neuronů (viz obrázek 34). Vstupní vrstvu tvoří n vstupních neuronů distribuujících do další vrstvy vstupní signály $x_1,..., x_n$. Druhá vrstva je tvořena p samoorganizačními jednotkami (viz Kohonenovy samoorganizační mapy), které jsou vzájemně propojeny, což není z obrázku patrné. Třetí vrstvu tvoří m Grossbergových jednotek *instar*. Jejich výstupy pak tvoří výstupy celé neuronové sítě.



O b r á z e k 3 4 : ***D o p ř e d n á s í ť t y p u c o u n t e r p r o p a g a t i o n***

Adaptivní dynamika této sítě probíhá ve dvou fázích. Nejprve se Kohonenovým učením bez učitele nastaví váhy v_{ij} $(i = 1,..., n; j = 1,..., p)$ samoorganizačních jednotek mezi vstupní a vnitřní vrstvou. Po skončení první fáze učení, ve které se používá jen vstupní části tréninkových vzorů, se váhy $\boldsymbol{v}$ fixují a dochází ke druhé fázi učení, která nastaví váhy w_{jk} $(j = 1,..., p; k = 1,..., m)$ mezi vnitřní a výstupní vrstvou. V této části adaptačního algoritmu je vypočten aktuální výstup sítě pro všechny neurony výstupní vrstvy a porovnán

s požadovaným výstupem sítě. Pokud není splněna podmínka ukončení, jsou váhové hodnoty $\boldsymbol{w}$ upravovány tzv. *Grossbergovým adaptačním pravidlem* (viz dále).

Pravidlo pro adaptaci váhových hodnot mezi vstupní a vnitřní vrstvou

$$\begin{aligned} v_{iJ}(new) &= v_{iJ} + \alpha\left(x_i - v_{iJ}\right) \\ &= (1-\alpha)v_{iJ}(old) + \alpha\, x_i , \end{aligned}$$

kde J je index vítězného neuronu v kompetici po předložení vstupu $\boldsymbol{x}$,;
x_i je inicializační hodnota *i.* neuronu ve vstupní vrstvě;
α je parametr učení; snižuje svou velikost v čase ($0<\alpha<1$);
doporučená inicializační hodnota je 0.6.

Pravidlo pro adaptaci váhových hodnot mezi vnitřní a výstupní vrstvou

$$\begin{aligned} w_{Jk}(new) &= w_{Jk} + a\left(y_k - w_{Jk}\right) \\ &= (1-a)w_{Jk}(old) + a y_k , \end{aligned}$$

kde w_{Jk} je skutečná aktivace *k.* neuronu ve výstupní vrstvě;
a je parametr učení; snižuje svou velikost v čase ($0.5 < a < 0.8$);
y_k je očekávaná aktivace *k.* neuronu ve výstupní vrstvě.

Aktivace neuronů vnitřní vrstvy definujeme následovně:

$$z_j = \begin{cases} 1 & \text{pokud } j = J \\ 0 & \text{jinak.} \end{cases}$$

Adaptační pravidlo pro váhové hodnoty na spojeních mezi vnitřní a výstupní vrstvou přepíšeme do tvaru delta pravidla (*Grossbergovo adaptační pravidlo*):

$$w_{jk}(new) = w_{jk} + a\, z_j\left(y_k - w_{Jk}\right).$$

Popis algoritmu

Krok 0. Inicializace všech váhových hodnot, parametrů učení, atd.

Krok 1. Pokud není splněna podmínka ukončení ***1. fáze*** adaptace, provádět kroky 2 až 7.

Krok 2. Pro každý vstupní vektor $\boldsymbol{x} = (x_1,...,x_n)$ opakovat kroky 3 až 5.

Krok 3. Aktivovat vstupní vrstvu vektorem $\boldsymbol{x}$.

Krok 4. Najít vítěze kompetice ve vnitřní vrstvě, označit jeho index *J.*

Krok 5. Aktualizace váhových hodnot na spojeních vedoucích k neuronu Z_J, tj. pro všechna i ($i = 1,..., n$) platí:

$$v_{iJ}(new) = (1-\alpha)v_{iJ}(old) + \alpha\, x_i$$

Krok 6. Snížit hodnotu parametru učení α.

Krok 7. Test podmínky ukončení 1. fáze.

Krok 8. Pokud není splněna podmínka ukončení ***2. fáze*** adaptace, provádět kroky 9 - 15.

(*Poznámka:* α má během celé 2. fáze adaptace velmi malou konstantní hodnotu.)

Krok 9. Pro každý tréninkový vstupní pár vektorů ***x:y***; ($\boldsymbol{x} = (x_1,...,x_n)$, $\boldsymbol{y} = (y_1,...,y_m)$), opakovat kroky 10 až 13.

Krok 10. Aktivovat vstupní vrstvu vektorem ***x***;
Aktivovat výstupní vrstvu vektorem ***y***.

Krok 11. Najít vítěze kompetice ve vnitřní vrstvě, označit jeho index *J*.

Krok 12. Aktualizace váhových hodnot na spojeních ze vstupní vrstvy do neuronu Z_J (α je velmi malé), tj. pro všechna *i* ($i = 1,..., n$) platí:

$$v_{iJ}(new) = (1-\alpha)v_{iJ}(old) + \alpha\, x_i$$

Krok 13. Aktualizace váhových hodnot na spojeních vedoucích z neuronu Z_J do výstupní vrstvy,
tj. pro všechna *k* ($k = 1,..., m$) platí:

$$w_{Jk}(new) = (1-a)w_{Jk}(old) + ay_k\ ,$$

Krok 14. Snížit hodnotu parametru učení *a*.

Krok 15. Test podmínky ukončení 2. fáze.

Shrňme nyní statistické vlastnosti naučené sítě: Díky samoorganizačnímu učení s využitím lokální paměti aproximují vektory ***v*** hustotu pravděpodobnosti vzorů. Víme, že neurony ve druhé vrstvě mají stejnou pravděpodobnost vítězství v kompetici, za předpokladu, že vybíráme vstupy náhodně s rozložením odpovídajícím tréninkové množině. Dále váhy výstupních neuronů jsou adaptovány tak, aby aproximovaly průměrnou výstupní hodnotu patřící těm vstupům, které aktivovaly odpovídající neurony ve druhé vrstvě.

Aktivní fáze counterpropagation

Krok 0. Inicializace všech váhových hodnot - viz *adaptivní fáze counterpropagation.*
Krok 1. Aktivovat vstupní vrstvu vektorem ***x***.
Krok 2. Najít vítěze kompetice ve vnitřní vrstvě, označit jeho index *J*.
Krok 3. Vypočítat aktivace neuronů výstupní vrstvy: $y_k = w_{Jk}$, ($k = 1,..., m$).

Používáme-li síť typu counterpropagation k aproximování nějakého zobrazení, **f:** $R^n \rightarrow R^m$, chová se optimálně v tom smyslu, že reprezentanti vstupů jsou zvoleni tak, aby měli stejnou pravděpodobnost výběru a výstupní hodnoty představují průměr funkčních hodnot v okolí těchto reprezentantů.

Nespornou výhodou neuronové sítě typu counterpropagation je rychlost její adaptace, nevýhodou pak je menší přesnost odezvy ve srovnáním s metodou backpropagation.

Úkoly:

Srovnejte řešení logické funkce „XOR" standardním adaptačním algoritmem vícevrstvé neuronové sítě (backpropagation) a adaptačním algoritmem modelu counterpropagation.

Na rozdíl od klasických počítačů, kdy klíčem k vyhledání položky v paměti je adresa, u asociativní paměti probíhá vybavení příslušné informace na základě její částečné znalosti (asociace). Např. v databázových aplikacích je znalost některých položek záznamu postačující k vyhledání celého záznamu. V zásadě budeme rozlišovat dva typy asociativní paměti, a to paměť *autoasociativní* a paměť *heteroasociativní*. U autoasociativní paměti půjde o upřesnění, či zúplnění vstupní informace na základě již naučeného. Naproti tomu u heteroasociativní paměti dochází k vybavení určité sdružené informace na základě vstupní asociace.

Klíčová slova této kapitoly:

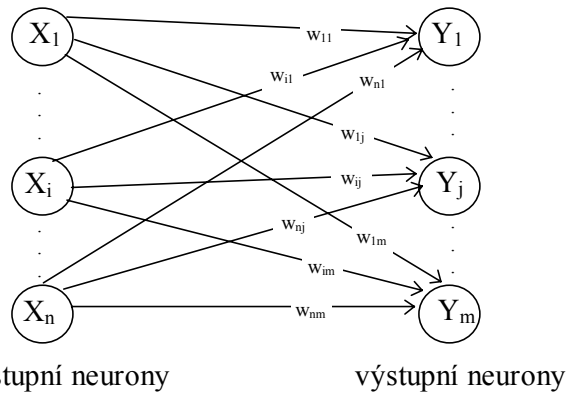
autoasociativní paměť, heteroasociativní paměť

Organizační i aktivní dynamika asocialitivní sítě je téměř identická jako u modelu *Madaline*. Jediný rozdíl spočívá v tom, že lineární asociativní síť v aktivním režimu místo afinních kombinací počítá jen lineární kombinace vstupů, tj. chybí formální jednotkový vstup rovněž i odpovídající biasy jsou nulové. V geometrické interpretaci to znamená, že příslušné nadroviny odpovídající výstupním neuronům sítě prochází počátkem.

Asociativní paměti neuronových sítí jsou sítě, ve kterých jsou váhové hodnoty determinovány takovým způsobem, aby si sítě byly schopny zapamatovat množinu P asociovaných vzorů. Každou asociaci tvoří pár vektorů ($\boldsymbol{s}(p), \boldsymbol{t}(p)$), kde $p = 1, 2, ..., P$). Každý vektor $\boldsymbol{s}(p)$ obsahuje n komponent a každý vektor $\boldsymbol{t}(p)$ obsahuje m komponent. Váhové hodnoty na příslušných spojích mohou být nalezeny např. Hebbovým adaptačním pravidlem pro asociované neuronové sítě. (viz dále). Slovně jej lze vyjádřit takto: změna synaptické váhy spoje mezi dvěma neurony je úměrná jejich souhlasné aktivitě, tj. součinu jejich stavů (opačná aktivita tuto vazbu zeslabuje). Lineární asociativní síť má schopnost *reprodukce*, tj. předložíme-li síti vstup (vstupní vektor $\boldsymbol{x}$), pak na něj odpoví požadovaným výstupem (výstupní vektor $\boldsymbol{y}$). Vstupní vektor $\boldsymbol{x}$ může být buď vektorem z tréninkové množiny, nebo jiným vektorem (tj. vektorem z tréninkové množiny obsahující šum).

Heteroasociativní paměť neuronové sítě

Architektura heteroasociativní paměti neuronové sítě je zobrazena na obrázku 35. Její adaptace probíhá podle Hebbova adaptačního pravidla pro asociované neuronové sítě.



