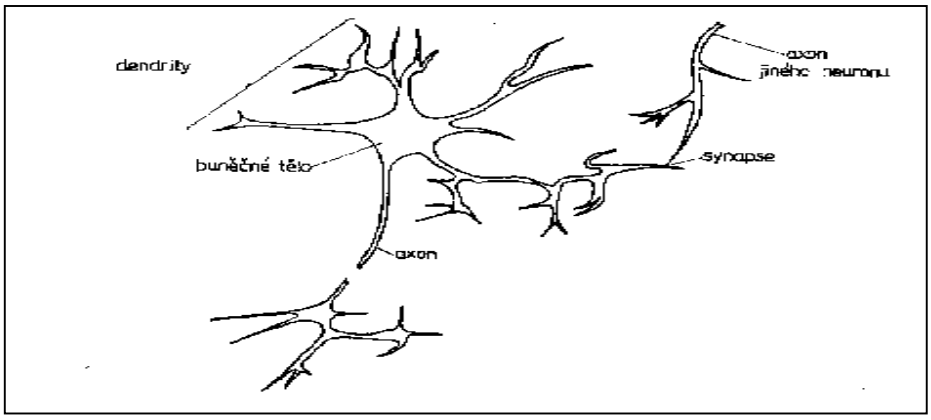
5.4 Neuronové sítě

Lidský mozek je složen asi z 10^{10} nervových buněk (neuronů) které jsou mezi sebou navzájem propojeny ještě řádově vyšším počtem vazeb [Novák a kol.,1992]. Začněme tedy nejdříve jedním neuronem. Na Obr. 1 je ukázka typického neuronu, který je tvořen tělem (soma) ze kterého vybíhá řada (desítky až stovky) kratších výběžků (dendrity) a jeden dlouhý výběžek (axon). Zatímco tělo neuronu je veliké řádově mikrometry a dendrity několik milimetrů, axon může dosahovat délky desítek centimetrů až jednoho metru. Axon každého neuronu je zakončen tzv. synapsí (typicky jedinou, ale může jich být i několik), která dosedá na jiný neuron. Přes synapse se přenášejí vzruchy mezi neurony; v důsledku chemických reakcí se v místě, kde synapse dosedá na neuron mění propustnost buněčné membrány neuronu, tím se lokálně mění koncentrace kladných i záporných iontů vně a uvnitř buňky a tedy i membránový potenciál. Některé synaptické vazby mají charakter excitační (zvyšují membránový potenciál), jiné mají charakter inhibiční (snižují membránový potenciál). Dílčí účinky synaptických vazeb se na neuronu kumulují a ve chvíli, kdy celkový membránový potenciál přesáhne určitý práh, neuron je aktivován a přes svoji synapsi začne působit na další neurony, se kterými je spojen.



Obr. 1 Biologický neuron

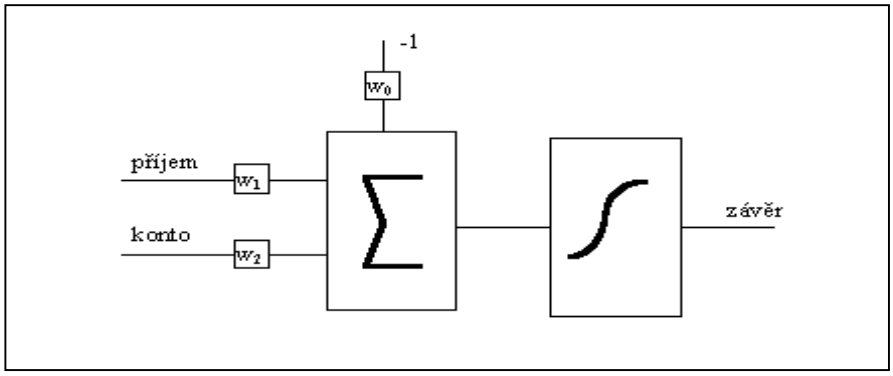
5.4.1 Model jednoho neuronu

Neuron tedy, zjednodušeně řečeno, přijímá kladné a záporné podněty od jiných neuronů a ve chvíli, kdy souhrn těchto podnětů překročí daný práh, sám se aktivuje. Výstupní hodnota neuronu je obvykle nějakou nelineární transformací souhrnu podnětů (viz Obr. 2). Z tohoto pohledu vycházejí matematické modely neuronu. Prvním byl tzv. „logický neuron“ McCullocha a Pittse z r. 1943, který pracoval se vstupními a výstupními hodnotami pouze 0 a 1. Dalším známým modelem byl Widrowův „adaptivní lineární neuron“ *Adaline* z roku 1960, který si popíšeme trochu podrobněji. Vstupem do *Adaline* jsou podněty (numerické hodnoty) označené $x_1,x_2,\ldots,x_m$. Každý podnět x_i je násoben vahou w_i; tento součin vstupuje do součtového členu, kde je vytvořen vážený součet[1]

$$\text{SUM} = \mathbf{w} \cdot \mathbf{x} = \sum_{i=1}^{m} w_i x_i$$

V případě, že tento vážený součet přesáhne práh w_0 je vystup $\hat{y}$ z *Adaline* roven 1, v opačném případě je výstup z *Adaline* roven 0 (někdy místo 0 může být hodnota -1). Na rozdíl od logického neuronu jsou tedy vstupy libovolná čísla, výstupy zůstávají dvouhodnotové:

[1] Zápis $\mathbf{w} \cdot \mathbf{x}$ vyjadřuje skalární součin.