Obrázek 35: ***Architektura heteroasociativní paměti neuronové sítě.***

Hebbovo adaptační pravidlo pro asociované neuronové sítě

je nejběžnější metodou pro stanovení váhových hodnot na spojích mezi jednotlivými neurony. Pracuje s vektory, které jsou zapsány v binární i bipolární reprezentaci. Jeho algoritmus probíhá v následujících krocích. Algoritmus není vhodný pro dopředné neuronové sítě, které adaptujeme metodou backpropagation.

Popis algoritmu

Krok 0. Inicializace všech váhových hodnot $w_{ij} = 0$, $(i = 1,...,n; j = 1,...m)$.

Krok 1. Pro každý testovací vzor, tj. tréninkový pár **s:t,** opakovat kroky (2 až 4).

Krok 2. Inicializovat *vrstvu X* vnějším vstupním vektorem.

$x_i = s_i$, $(i = 1,...,n)$.

Krok 3. Inicializovat *vrstvu Y* vnějším vstupním vektorem..

$y_j = t_j$, $(j = 1,...m)$.

Krok 4. Nastavit váhové hodnoty $(i = 1,...,n; j = 1,...m)$;

$$w_{ij}(new) = w_{ij}(old) + x_i y_j .$$

Celý algoritmus přepíšeme ještě názornějším způsobem:
Nejprve inicializujeme všechny váhové hodnoty číslem 0, tj. $w_{ij} = 0$, $(i = 1,...,n; j = 1,...m)$.

Vstupní vektor

$$\boldsymbol{s} = (s_1, ..., s_i, ..., s_n)$$

tvoří sloupcovou matici **S** typu $n \times 1$, *tj* $\boldsymbol{S} = \boldsymbol{s}^T$.

Asociovaný výstupní vektor

$$\boldsymbol{t} = (t_1, ..., t_j, ..., t_m)$$

tvoří řádkovou matici **T** typu $1 \times m$, *tj* $\boldsymbol{T} = \boldsymbol{t}$.

Součin obou matic **S** a **T**

$$\mathbf{ST} = \begin{bmatrix} s_1 \\ \vdots \\ s_i \\ \vdots \\ s_n \end{bmatrix} \begin{bmatrix} t_1 & \cdots & t_j & \cdots & t_m \end{bmatrix} = \begin{bmatrix} s_1 t_1 & \cdots & s_1 t_j & \cdots & s_1 t_m \\ \vdots & . & \vdots & . & \vdots \\ s_i t_1 & \cdots & s_i t_j & \cdots & s_i t_m \\ \vdots & . & \vdots & . & \vdots \\ s_n t_1 & \cdots & s_n t_j & \cdots & s_n t_m \end{bmatrix}$$

pak tvoří váhovou matici, ve které jsou uloženy informace o asociovaném páru vektorů ***s:t***.

Jelikož budeme dále pracovat s P vzory, musí být i ve váhové matici uložené informace o P asociovaných vektorech $\boldsymbol{s}(p):\boldsymbol{t}(p)$, $p = 1, 2,..., P$, *kde*

$$\boldsymbol{s}(p) = (s_1(p), ..., s_i(p), ..., s_n(p))$$

a

$$\boldsymbol{t}(p) = (t_1(p), ..., t_j(p), ..., t_m(p)).$$

Tyto vektory tvoří váhovou matici $\boldsymbol{W} = \{w_{ij}\}$, tj.

$$w_{ij} = \sum_{p=1}^{P} s_i(p)\, t_j(p),$$

která má ve vektorové reprezentaci tvar:

$$\mathbf{W} = \sum_{p=1}^{P} \mathbf{s}^T(p)\mathbf{t}(p).$$

Nyní můžeme přistoupit k popisu algoritmu heteroasociativní paměti, který nalezne pro každý vstupní vektor ***x*** vhodný výstupní vektor ***y***. Vstupní vektor ***x*** může být, jak již bylo uvedeno, buď jedním z naučených vzorů, nebo novým vzorem (např. tréninkovým vzorem obsahujícím šum).

Popis algoritmu heteroasociativní paměti

Krok 0. Inicializace všech váhových hodnot w_{ij}, ($i = 1,...,n; j = 1,...m$) podle Hebbova pravidla adaptace pro asociativní sítě.

Krok 1. Pro vstupní vektor $\boldsymbol{x} = (x_1, ..., x_i, ..., x_n)$ opakovat kroky (2 až 4).

Krok 2. Inicializovat *vrstvu X* daným vstupním vektorem.

Krok 3. Vypočítat potenciál výstupních neuronů

$$y\_in_j = \sum_i x_i w_{ij}\ , (j = 1,..., m).$$

Krok 4a. Vypočítat aktivaci výstupních neuronů;

$$y_j = \begin{cases} 1 & \text{pokud } y\_in_j > 0 \\ 0 & \text{pokud } y\_in_j = 0 \\ -1 & \text{pokud } y\_in_j < 0, \end{cases}$$

pro bipolární reprezentaci (práh $\theta = 0$).

Krok 4b. Vypočítat aktivaci výstupních neuronů;

$$y_j = \begin{cases} 1 & \text{pokud } y\_in_j > 0 \\ 0 & \text{pokud } y\_in_j \le 0, \end{cases}$$

pro binární reprezentaci(práh $\theta = 0$).

Poznámka:

Heteroasociativní paměť není iterační.

Autoasociativní neuronové sítě

Dopředné autoasociativní neuronové sítě jsou speciálním případem heteroasociativních sítí popsaných v předcházející kapitole. Pro autoasociativní sítě jsou oba tréninkové vektory (tj. vstupní vektor ***s*** a výstupní vektor ***t***) identické. Každou asociaci proto tvoří pár vektorů ***s(p)***:***s(p)***, kde $p = 1, 2,..., P$. Každý vektor ***s(p)*** obsahuje n komponent. Váhové hodnoty na příslušných spojích jsou rovněž nastaveny Hebbovým adaptačním pravidlem pro asociované neuronové sítě (při řešení úloh jsou dosahovány lepší výsledky s vektory pracující s bipolární reprezentací než s vektory pracující v binární reprezentaci).

Příklad:

Popíšeme proces uložení jednoho vzoru reprezentovaného vektorem $\boldsymbol{s}$ =(1, 1, 1, -1) v autoasociativní paměti a pak jeho následné vybavení.
Jak již bylo uvedeno, autoasociativní paměť je speciálním případem heteroasociativní paměti (kde $\boldsymbol{t}(p) = \boldsymbol{s}(p)$). Při řešení příkladu proto využijeme algoritmus uvedených v předcházející kapitole.

Krok 0. Vektor $\boldsymbol{s}$ =(1, 1, 1, -1) je uložen ve váhové matici

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & 1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{bmatrix}.$$

Krok 1. Pro vstupní vektor $\boldsymbol{s}$ =(1, 1, 1, -1).

Krok 2. $\boldsymbol{x}$ = (1, 1, 1, -1).

Krok 3. $\boldsymbol{y\_in}$ = (4, 4, 4, -4).

Krok 4a. $\boldsymbol{y}$ = f(4, 4, 4, -4) = (1, 1, 1, -1).

Vidíme, že vektor **y** je totožný s vektorem **s**. Lze tedy říci, že na vstup byl dodán vektor z tréninkové množiny.

Speciálním případem autoasociativní neuronové sítě je *iterativní autoasociativní síť*. Z následujícího příkladu uvidíme, že síť v určitém případě nereaguje přímo na vstupní signál naučeným výstupem. Pokud vstupní signál není totožný s naučeným vzorem, ale liší se od něj pouze v tom smyslu, že místo +1 nebo --1 obsahuje 0, potom lze výstupní hodnoty ze sítě opět považovat za její vstupní signál, atd. Požadovaný výstupní signál pak dostaneme po určitém počtu *iterací*.

Příklad.

Mějme v autoasociativní paměti uložen jeden vzor reprezentovaný vektorem $\boldsymbol{s}$ =(1, 1, 1, -1). Váhová matice má tvar (v autoasociativních sítích nejsou neurony spojeny samy se sebou, je tedy běžnější zapisovat váhovou matici s nulami na hlavní diagonále, aniž by to mělo vliv na další řešení úlohy):

$$\mathbf{W} = \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}.$$

Vstupní vektor $\boldsymbol{s}$ =(1, 0, 0, 0) je případ vektoru, ve kterém jsou na rozdíl od naučeného vzoru nahrazeny tři jeho komponenty nulami. Iterační proces pak s tímto vektorem probíhá následovně:

Vstupní vektor $\boldsymbol{s}$ =(1, 0, 0, 0):

$$(1, 0, 0, 0) \cdot \boldsymbol{W} = (0, 1, 1, -1) \rightarrow \textit{iterace}$$
$$(0, 1, 1, -1) \cdot \boldsymbol{W} = (3, 2, 2, -2) \rightarrow (1, 1, 1, -1),$$

což je uložený vzor. Pokud je tedy na vstupu vektor (1, 0, 0, 0), tak po dvou iteracích bude na výstupu vektor (1, 1, 1, -1).

Úkoly:

Důkladně si ještě jednou prostudujte Hebbovo adaptační pravidlo pro asociované neuronové sítě (včetně obou řešených příkladů). V další kapitole na něj budeme navazovat!

HOPFIELDOVA SÍŤ.

Model Hopfieldovy sítě vychází z iterační autoasociativní paměti. Proto dříve než se pustíte do studia této kapitoly vám doporučuji, abyste se důkladně seznámili s obsahem kapitoly „*Asociativní neuronové sítě*“.

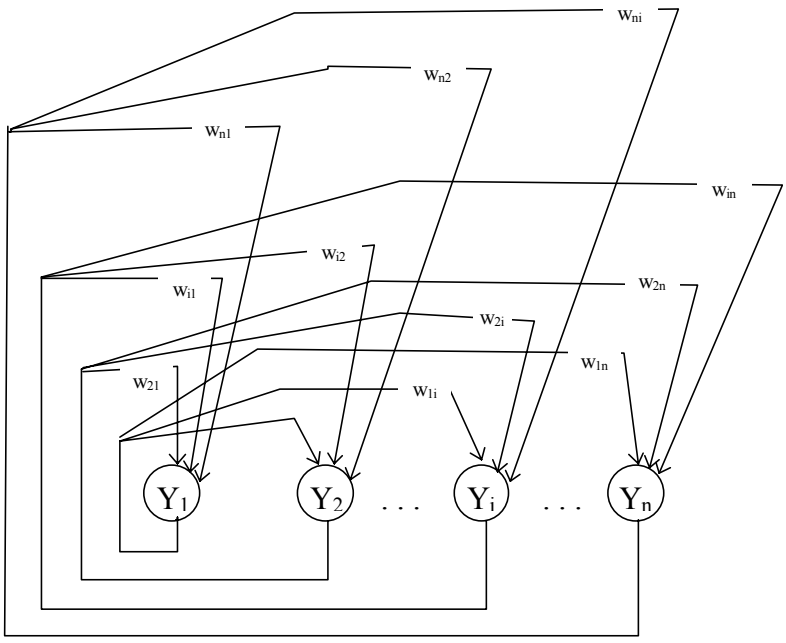
V této kapitole se seznámíte s modelem diskrétní i spojité Hopfieldovy sítě. V závěru je pak uveden algoritmus řešení „*problému obchodního cestujícího*“ spojitou Hopfieldovou síti.

Klíčová slova této kapitoly:

diskrétní Hopfieldova síť, spojitá Hopfieldova síť, proces relaxace, energetická funkce sítě, problém obchodního cestujícího (angl. Travelling Salesman Problem - TSP).

Diskrétní Hopfieldova síť

Diskrétní Hopfieldova síť se používá jako iterační autoasociativní paměť. Autorem této neuronové sítě je John Hopfield, který se zabýval studiem neuronů podobných perceptronům. Model Hopfieldovy neuronové sítě je založen na využití energetické funkce svázané s neuronovou sítí tak, jak je to běžné u fyzikálních systémů. Organizační dynamika diskrétní Hopfieldovy sítě specifikuje úplnou topologii cyklické neuronové sítě s n neurony, kde každý neuron v síti je spojen se všemi ostatními neurony sítě, tj. má všechny neurony za své vstupy. Obecně platí, že může být spojen i sám se sebou. Všechny neurony v síti jsou tedy zároveň vstupní i výstupní. Architektura Hopfieldovy sítě je znázorněna na obrázku 36. Každý spoj v síti mezi neuronem i ($i = 1, ..., n$) a neuronem j ($j = 1, ..., n$) je ohodnocen celočíselnými synaptickými vahami w_{ij} a w_{ji}, které jsou symetrické, tj. $w_{ij} = w_{ji}$. V základním modelu platí, že žádný neuron není spojen sám se sebou, tj. odpovídající váhy $w_{jj} = 0$ ($j = 1, ..., n$) jsou nulové.



Obrázek 36: ***Model diskrétní Hopfieldovy sítě.***

Hlavní myšlenka adaptace Hopfieldova modelu spočívá v tom, že jsou nejprve inicializovány všechny neurony sítě buď binárními hodnotami {0, 1} nebo bipolárními hodnotami {-1,+1}. Vzhledem k tomu, že jsou všechny neurony navzájem propojeny, začnou se ovlivňovat. To znamená, že jeden neuron se snaží ostatní neurony excitovat na rozdíl od jiného, který se snaží o opačné. Probíhá cyklus postupných změn excitací neuronů až do okamžiku nalezení kompromisu - síť relaxovala do stabilního stavu. Jinými slovy výstupy předchozího kroku se staly novými vstupy současného kroku. Tento proces je vysvětlitelný následujícím algoritmem: tréninkové vzory nejsou v Hopfieldově síti uloženy přímo, ale jsou reprezentovány pomocí vztahů mezi stavy neuronů.

První popis adaptačního algoritmu Hopfieldovy sítě pochází z roku 1982 a používá binární hodnoty pro excitace neuronů.

Požadovaná funkce sítě je specifikována tréninkovou množinou P vzorů $\mathbf{s}(p)$, $p = 1, ..., P$, z nichž každý je zadán vektorem n binárních stavů vstupních resp. výstupních neuronů, které v případě autoasociativní paměti splývají:

$$\mathbf{s}(p) = \left(s_1(p), \ldots, s_i(p), \ldots, s_n(p)\right),$$

potom je váhová matice $\mathbf{W} = \{w_{ij}\}$ dána následujícím vztahem:

$$w_{ij} = \sum_p \left[2s_i(p) - 1\right]\left[2s_j(p) - 1\right] \qquad \text{pro } i \neq j$$

a

$$w_{i\,i} = 0.$$

Jiný popis adaptačního algoritmu Hopfieldovy sítě pochází z roku 1984 a pracuje s bipolárními hodnotami pro excitace neuronů.

Požadovaná funkce sítě je rovněž specifikována tréninkovou množinou P vzorů $\mathbf{s}(p)$, $p = 1, ..., P$, z nichž každý je zadán vektorem n bipolárních stavů vstupních resp. výstupních neuronů, které v případě autoasociativní paměti splývají:

$$\mathbf{s}(p) = \left(s_1(p), \ldots, s_i(p), \ldots, s_n(p)\right),$$

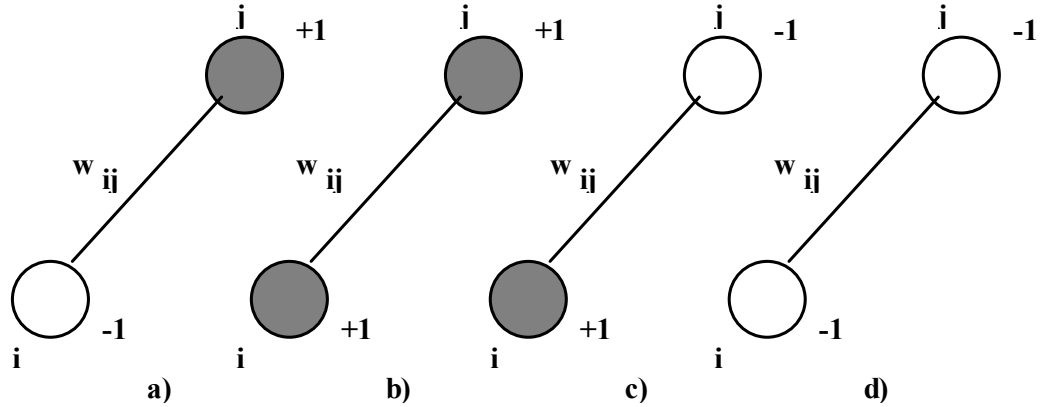
potom je váhová matice $\mathbf{W} = \{w_{ij}\}$ dána následujícím vztahem:

$$w_{ij} = \sum_p s_i(p)\, s_j(p) \qquad \text{pro } i \neq j$$

a

$$w_{i\,i} = 0.$$

Smysl této adaptace vah si ozřejmíme na následujícím obrázku:



Obrázek 37: ***Adaptace vah Hopfieldovy sítě***

V případě varianty b) a d) je stav excitace obou neuronů totožný. Dle výše uvedeného to znamená, že nová hodnota váhy na spojení mezi oběma neurony bude dána vztahem:

$$w_{ij}(new) = w_{ij}(old) + x_i \cdot x_j = w_{ij}(old) + 1.$$

Znamená to "posílení" propojení mezi těmito neurony a v případě relaxace sítě pak oba neurony budou mít snahu dosáhnout stejného stavu. Čím více bude vzorů s tímto stavem obou neuronů, tím větší bude snaha o dosažení totožného stavu obou těchto neuronů při relaxaci.

V případech a) a c) pak bude postup obrácený. Nová váha propojení bude mít následující hodnotu:

$$w_{ij}(new) = w_{ij}(old) + x_i \cdot x_j = w_{ij}(old) - 1$$

a vazba bude směřovat k takovému stavu, aby při relaxaci sítě byly stavy obou neuronů různé.

Popis algoritmu

Dále budeme pracovat s binárním vzorem, protože vyjádření aktivační funkce v bipolární reprezentaci je mnohem jednodušší než kdybychom postupovali opačně.

Krok 0. Inicializace vah, tj. zapamatování vzorů.

(***Použitím Hebbova adaptačního pravidla pro asociované sítě***.)

Dokud síť nezrelaxovala do stabilního stavu, opakovat kroky (1 až 7).

Krok 1. Pro každý vstupní vektor $\boldsymbol{x}$, opakovat kroky (2 až 6).

Krok 2. Inicializace sítě vnějším vstupním vektorem $\boldsymbol{x}$:

$$y_i = x_i, \quad (i = 1, \ldots, n)$$

Krok 3. Pro každý neuron Y_i opakovat kroky (4 až 6).

(Neurony jsou uspořádány náhodně)

Krok 4 Vypočítat vnitřní potenciál neuronu:

$$y\_in_i = x_i + \sum_j y_j w_{ji}.$$

Krok 5 Stanovení výstupu neuronu lze chápat jako aplikaci aktivační funkce:

$$y_i = \begin{cases} 1 & \text{pokud } y\_in_i > \theta_i \\ y_i & \text{pokud } y\_in_i = \theta_i \\ 0 & \text{pokud } y\_in_i < \theta_i. \end{cases}$$

Krok 6 Transport hodnoty y_i ostatním neuronům.

(Takto budeme aktualizovat hodnoty aktivačního vektoru.)

Krok 7. Test konvergence.

Prahová hodnota θ_i je obvykle nulová. Aktualizace neuronů probíhají sice v náhodném pořadí, ale musí být prováděny stejnou průměrnou rychlostí.

Příklad:

Pomocí diskrétního Hopfieldova modelu určit, jestli je vstupní vektor „naučeným“ vzorem (tj. byl součástí trénovací množiny) nebo „nenaučeným“ vzorem.

Krok 0. Váhová matice pro zapamatování vektoru (1, 1, 1, 0) (přepis vektoru do bipolární reprezentace (1, 1, 1, -1)) má tvar:

$$\mathbf{W} = \begin{bmatrix} 0 & 1 & 1 & -1 \\ 1 & 0 & 1 & -1 \\ 1 & 1 & 0 & -1 \\ -1 & -1 & -1 & 0 \end{bmatrix}$$

Krok 1. Vstupní vektor je $\boldsymbol{x} = (0, 0, 1, 0)$. Pro tento vektor opakovat kroky (2 až 6).

Krok 2. $\boldsymbol{y} = (0, 0, 1, 0)$.

Krok 3. Vybrat Y_1 a aktualizovat jeho aktivaci:

Krok 4 $$y\_in_1 = x_1 + \sum_j y_j w_{j1} = 0 + 1.$$

Krok 5 $y\_in_1 > 0 \rightarrow y_1 = 1.$

Krok 6 $\boldsymbol{y} = (1, 0, 1, 0)$.

Krok 3. Vybrat Y_4 a aktualizovat jeho aktivaci:

Krok 4 $$y\_in_4 = x_4 + \sum_j y_j w_{j4} = 0 + (-2).$$

Krok 5 $y\_in_1 < 0 \rightarrow y_4 = 0.$

Krok 6 $y = (1, 0, 1, 0).$

Krok 3. Vybrat Y_3 a aktualizovat jeho aktivaci:

Krok 4 $$y\_in_3 = x_3 + \sum_j y_j w_{j3} = 1 + 1.$$

Krok 5 $y\_in_3 > 0 \rightarrow y_3 = 1.$

Krok 6 $\boldsymbol{y} = (1, 0, 1, 0)$.

Krok 3. Vybrat Y_2 a aktualizovat jeho aktivaci:

Krok 4 $$y\_in_2 = x_2 + \sum_j y_j w_{j2} = 0 + 2.$$

Krok 5 $y\_in_2 > 0 \rightarrow y_2 = 1.$

Krok 6 $\boldsymbol{y} = (1, 1, 1, 0)$.

Krok 7. Test konvergence.

Aktivace každého neuronu byla aktualizována alespoň jednou během celého výpočtu. Vstupní vektor konverguje k uloženému vzoru.

Funkce energie

K lepšímu pochopení aktivní dynamiky Hopfieldovy sítě byla Hopfieldem, v analogii s fyzikálními ději definována tzv. *energetická funkce E* sítě, která každému stavu sítě přiřazuje jeho potenciální energii. Energetická funkce je tedy funkce, která je zdola ohraničená a pro daný stav systému je nerostoucí. V teorii neuronových sítí se *stavem systému* rozumí množina aktivací všech neuronů. Pokud je již tato energetická funkce

nalezena, bude síť konvergovat ke stabilní množině aktivací neuronů v daném časovém okamžiku. Energetická funkce pro diskrétní Hopfieldovu síť je dána následovně:

$$E = -0{,}5\sum_{i\neq j}\sum_{j} y_i y_j w_{ij} - \sum_i x_i y_i + \sum_i \theta_i y_i .$$

Z definice energetické funkce vyplývá, že stavy sítě s nízkou energií mají největší stabilitu. Pokud se aktivace sítě změní o Δy_i , změna energie je pak dána následovně:

$$\Delta E = -\left[\sum_j y_j w_{ij} + x_i - \theta_i\right]\Delta y_i .$$

(Vztah závisí na skutečnosti, že v daném časovém okamžiku může svou aktivaci aktualizovat vždy pouze jeden neuron sítě.)

Nyní uvažujme dva případy, které mohou nastat při změně Δy_i v aktivaci neuronu Y_i.; Δy_i je kladné, pokud je i výraz $\left[\sum_j y_j w_{ij} + x_i - \theta_i\right]$ kladný a Δy_i je záporné, pokud je tentýž výraz záporný. V obou případech je $\Delta E<0$, tj. energie nemůže růst. Protože je energetická funkce ohraničená, síť musí dosáhnout stabilního stavu (tj. energie se v následujících iteracích již nemění).

Hopfieldova síť má ve srovnání s vícevrstvou sítí adaptovanou učícím algoritmem backpropagation opačný charakter aktivní a adaptivní dynamiky. Zatímco adaptace Hopfieldovy sítě podle Hebbova zákona je jednorázovou záležitostí, jejíž trvání závisí jen na počtu tréninkových vzorů, učící algoritmus backpropagation realizuje iterativní proces minimalizující chybu sítě gradientní metodou bez záruky konvergence. Na druhou stranu délka trvání aktivní fáze vícevrstvé sítě je dána pouze počtem vrstev, zatímco aktivní režim Hopfieldovy sítě představuje iterativní proces minimalizující energii sítě diskrétní variantou gradientní metody s nejistou konvergencí.

Cílem adaptace Hopfieldovy sítě podle Hebbova zákona je nalezení takové konfigurace, aby funkce sítě v aktivním režimu realizovala autoasociativní paměť. To znamená: bude-li vstup sítě blízký nějakému tréninkovému vzoru, výstup sítě by měl potom odpovídat tomuto vzoru. Z hlediska energie by každý tréninkový vzor měl být lokálním minimem energetické funkce, tj. stabilním stavem sítě. V jeho blízkém okolí, v tzv. *oblasti atrakce*, se nachází všechny vstupy blízké tomuto vzoru. Ty představují počáteční stavy sítě, ze kterých se při minimalizaci energetické funkce v aktivním režimu síť dostane do příslušného minima, tj. stabilního stavu odpovídajícího uvažovanému tréninkovému vzoru. Geometricky se tedy energetická plocha rozpadá na oblasti atrakce lokálních minim a příslušná funkce Hopfieldovy sítě přiřadí v aktivním režimu ke každému vstupu náležejícímu do oblasti atrakce nějakého lokálního minima právě toto minimum.

Při učení Hopfieldovy sítě podle Hebbova zákona pro asociativní sítě samovolně vznikají na energetické ploše lokální minima, tzv. *nepravé vzory* (*fantomy*), které neodpovídají žádným tréninkovým vzorům. Výstup sítě pro vstup dostatečně blízký takovému fantomu neodpovídá žádnému vzoru, a tudíž nedává žádný smysl. Existují varianty adaptivní dynamiky Hopfieldovy sítě, při nichž se takto vzniklé fantomy mohou dodatečně odučit.

Kapacita Hopfieldovy paměti

Hopfield experimentálně nalezl, že počet binárních vzorů, který může být zapamatován a opětovně vyvolán s požadovanou přesností, je dán přibližně

$$P \approx 0{,}15n,$$

kde n je počet neuronů v síti.

Pro sítě pracující s bipolárními vzory byl odvozen obdobný vztah:

$$P \approx \frac{n}{2\log_2 n}.$$

I když se v praxi ukazuje, že uvedené teoretické odhady jsou poněkud nadhodnocené, přesto základní model Hopfieldovy autoasociativní paměti má díky své malé kapacitě spíše teoretický význam. V literatuře přesto existuje mnoho modifikací tohoto modelu, které se snaží uvedený nedostatek odstranit.

Spojitá Hopfieldova síť

Spojitá Hopfieldova síť je příkladem modelu, u kterého je vývoj reálného stavu v aktivním režimu nejen spojitou funkcí vnitřního potenciálu, ale navíc i spojitou funkcí času. Aktivní dynamika je v takových případech obvykle zadána diferenciální rovnicí, jejíž řešení nelze explicitně vyjádřit, proto tyto modely (pokud nepracujeme s jejich diskrétní verzí) nejsou vhodné pro analogovou hardwarovou implementaci pomocí elektrických obvodů.

Protože spojitá Hopfieldova síť je modifikací diskrétní Hopfieldovy sítě, jsou také spojení mezi libovolnými dvěma neurony obousměrné a rovněž i váhové hodnoty na těchto spojeních jsou symetrické.

V tomto modelu budeme používat následující značení:

U_i	*i.* neuron.
w_{ij}	Váhová hodnota přiřazena spojení mezi U_i a U_j; $w_{ij} = w_{ji.}$
u_i	Vnitřní potenciál neuronu U_i.
$v_i=g(u_i)$	Aktivace neuronu U_i.

Pokud budeme definovat funkci energie vztahem

$$E = 0{,}5\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} v_i v_j + \sum_{i=1}^{n} \theta_i v_i ,$$

potom bude síť konvergovat ke stabilní konfiguraci, tj. funkce energie dosáhne svého minima, když

$$\frac{d}{dt} E \le 0.$$

Podle tohoto tvaru energetické funkce bude síť konvergovat, budou-li se aktivity všech neuronů měnit (v čase) podle následující diferenciální rovnice

$$\frac{d}{dt} u_i = -\frac{\partial E}{\partial v_i} = -\sum_{j=1}^{n} w_{ij} v_j - \theta_i ,$$

jak je ukázáno dále.

Spojitá Hopfieldova síť může být použita buď jako autoasociativní paměť(stejně jako diskrétní Hopfieldova síť), nebo k řešení optimalizačních problémů zadaných formou omezujících podmínek. Mezi takto zadané úlohy patří např. *Problém obchodního cestujícího.* Princip hledání lokálního minima energetické funkce Hopfieldovy sítě je v této úloze využitelný zcela jiným způsobem než bylo zatím uvedeno: Dokážeme-li formulovat omezení nějaké optimalizační úlohy ve formě energetické funkce neuronové sítě, pak proces její relaxace povede k nalezení některého z optimálních, či alespoň suboptimálních řešení. Ve srovnání z předchozím tedy nebudeme síť adaptovat na základě prvků trénovací množiny, ale pokusíme se stanovit váhy mezi jednotlivými neurony na základě porovnání obecně definované funkce energie Hopfieldovy sítě a energetické funkce vyjadřující naše omezující podmínky. Tento proces probíhá v adaptivním režim sítě. V aktivním režimu potom hledá síť přípustné řešení daného problému.

Spojitá Hopfieldova síť a problém obchodního cestujícího

Problém obchodního cestujícího (angl. *Travelling Salesman Problem* - TSP) je klasickou úlohou, kterou lze výše uvedeným postupem úspěšně řešit. Cílem úlohy je navštívit všechna města oblasti tak, aby žádné z nich nebylo navštíveno dvakrát a přitom, aby délka trasy byla co nejmenší. Nejlepší řešení pro TSP je velmi složité najít, neboť čas potřebný pro jeho nalezení roste exponenciálně s počtem měst. Proto každé "dostatečně dobré" řešení bude pro nás zajímavé.

Omezení úlohy TSP lze formulovat následovně: *každé město může být navštíveno pouze jednou a trasa musí být co nejkratší*. Pokud se nám podaří sestavit energetickou funkci sítě tak, že bude tato omezení odrážet, pak její minimalizace povede k řešení optimalizující zmíněná omezení. Poněvadž výsledkem je seznam měst navštívených v určitém pořadí, budeme potřebovat nějakým způsobem tento fakt vyjádřit. Jestliže budeme chtít navštívit n měst, pak každé z nich se bude nacházet v seznamu na některé z n pozic. Pro potřeby řešení úlohy TSP tedy použijeme čtvercovou matici obsahující n x n neuronů (všechny jsou vzájemně propojené), kde města jsou reprezentována řádky matice a pořadí jejími sloupci.

Město/Pořadí	1	2	3	...	n
A	$U_{A,1}$	$U_{A,2}$	$U_{A,3}$	...	$U_{A,n}$
B	$U_{B,1}$	$U_{B,2}$	$U_{B,3}$	...	$U_{B,n}$
C	$U_{C,1}$	$U_{C,2}$	$U_{C,3}$	...	$U_{C,n}$
...	...	...	...	...	...
N	$U_{N,1}$	$U_{N,2}$	$U_{N,3}$	...	$U_{N,n}$

Aktivitu neuronu v i-tém sloupci a j-tém řádku interpretujeme jako skutečnost, že obchodní cestující navštíví j-té město jako i-té v pořadí na své trase.

Pokusme se nyní formulovat jednotlivá omezení. Je celkem zřejmé, že ne všechny stavy neuronové sítě odpovídají přípustnému řešení problému. Přípustnost řešení je zapotřebí zohlednit v příslušné minimalizované účelové funkci. Prvním indexem - *x, y, atd.* budeme označovat „město“, zatímco druhým indexem - *i, j, atd.* budeme označovat jeho „pořadí“ na trase.

Prvním požadavkem je, aby obchodní cestující navštívil každé město nejvýše jednou, tzn. aby na konci adaptace sítě byl v každém řádku aktivní nejvýše jeden neuron. Tomu odpovídá minimalizace následujícího výrazu:

$$E_A = \frac{A}{2} \sum_x \sum_i \sum_{j \neq i} v_{x,i} v_{x,j} \ .$$

Parametr A>0 je míra vlivu E_A při minimalizaci celkové energetické funkce.

Druhým požadavkem je, aby obchodní cestující byl při každé ze svých zastávek pouze v jednom městě, tzn. aby byl na konci adaptace sítě v každém sloupci aktivní nejvýše jeden neuron. Tomu odpovídá minimalizace následujícího výrazu:

$$E_B = \frac{B}{2} \sum_i \sum_x \sum_{x \neq y} v_{x,i} v_{y,i} \ .$$

Parametr B>0 je míra vlivu E_B při minimalizaci celkové energetické funkce.

Třetím požadavkem je, aby obchodní cestující projel všemi n městy, čemuž odpovídá aktivita právě n neuronů v síti. Toho dosáhneme minimalizací následujícího výrazu:

$$E_C = \frac{C}{2} \left[n - \sum_x \sum_i v_{x,i} \right]^2 .$$

Parametr C>0 je opět míra vlivu E_C při minimalizaci celkové energetické funkce, n je celkový počet měst na trase.

Současnou minimalizací E_A, E_B, E_C ve stavovém prostoru sice máme zajištěno, že neuronová síť skončí svou činnost ve stavu odpovídajícím přípustnému řešení problému obchodního cestujícího, tzn. že bude nalezena okružní trasa, ale její délka nemusí být zrovna nejkratší. Proto ještě musíme uvedenou podmínku zohlednit v účelové funkci, a to následovně:

Čtvrtým požadavkem je, aby nalezená trasa byla co nejkratší. Toho dosáhneme, budeme-li minimalizovat následující výraz:

$$E_D = \frac{D}{2} \sum_x \sum_{y \neq x} \sum_i d_{x,y} v_{x,i} \left(v_{y,i+1} + v_{y,i-1} \right)$$

kde $d_{x,y}$ určuje vzdálenost mezi x-tým a y-tým městem. Parametr $D>0$ je opět míra vlivu E_D při minimalizaci celkové energetické funkce.

Výslednou účelovou funkci E, pomocí které získáme přípustné optimální řešení problému obchodního cestujícího získáme součtem všech dílčích energetických funkcí, tj.

$$E = E_A + E_B + E_C + E_D .$$

Dále musíme uvést diferenciální rovnici pro nalezení aktivace neuronu U_X:

$$\frac{d}{dt} u_{X,I} = -\frac{u_{X,I}}{\tau} - A \sum_{j \neq I} v_{X,j} - B \sum_{y \neq X} v_{y,I} - C \left[n - \sum_x \sum_i v_{x,i} \right] - D \sum_{y \neq X} d_{X,y} \left(v_{y,I+1} + v_{y,I-1} \right) .$$

Řešením rovnice dostaneme výstupní signál v_i, jehož hodnota je určena aplikací sigmoidní aktivační funkce (v intervalu mezi 0 a 1)

$$v_i = g(u_i) = 0.5 \left[1 + \tanh(\alpha \, u_i) \right].$$

Porovnáním účelové funkce obchodního cestujícího s energetickou funkcí spojité Hopfieldovy sítě získáme synaptické váhy, které během aktivního režimu neuronové sítě zajistí minimalizaci E. Váhovou hodnotu na spojení mezi neurony $U_{x\,i}$ a $U_{y\,j}$ dostaneme následovně:

$$w(x,i\,;y,j) = -A\delta_{x\,y}\left(1-\delta_{i\,j}\right) - B\delta_{i\,j}\left(1-\delta_{x\,y}\right) - C - D\,d_{x\,y}\left(\delta_{i,j+1} + \delta_{i,j-1}\right),$$

kde $\delta_{i\,j}$ je Kroneckerovo delta, tzn. $\delta_{i\,j} = 1$, jestliže $i = j$ a $\delta_{i\,j} = 0$, jestliže $i \neq j$. Každý neuron dále obdrží externí vstupní signál

$$I_{x\,i} = +C\,N,$$

kde N je parametr, jehož hodnota je větší než n (tj. počet měst na trase).

Popis algoritmu

Krok 0. Inicializovat aktivace všech neuronů sítě.

Inicializovat Δt malou hodnotou.

Krok 1. Pokud není splněna podmínka ukončení, opakovat kroky (2 až 6).

Krok 2. Provádět kroky (3 až 5) n^2-krát (n je počet měst).

Krok 3. Vybrat náhodně neuron.

Krok 4. Změnit hodnotu jeho vnitřní energie*:*

$$\begin{aligned} u_{x,i}(new) = u_{x,i}(old) \\ +\Delta t\Big[-u_{x,i}(old) \\ -A\sum_{j\neq i} v_{x,j} \\ -B\sum_{y\neq x} v_{y,i} \\ -C\Big(n-\sum_{x}\sum_{j} v_{x,j}\Big) \\ -D\sum_{y\neq x} d_{x,y}\left(v_{y,i+1}+v_{y,i-1}\right)\Big]. \end{aligned}$$

Krok 5. Vypočítat hodnotu na jeho výstupu *:*

$$v_{x,i} = 0.5\Big[1+\tanh(\alpha\ u_{x,i})\Big].$$

Krok 6. Test podmínky ukončení.

Vzhledem k tomu, že nalezené minimum energetické funkce nemusí odpovídat minimu globálnímu, nemusí být nalezené řešení problému obchodního cestujícího (v aktivním režimu spojité Hopfieldovy sítě) optimální a dokonce nemusí být v některých případech ani přípustné. Velice důležité je správné nastavení parametrů A, B, C, D, N a α. Vhodným nastavením těchto parametrů a více pokusy s různým počátečním nastavením sítě lze dosáhnout lepší aproximace optimálního řešení. Obecně však neexistuje návod na efektivní nastavení uvedených parametrů tak, aby neuronová síť konvergovala co nejlépe ke globálnímu minimu.

Korespondenční úkoly:

Vytvořte počítačový program pro řešení „problému obchodního cestujícího“ spojitou Hopfieldovou sítí. Optimalizujte s jeho použitím trasu mezi pěti městy (stanovte si sami vzdálenosti mezi jednotlivými městy).

DVOUSMĚRNÁ ASOCIATIVNÍ PAMĚŤ.

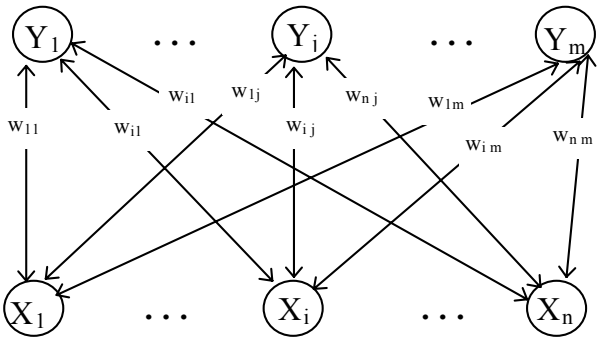
Dvousměrná asociativní paměť (BAM angl. *Bidirectional Associative Memory*) je variantou heteroasociativní rekurentní neuronové sítě. Autorem řady publikací toto téma je především B.Kosko a C.Guest.

Klíčová slova této kapitoly:

Dvousměrná asociativní paměť (BAM angl. Bidirectional Associative Memory).

Dvousměrná asociativní paměť si v adaptivním režimu zapamatuje množinu asociovaných vzorů jako sumaci bipolárních korelačních matic (typu *m, n* pro každý zapamatovaný vzor*)*. Struktura BAM je dána dvěmi vrstvami neuronů (vrstva *X* obsahuje *n* neuronů a vrstva *Y* obsahuje *m* neuronů), které jsou vzájemně úplně propojeny obousměrnými vazbami (viz obrázek 38). Jestliže váhová matice pro signál transportovaný vrstvou *X* do vrstvy *Y* je ***W***, pak váhová matice pro signál transportovaný vrstvou *Y* do vrstvy *X* je $\boldsymbol{W}^T$.

Aktivní režim BAM probíhá tak, že si neurony obou vrstev neustále posílají mezi sebou signál (tj. oběma směry), až všechny neurony dosáhnou rovnovážný stav (tj. aktivace se nemění během několika kroků). Existují tři základní varianty BAM - binární, bipolární a spojitá.



Obrázek 38: ***Dvousměrná asociativní paměť.***

Diskrétní BAM

Obě formy BAM (tj. binární i bipolární) jsou velmi příbuzné. V každé z nich lze váhové hodnoty nalézt ze sumace aktivačních hodnot neuronů odpovídajících si tréninkových párů. Aktivační funkce je skoková s možností nenulového prahu.

Vytvoření váhové matice:

váhová matice pro zapamatování množiny vstupních a odpovídajících výstupních vektorů ***s(t) : t(p),*** *p = 1, ..., P*, kde

$$\mathbf{s}(p) = \left(s_1(p), \ldots, s_i(p), \ldots, s_n(p)\right)$$

a

$$\mathbf{t}(p) = \left(t_1(p), \ldots, t_j(p), \ldots, t_m(p)\right),$$

může být determinována Hebbovým pravidlem pro asociované sítě. Zápis výsledných hodnot pro váhovou matici $\boldsymbol{W}$ závisí na tom, jestli je tréninkový vektor binární nebo bipolární. Pro binární vstupní vektory je váhová matice $\mathbf{W} = \{w_{ij}\}$ tvořená prvky, které jsou definovány následujícím vztahem:

$$w_{ij} = \sum_p \big(2s_i(p) - 1\big)\big(2t_j(p) - 1\big).$$

Pro bipolární vstupní vektor je váhová matice $\mathbf{W} = \{w_{ij}\}$ tvořená prvky, které jsou definovány následujícím vztahem:

$$w_{ij} = \sum_p s_i(p)\, t_j(p).$$

Aktivační funkce:

Aktivační funkcí pro diskrétní BAM je odpovídající skoková funkce, která závisí na kódování tréninkových vektorů.

Pro *binární* vstupní vektory má aktivační funkce pro vrstvu $\boldsymbol{Y}$ tvar:

$$y_j = \begin{cases} 1 & \text{pokud } y\_in_j > 0 \\ y_j & \text{pokud } y\_in_j = 0 \\ 0 & \text{pokud } y\_in_j < 0, \end{cases}$$

a aktivační funkce pro vrstvu $\boldsymbol{X}$ má tvar:

$$x_i = \begin{cases} 1 & \text{pokud } x\_in_i > 0 \\ x_i & \text{pokud } x\_in_i = 0 \\ 0 & \text{pokud } x\_in_i < 0. \end{cases}$$

Pro *bipolární* vstupní vektory má aktivační funkce pro vrstvu $\boldsymbol{Y}$ tvar:

$$y_j = \begin{cases} 1 & \text{pokud } y\_in_j > \theta_j \\ y_j & \text{pokud } y\_in_j = \theta_j \\ -1 & \text{pokud } y\_in_j < \theta_j, \end{cases}$$

a aktivační funkce pro vrstvu $\boldsymbol{X}$ má tvar:

$$x_i = \begin{cases} 1 & \text{pokud } x\_in_i > \theta_i \\ x_i & \text{pokud } x\_in_i = \theta_i \\ -1 & \text{pokud } x\_in_i < \theta_i. \end{cases}$$

Popis algoritmu

Krok 0. Inicializace vah, tj. zapamatování P vzorů.

Krok 1. Pro každý testovací vzor opakovat kroky (2 až 6).

Krok 2a. Inicializovat *vrstvu X* vnějším vstupním vektorem ***x***.
(tj. nastavit aktivace neuronů ve vrstvě *X* hodnotami vektoru ***x***)

Krok 2b. Inicializovat *vrstvu Y* vnějším vstupním vektorem ***y***.
(Jeden ze dvou vstupních vektorů musí být nulový vektor.)

Krok 3. Pokud aktivace neuronů nekonvergují, opakovat kroky (4 až 6).

Krok 4. Aktualizovat aktivace neuronů ve vrstvě *Y*.

Vypočítat vnitřní potenciál neuronu:

$$y\_in_j = \sum_i w_{ij} x_i .$$

Vypočítat aktivace neuronů

$$y_j = f\left(y\_in_j\right).$$

Transportovat signál vrstvě *X*.

Krok 5. Aktualizovat aktivace neuronů ve vrstvě *X*.

Vypočítat vnitřní potenciál neuronu:

$$x\_in_i = \sum_j w_{ij} y_j .$$

Vypočítat aktivace neuronů

$$x_i = f\left(x\_in_i\right).$$

Transportovat signál vrstvě *Y*.

Krok 6. *Test konvergence.*

Pokud aktivace vektorů ***x*** a ***y*** dosáhly rovnovážného stavu, pak stop; jinak pokračovat.

Příklad:

Načrtněme nyní možnosti použití diskrétní sítě BAM s bipolárním kódováním vektorů, která mapuje dva jednoduché znaky následujícím způsobem:

```
. # .          . # #
# . #          # . .
# # #          # . .
# . #          # . .
# . #          . # #
(-1. 1)          (1, 1)
```

Váhové matice jsou potom vyjádřeny takto:

(A → -1, 1)

$$\begin{bmatrix} 1 & -1 \\ -1 & 1 \\ 1 & -1 \\ -1 & 1 \\ 1 & -1 \\ -1 & 1 \\ -1 & 1 \\ -1 & 1 \\ -1 & 1 \\ -1 & 1 \\ 1 & -1 \\ -1 & 1 \\ -1 & 1 \\ 1 & -1 \\ -1 & 1 \end{bmatrix}$$

(C → 1, 1)

$$\begin{bmatrix} -1 & -1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ -1 & -1 \\ -1 & -1 \\ 1 & 1 \\ -1 & -1 \\ -1 & -1 \\ 1 & 1 \\ -1 & -1 \\ -1 & -1 \\ -1 & -1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$$

(***W***, zapamatování obou vzorů)

$$\begin{bmatrix} 0 & -2 \\ 0 & 2 \\ 2 & 0 \\ 0 & 2 \\ 0 & -2 \\ -2 & 0 \\ 0 & 2 \\ -2 & 0 \\ -2 & 0 \\ 0 & 2 \\ 0 & -2 \\ -2 & 0 \\ -2 & 0 \\ 1 & 0 \\ 0 & 2 \end{bmatrix}$$

Nyní ověříme správnost zapamatování obou vstupních vzorů:

VSTUPNÍ VZOR *A*
(-1 1 -1 1 -1 1 1 1 1 1 -1 1 1 -1 1) ***W*** = (-14, 16) → (-1, 1).

VSTUPNÍ VZOR *C*
(-1 1 1 1 -1 -1 1 -1 -1 1 -1 -1 -1 1 1) ***W*** = (14, 18) → (1, 1).

Dále ukážeme, že i vrstva *Y* může být použitá jako vstupní vrstva. Váhovou matici ***W*** musíme pro tento účel transponovat, tj.

$$\mathbf{W}^{\mathrm{T}} = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 & -2 & 0 & -2 & -2 & 0 & 0 & -2 & -2 & 2 & 0 \\ -2 & 2 & 0 & 2 & -2 & 0 & 2 & 0 & 0 & 2 & -2 & 0 & 0 & 0 & 2 \end{bmatrix}.$$

Pro vstupní vektor asociovaný se vzorem *A*, tj. (-1, 1) dostaneme:
(-1, 1) $\boldsymbol{W}^T$ =

$$(-1,1)\begin{bmatrix} 0 & 0 & 2 & 0 & 0 & -2 & 0 & -2 & -2 & 0 & 0 & -2 & -2 & 2 & 0 \\ -2 & 2 & 0 & 2 & -2 & 0 & 2 & 0 & 0 & 2 & -2 & 0 & 0 & 0 & 2 \end{bmatrix}$$

$$= (-2 \;\; 2 \;\; -2 \;\; 2 \;\; -2 \;\; 2 \;\; 2 \;\; 2 \;\; 2 \;\; 2 \;\; -2 \;\; 2 \;\; 2 \;\; -2 \;\; 2)$$

$$\rightarrow (-1 \;\; 1 \;\; -1 \;\; 1 \;\; -1 \;\; 1 \;\; 1 \;\; 1 \;\; 1 \;\; 1 \;\; -1 \;\; 1 \;\; 1 \;\; -1 \;\; 1),$$

což je vzor *A*.

Pro vstupní vektor asociovaný se vzorem *C*, tj. (1, 1) dostaneme:
(1, 1) $\boldsymbol{W}^T$ =

$$(1,1)\begin{bmatrix} 0 & 0 & 2 & 0 & 0 & -2 & 0 & -2 & -2 & 0 & 0 & -2 & -2 & 2 & 0 \\ -2 & 2 & 0 & 2 & -2 & 0 & 2 & 0 & 0 & 2 & -2 & 0 & 0 & 0 & 2 \end{bmatrix}$$

$$= (-2 \;\; 2 \;\; 2 \;\; 2 \;\; -2 \;\; -2 \;\; 2 \;\; -2 \;\; -2 \;\; 2 \;\; -2 \;\; -2 \;\; -2 \;\; 2 \;\; 2)$$

$$\rightarrow (-1 \;\; 1 \;\; 1 \;\; 1 \;\; -1 \;\; -1 \;\; 1 \;\; -1 \;\; -1 \;\; 1 \;\; -1 \;\; -1 \;\; -1 \;\; 1 \;\; 1),$$

což je vzor *C*.

Informace o obou vzorech jsou uloženy ve váhové matici ***W***. Síť je tedy ukončila fázi adaptace a je připravena k aktivní fázi.

Spojitá BAM

Spojitá dvousměrná asociativní paměť transformuje použitím sigmoidní aktivační funkce hladký a spojitý vstupní signál z intervalu [0,1] na signál výstupní.

Pro binární vstupní vektory ***s(t) : t(p),*** *p = 1, ..., P,* jsou váhové hodnoty determinovány již dříve uvedeným vztahem

$$w_{ij} = \sum_p \left(2s_i(p) - 1\right)\left(2t_j(p) - 1\right).$$

Aktivační funkce je logistická sigmoida

$$f(x) = \frac{1}{1 + \exp\left(-y\_in_j\right)},$$

kde

$$y\_in_j = b_j + \sum_i x_i w_{ij} \ .$$

Analogické vztahy lze odvodit i pro neurony ve vrstvě *X*.

Kapacita paměti BAM

Kapacita paměti BAM je velmi omezená. Kosko odhadl, že její maximální velikost je *min(n, m)*, kde *n* je počet neuronů ve vrstvě *X* a *m* je počet neuronů ve vrstvě *Y*. Snahou současného výzkumu a vývoje je zdokonalit BAM tak, aby toto omezení bylo redukováno na maximální možnou míru a aby bylo možné plně využít tak možnosti modelu neuronové sítě.

Úkoly:

Použijte diskrétní síť BAM, která mapuje tři jednoduché znaky n8sleduj9c9m způsobem:

```
. # .          # . #          # # #
# # #          . # .          # . #
. # .          # . #          # # #
```

(--1 1 –1 1 1 1 –1 1 –1) *(1 –1 1 –1 1 –1 1 –1 1)* *(1 1 1 1 –1 1 1 1 1).*

Korespondenční úkoly:

Vytvořte počítačový program pro realizaci adaptačního algoritmu diskrétní sítě BAM.

POSTAVENÍ NEURONOVÝCH SÍTÍ V INFORMATICE.

V této závěrečné kapitole se postupně zamyslíme nad třemi tématy: (1) *neuronové sítě a von neumanovská architektura počítače*; (2) *aplikace neuronových sítí* a (3) *implementace neuronových sítí a neuropočítače.*

Klíčová slova této kapitoly:
NETtalk, *neuropočítače*, *netware.*

Neuronové sítě a von neumanovská architektura počítače [6]

V jistém smyslu neuronové sítě představují univerzální výpočetní prostředek, a tedy mají stejnou výpočetní sílu jako klasické počítače např. von neumannovské architektury (tj. pomocí neuronových sítí lze principiálně spočítat vše, co umí např. osobní počítač a naopak). Tato jejich vlastnost by vzhledem k existenci stovek různých univerzálních výpočetních modelů nebyla tak výjimečná. Navíc je funkce popsána velkým počtem váhových parametrů a vůbec není zřejmé, jak bychom požadovanou funkci v tomto výpočetním modelu naprogramovali.

Hlavní výhodou a zároveň odlišností neuronových sítí od klasické von neumannovské architektury je jejich schopnost učit se. Požadovanou funkci sítě neprogramujeme tak, že bychom popsali přesný postup výpočtu její funkční hodnoty, ale síť sama abstrahuje a zobecňuje charakter funkce v adaptivním režimu procesu učení ze vzorových příkladů. V tomto smyslu neuronová síť připomíná inteligenci člověka, který získává mnohé své znalosti a dovednosti ze zkušenosti, kterou ani není ve většině případů schopen formulovat analyticky podle příslušných pravidel či algoritmu. V následujícím výkladu uvedu několik motivačních (trochu nadnesených) příkladů, které nám pomohou tento fenomén pochopit.

Představme si zedníka, který by chtěl svého učně naučil omítat zeď. Pravděpodobně ti, kdo se někdy sami pokoušeli omítnout svůj dům, ví že první pokusy nebývají moc zdvořilé (polovina malty většinou končí no zemi.),Jak groteskní by bylo teoretické školení zedníka, který by na tabuli vylekanému učni napsal diferenciální rovnice popisující trajektorii (dráhu) a rychlost pohybující se ruky, popř. zápěstí, při nahazování malty na omítanou zeď. I kdyby zednický učeň měl základy v diferenciálním počtu, omítat by se tímto způsobem nenaučil. Tuto dovednost může totiž učeň získat jen pozorováním zedníka při omítání a vlastními pokusy korigovanými učitelem.

Dalším demonstračním příkladem popsaným v literatuře je balancování tyče na koštěti. Byla sestrojena neuronová síť, která dokázala napodobit dovednost cirkusového klauna, který na svém nose drží koště ve vertikální poloze. Při vlastním experimentu byl použit speciální vozík, na kterém bylo koště volně upevněno (pro jednoduchost v jedné rovině) tak, že by bez zachycení spadlo. Neuronová síť se učila nejprve na základě odchylky (úhlu) koštěte od vertikální polohy a později od filtrovaného obrazu násady koštěte snímaného kamerou určit posuv vozíku (v jedné přímé dráze) tak, aby koště nespadlo. Tréninkové vzory pro její adaptaci, kde vstup odpovídal filtrovanému obrazu koštěte a výstup posuvu vozíku, byly získány od demonstrátora (při zpomalené počítačové simulaci), který nějaký čas pohyboval vozíkem tak, aby koště nespadlo. Po čase neuronová síť sama úspěšně převzala jeho úlohu řízení (již skutečného) vozíku. I zde by bylo možné teoreticky sestavit diferenciální rovnice pro pohyb vozíku, ale než by je klasický počítač von neumannovské architektury vyřešil, koště by pravděpodobně spadlo. Na druhou stranu v tomto jednoduchém demonstračním příkladě (varianta se vstupním úhlem) existuje úspěšný řídící systém založený na klasické teorii řízení.

Podobným příkladem popsaným v literatuře je řízení přítoku látek potřebných ve složitém výrobním procesu, kde je prakticky nemožné sestavit analytický model. V praxi byla tato činnost prováděna zkušeným pracovníkem, který na základě informací z různých měřidel reguloval pomocí pák přítok jednotlivých látek. Uvedený pracovník není schopen vyjádřit prostřednictvím přesných pravidel pohyb s regulačními pákami. I zde byla zapojena neuronová síť, která se na základě příkladů stavů měřidel a odpovídajících reakcí pracovníka sama po nějakém čase naučila regulovat přítok látek.

Z uvedených příkladů vyplývá, že neuronová síť modeluje schopnost člověka učit se z příkladů dovednosti či znalosti, které není schopen řešit algoritmicky pomocí klasických počítačů von neumannovské architektury, protože chybí analytický popis nebo jejich analýza je příliš složitá. Tomu potom odpovídají oblasti aplikace neuronových sítí (viz dále), kde klasické počítače selhávají. Zřejmě si také nestačí pamatovat všechny vzorové příklady (tréninkovou množinu) nazpaměť (např. v tabulce uložené v paměti klasického počítače). navíc je potřeba generalizovat (zobecňovat)jejich zákonitosti, které by umožnily řešit podobné příklady, s nimiž se neuronová síť při učení ještě nesetkala. Např. v případě rozpoznávání písmen si není možné pamatovat všechny možné tvary obrazu jednotlivých písmen.

Dalším ilustračním příkladem důležitosti generalizační schopnosti lidské inteligence, je příprava studenta na zkoušku z matematiky. Je zřejmé, že naučení všech vzorových příkladů ve sbírce nazpaměť bez náležitého pochopení postupů řešení nezaručuje úspěšné složení zkoušky. Student pravděpodobně u zkoušky neuspěje, pokud nedostane identický příklad ze sbírky, ale bude mu zadána úloha jen s podobným postupem řešení. Nestačí se totiž nazpaměť naučit vzorové příklady, ale je potřeba umět zobecnit zákonitosti jejich řešení.

Schopnost učit se zobecňovat je typickou vlastností lidské inteligence. Velkým problémem pro hodnocení generalizační schopnosti neuronové sítě je, že není jasné, jakým způsobem definovat, co je správná generalizace. Uvažujme otázku z testu inteligence, kdy se má doplnit další člen posloupnosti 1, 2, 3, ... Většina lidí by asi doplnila následující číslo 4. Představme si , ale matematika, který si všimne, že číslo 3 je součtem dvou předcházejících čísel 1 a 2, a dle této komplikovanější souvislosti doplní místo čísla 4 číslo 5, které je opět součtem dvou předcházejících čísel 2 a 3. Kromě toho, že bude některými „normálními“ lidmi považován za podivína, není vůbec zřejmé, které ze dvou uvedených doplnění je správnou generalizací zákonitosti této posloupnosti. A takových doplnění, které je možné nějakým způsobem zdůvodnit, existuje jistě nekonečně mnoho.

Díky tomu, že neumíme definovat (formalizovat), a tedy ani měřit generalizační schopnosti neuronových sítí, chybí základní kritérium, které by rozhodlo, jaké modely neuronových sítí jsou v konkrétním případě dobré, či lepší než jiné apod. Generalizační schopnosti navržených modelů neuronových sítí se většinou ilustrují na jednotlivých příkladech, které (možná díky vhodnému výběru) vykazují dobré vlastnosti, ale tyto vlastnosti nelze nijak formálně ověřit (dokázat). Tento stav je také příčinou krize základního výzkumu neuronových sítí.

Na druhou stranu úspěšné aplikace neuronových sítí při řešení důležitých praktických úloh, kde klasické počítače neuspěly, i to, že simulace (velmi zjednodušených modelů) biologických neuronových sítí vykazují prvky podobné lidské inteligenci, naznačují, že tyto modely vystihují určité rysy, důležité pro napodobení inteligentních činností člověka, které počítače von neumannovské architektury postrádají. Základním rysem biologických nervových systémů je hustě propojená síť velkého počtu výpočetních prvků (neuronů), které samy počítají jen jednoduché funkce, což v případě matematických modelů neuronových sítí pravděpodobně vytváří výpočetní paradigma postačující k napodobení inteligentního chování.

Systematická logika a přesnost klasických počítačů je u neuronových sítí nahrazena asociací s neurčitostí, kdy se k novému problému „vybaví“sdružený (podobný) vzorový příklad (tréninkový vzor), ze kterého je zobecněno jeho řešení. Také místo explicitní reprezentace dat v paměti klasických počítačů jsou informace v neuronových sítích zakódovány implicitně a jednotlivým číselným parametrům sítě (kromě vstupů a výstupů) není přiřazen přesný význam. Zatímco klasické počítače jsou citlivé na chybu a změna jednoho bitu může znamenat celkový výpadek systému, neuronové sítě jsou robustní. Je například známo, že po neurochirurgických operacích kdy je pacientovi odebrána část tkáně mozkové kůry, pacient přechodně zapomíná některé funkce (např. schopnost mluvit) nebo u nich ztrácí určitou obratnost (např. koktá), ale brzy se znovu tyto schopnosti obnoví, či zdokonalí, protože jiné neurony převezmou funkci těch původních. Tento jev lze pozorovat i u modelů neuronových sítí, kdy odebráním několika málo neuronů nemusí síť nutně ztratit svou funkčnost, ale způsobí to třeba jen menší nepřesnost výsledných odpovědí. Dále u klasických počítačů von neumannovské architektury je sekvenční běh programu lokalizován např. pomocí čítače instrukcí. V neuronových sítích je naopak výpočet distribuován po celé síti a je přirozeně paralelní.

Při srovnávání modelů neuronových sítí s klasickou von neumannovskou architekturou počítače je možné vypozorovat střet dvou inteligencí: biologické a křemíkové. Východiskem, které může nalézt v dnešním přetechnizovaném světě širší uplatnění, je symbióza obou přístupů. Myšlenka vytvořit počítač ke svému obrazu nabývá v poslední době konkrétnější podoby.

Aplikace neuronových sítí [6]

Porovnání modelů neuronových sítí s počítači von neumannovské architektury naznačuje možné oblasti jejich aplikace tam, kde klasické počítače selhávají. Jedná se především o praktické problémy, u kterých není znám algoritmus nebo jejich analytický popis je pro počítačové zpracování příliš komplikovaný. Typicky se

neuronové sítě dají použít tam, kde jsou k dispozici příkladová data, která dostatečně pokrývají problémovou oblast. Výhody neuronových sítí oproti klasickým počítačům samozřejmě neznamenají, že by neuronové sítě mohly nahradit současné počítače, protože v případě mechanických výpočtů (např. násobení), které lze jednoduše algoritmicky popsat, nemohou (stejně jako lidé) v rychlosti a přesnosti klasickým počítačům konkurovat. Neuronové sítě ve formě specifických modulů pravděpodobně jen obohatí von neumannovské architektury. V následujícím výkladu uvedeme několik možných oblastí aplikace neuronových sítí.

Neuronové sítě lze přirozeným způsobem použít k *rozpoznávání obrazců*, např. rozpoznávání nascanovaných, psaných resp. tištěných znaků (číslic, písmen apod.). Obraz jednoho znaku nejprve odseparujeme od okolního textu (např. se určí krajní body obrazu) a potom se znormuje, tj. zobrazí do standardizované matice (např. $15 \times 10 = 150$) bodů. Jednotlivé body pak odpovídají vstupům neuronové sítě, které jsou např. aktivní. právě když čára v obrazu právě zasahuje příslušné body. Každý výstupní neuron v síti představuje možný znak, který je rozpoznán, právě když je tento neuron aktivní. Tréninkovou množinu lze např. vytvořit přepsáním nějakého textu, který je již k dispozici v počítači (odpovídá požadovaným výstupům tréninkových vzorů, tj. identifikovaným znakům), takovým způsobem (např. rukou), pro který budeme neuronovou síť k rozpoznávání potřebovat (představuje odpovídající příklady obrazových vstupů tréninkových vzorů). Neuronovou síť pak lze pomocí této množiny učit tak dlouho, dokud není sama schopna rozpoznávat příslušné znaky. Tímto postupem můžeme v relativně krátké době docílit spolehlivosti např. 95% správně rozpoznaných znaků. Podobný postup lze využít např. v robotice pro zpracování vizuálních informací či při vyhodnocování družicových snímků apod.

Další možnou oblastí aplikace neuronových sítí je *řízení* složitých zařízení v dynamicky se měnících podmínkách. V minulé kapitole jsme uvedli dva motivační příklady z této oblasti: balancování koštěte a regulace přítoku látek ve složitém výrobním procesu. Dalším demonstračním příkladem řídícího systému popsaného v literatuře je autopilot automobilu, který se v počítačové simulaci pohybuje na dvouproudé dálnici spolu s auty jedoucími stejným směrem. Auto řízené neuronovou sítí určovalo na základě vzdálenosti a rychlosti nejbližších aut v obou pruzích svou vlastní rychlost a změnu pruhu. Dále neuronová síť ovládala volant podle zakřivení dálnice, polohy auta v pruhu a aktuálního úhlu volantu. Je zajímavé, že neuronová síť se kromě úspěšného řízení vozidla (bez kolizí) včetně předjíždění naučila i různé zvyky a styl jízdy (např. riskantní rychlá jízda a časté předjíždění nebo naopak opatrná pomalá jízda) podle řidičů - trenérů, od kterých byly získány tréninkové vzory.

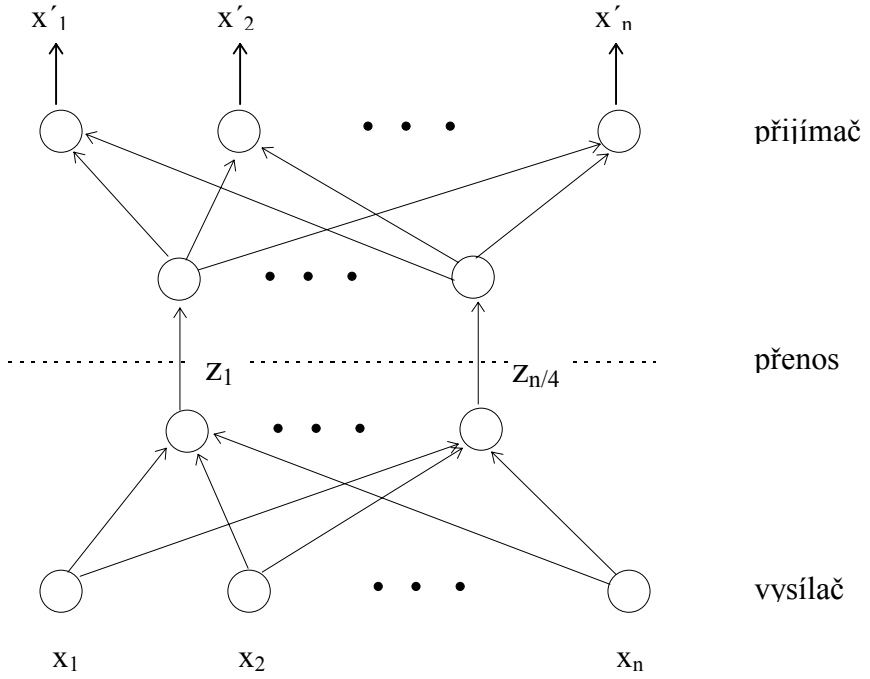
Jinou důležitou aplikační oblastí neuronových sítí je *predikce* a příp. následné *rozhodování*. Typickými příklady z této oblasti jsou předpověď počasí, vývoj cen akcií na burze, spotřeba elektrické energie apod. Např. při meteorologické předpovědi jsou vstupem neuronové sítě odečty základních parametrů (např. teplota, tlak apod.) v čase a učitelem je skutečný vývoj počasí v následujícím období. Uvádí se, že u předpovědi počasí v rozpětí několika dnů byla síť úspěšnější než meteorologové.

Jiným příkladem uplatnění neuronových sítí je *analýza signálů* jako např. EKG, EEG apod. Spojitý signál je vzorkován ve stejných časových intervalech a několik posledních diskrétních hodnot úrovně signálu slouží jako vstup do např. dvouvrstvé neuronové sítě. Naučená neuronová síť je schopna identifikovat specifický tvar signálu, který je důležitý pro diagnostiku. Např. neuronová síť s topologií 40 - 17 - 1 byla použita pro klasifikaci EEG signálů se specifickými α-rytmy.

Další oblastí aplikace neuronových sítí je *transformace signálů*, jehož příkladem je systém NETtalk, určený pro převod anglicky psaného textu na mluvený signál. Tento systém je založen na neuronové síti s topologií 203 - 80 - 26 s 7×29 vstupními neurony pro zakódování kontextu 7 písmen psaného textu každému z 26 písmen anglické abecedy a čárce, tečce a mezeře odpovídá jeden neuron, který je při jejich výskytu aktivní), 80 skrytými neurony v mezilehlé vrstvě 26 výstupními neurony reprezentují fonény odpovídajícího mluveného signálu. Funkce sítě je následující: vstupní text se postupně přesouvá u vstupních neuronů po jednom písmenu zprava doleva a přitom je aktivní právě ten výstupní neuron, který reprezentuje fonén odpovídající prostřednímu ze 7 písmen vstupního textu. V našem příkladě se čte prostřední písmeno „C“ v anglickém slově „CONCEPT“ s výslovností [´konsept], kterému odpovídá fonén [s]. Stejné písmeno „C“ na začátku tohoto slova však v daném kontextu odpovídá fonénu [k]. Úspěšná implementace systému NETtalk vedla ke snaze vytvořit systém založený na neuronové síti s obrácenou funkcí, která by převáděla mluvený jazyk do psané formy (tzv. fonetický psací stroj).

Další možností využití neuronových sítí je *komprese dat* např. pro přenos televizního signálu, telekomunikaci apod. Pro tento účel byla vyvinuta technika použití neuronové sítě se dvěma vnitřními vrstvami a s topologií *n - n/4 - n/4 - n* (tj. *n* neuronů ve vstupní vrstvě, *n/4* neuronů ve vnitřních vrstvách a n neuronů ve výstupní vrstvě). Počet neuronů ve vnitřních vrstvách je výrazně menší než je počet neuronů ve vstupní a výstupní vrstvě. Počet neuronů ve vstupní i výstupní vrstvě je stejný, protože obě vrstvy reprezentují stejný obrazový signál. Tato neuronová síť se učí různé obrazové vzory tak, že vstup i výstup tréninkových vzorů představují totožný obraz. Síť tak pro daný obrazový vstup odpovídá přibližně stejným výstupem. Při vlastním přenosu je pro daný obrazový signál $x_1 \ ..., x_n$ u vysílače nejprve vypočten stav skrytých neuronů $z_1, ..., z_{n/4}$ a takto kompresovaný obraz je přenášen informačním kanálem k příjemci, který jej dekóduje výpočtem stavů výstupních neuronů $x´_1, ..., x´_n$. Tímto způsobem je získán téměř původní obraz. Při vlastním experimentu se

ukázalo, že kvalita přenosu (srovnatelná s jinými způsoby komprese dat) závisí na tom, zda jsou přenášené obrazy podobné tréninkovým vzorům, na které se síť adaptovala.



Obrázek 39: Komprese při přenosu signálu pomocí neuronové sítě s topologií *n - n/4 - n/4 – n*.

Posledním oborem aplikace neuronových sítí, který zde uvedeme, jsou *expertní systémy.* Velkým problémem klasických expertních systémů založených na pravidlech je vytvoření báze znalostí, která bývá časově velmi náročnou záležitostí s nejistým výsledkem. Neuronové sítě představují alternativní řešení, kde reprezentace znalostí v bázi vzniká učením z příkladových inferencí. V tomto případě aktivní režim neuronové sítě zastupuje funkci inferenčního stroje. Na druhou stranu implicitní reprezentace znalostí neumožňuje pracovat s neúplnou informací a neposkytuje zdůvodnění závěrů, což jsou vlastnosti, bez kterých se prakticky použitelný expertní systém neobejde. Tento problém částečně řeší univerzální neuronový expertní systém EXPSYS, který obohacuje vícevrstvou neuronovou síť o intervalovou aritmetiku pro práci s nepřesnou informací a o heuristiku analyzující síť, která umožňuje jednoduché vysvětlení závěrů. Systém EXPSYS byl úspěšně aplikován v energetice a medicíně. Např. v lékařské aplikaci jsou zakódované příznaky onemocnění a výsledky různých vyšetření vstupem do neuronové sítě a diagnózy, popř. doporučená léčba jsou jejím výstupem. Tréninkovou množinu lze získat z kartotéky pacientů.

Implementace neuronových sítí a neuropočítače [6]

Odlišná architektura neuronových sítí vyžaduje speciální hardwarovou realizaci. V této souvislosti hovoříme o tzv. *neuropočítačích*. Avšak vzhledem k rozšířenosti klasických počítačů a kvůli problémům spojeným s hardwarovou realizací neuronových sítí zatím nejjednodušší implementací neuronových sítí, se kterou se nejčastěji (zvláště v České republice) setkáváme, je tzv. *netware*, což je software pro klasické počítače (např. PC), který modeluje práci neuronové sítě. Jedná se většinou o demonstrační programy s efektním uživatelským interfacem, které simulují práci nejznámějších modelů neuronových sítí na jednoduchých příkladech. V některých již dokonalejších programech je možné zadat vlastní aktivní i adaptivní dynamiku, což umožňuje relativně rychle přizpůsobit model neuronové sítě danému praktickému problému nebo ověřit použitelnost navrženého nového modelu. Existují i programovací jazyky (a jejich překladače) pro klasické počítače, které podporují programovou implementaci neuronových algoritmů. Příkladem takového programovacího jazyka je AXON, který je podobný jazyku C. Dokonalejší netware většinou podporuje využití specializovaných koprocesorů (které je možno např. připojit k PC), které efektivně implementují neuronové funkce a urychlují časově náročné učení.

Vlastní neuropočítače většinou nepracují samostatně, ale jsou napojeny na klasické počítače, které mohou realizovat např. uživatelský interface. To je dáno především tím, že neuropočítače nejsou používány jako univerzální počítače, ale převážně fungují jako specializovaná zařízení pro řešení specifických úloh, Malé neuropočítače jsou spojeny přímo se sběrnicí klasického počítače a větší se mohou uplatnit jako servery na lokální síti.Podle způsobu aktualizace parametrů neuronové sítě rozdělujeme neuropočítače na *spojité* a *diskrétní* a podle typu reprezentace těchto číselných parametrů máme *analogové, digitální,* resp. *hybridní* (kombinace analogových a digitálních) neuropočítače. Zřídkakdy jeden neuron v implementované síti odpovídá jednomu procesoru neuropočítače (tzv. *plně implementované* neuropočítače), což se využívá pro velmi rychlé výpočty v reálném čase. Většinou se konstruují tzv. *virtuální* neuropočítače, kde jeden procesor vykonává práci stovek i tisíců neuronů části implementované neuronové sítě.

Z hlediska technologie je většina neuropočítačů založena na klasické mikroelektronice (např. VLSI technologie), kde neurony odpovídají hradlům (např. speciálním tranzistorům) a váhy synaptických spojů jsou reprezentovány rezistorovými vazbami. Tento přístup však s sebou přináší technické problémy jako je velká hustota propojení neuronů (roste řádově kvadraticky vzhledem k počtu neuronů) nebo adaptovatelnost vah u všech těchto spojů. Proto adaptivní režim neuronové sítě je někdy předem realizován odděleně pomocí dostupného netwaru na klasickém počítači a výsledná konfigurace sítě je napevno zapojena do příslušného obvodu neuropočítače. Také se stále více uplatňuje optoelektronika a dlouhodobější výhledy počítají s úplně odlišnými technologiemi, jako např. molekulární elektronika, hybridní biočipy apod.

Korespondenční úkoly:

Vypracujte seminární práci na téma „Použití neuronových sítí v …“ (oblast použití si zvolte sami). Informace hledejte především na www-stránkách.

Literatura:

[1] Beale, R. - Jackson, T.: Neural Computing: An Introduction. J W Arrowsmith Ltd, Bristol, Greit Britain 1992.

[2] Fausett, L. V.: Fundamentals of Neural Networks. Prentice-Hall, Inc., Englewood Cliffs, New Jersey 1994.

[3] Herz, J. - Krogh, A. - Palmer, R. G.: Introduction to the Theory of Neural Computation. Addison Wesley Publishing Company, Redwood City 1991.

[4] Kvasnička, V. - Beňušková, L. - Pospíchal, J. - Farkaš, I. - Tiňo, P. - Kráľ, A.:Úvod do teórie neurónových sietí. IRIS, Bratislava 1997.

[5] Novák, M.: Neuronové sítě a neuropočítače. Výběr, Praha 1992.

[6] Šíma, J. - Neruda, J.: Teoretické otázky neuronových sítí. Matfyzpress, Praha 1996.

[7] Vondrák, I.: Umělá inteligence a neuronové sítě. Skripta VŠB-TU, Ostrava 1995.

[8] Volná, E.: Neuronové sítě a genetické algoritmy. Skripta Ostravské univerzity, Ostrava 1998.