Obr. 2 Schéma neuronu

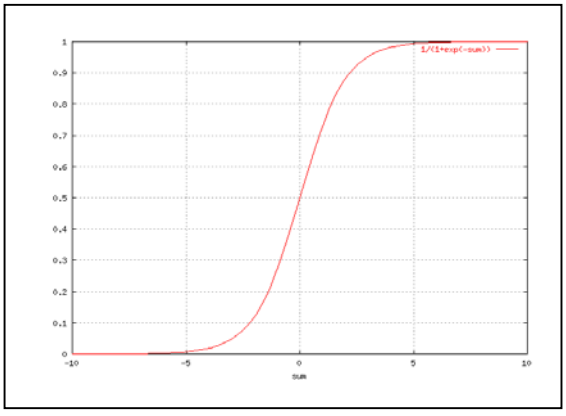
$$\hat{y} = 1 \quad \text{pro} \sum_{i=1}^{m} w_i x_i \geq w_0$$

$$\hat{y} = 0 \quad \text{pro} \sum_{i=1}^{m} w_i x_i < w_0$$

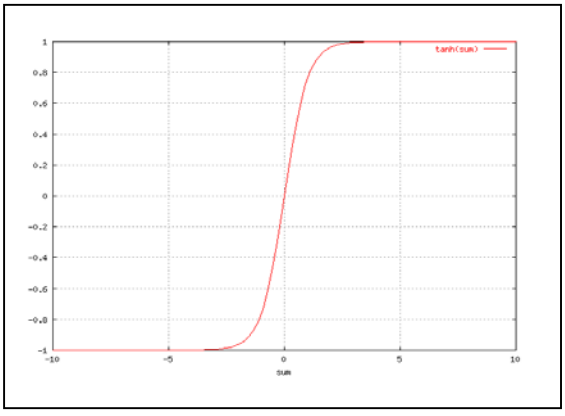
Nelinearita v podobě skokové funkce, tak jak je použita v případě *Adaline* (dávající pouze dvě výstupní hodnoty neuronu), je ve složitějších modelech neuronu nahrazena hladkými funkcemi, které nabývají hodnot z celého intervalu[2]. Nejpoužívanějšími přenosovými (aktivačními) funkcemi jsou:

- sigmoidální funkce $f(SUM) = \frac{1}{1 + e^{-SUM}}$; v tomto případě výstup neuronu $\hat{y}$ nabývá hodnot z intervalu $[0, 1]$ (Obr. 3),
- hyperbolický tangens $f(SUM) = \tanh(SUM)$; v tomto případě výstup neuronu $\hat{y}$ nabývá hodnot z intervalu $[-1, 1]$ (Obr. 4).

Někdy naopak nelineární funkce chybí; resp. $f(SUM) = SUM$. V tomto případě neuron realizuje pouze vážený součet vstupů [3].



Obr. 3 Sigmoida



Obr. 4 Hyperbolický tangens

[2] Požadavek na to, aby funkce byla hladká, je důležitý pro algoritmus nastavování vah v procesu učení. Hladká funkce je totiž diferencovatelná.

[3] Tento typ neuronů se někdy používá ve výstupní vrstvě neuronových sítí.

Činnost *Adaline* má jednoduchou geometrickou interpretaci. Jednotlivé vstupní podněty $x_1,x_2,...,x_m$ mohou představovat hodnoty vstupních atributů nějakého objektu (např. teplota, výška, váha ...). Každý objekt lze pak reprezentovat jako bod $\boldsymbol{x} = x_1,x_2,...,x_m$ v m-rozměrném prostoru. Bod $\boldsymbol{x}$ leží v jedné ze dvou částí prostoru oddělených od sebe rozdělující nadrovinou (pro $m = 2$ se pohybujeme v rovině a rozdělující nadrovina je přímka). Body ležící v jedné části prostoru můžeme považovat za obrazy objektů patřících do téže třídy. *Adaline* lze tedy považovat za *lineární klasifikátor* objektů do *dvou tříd*.

Pro ilustraci opět použijeme náš bankovní příklad, a sice v podobě dvou numerických atributů *konto* a *příjem* (Tab. 1). Adaptivní lineární neuron z Obr. 2 bude mít pro tato data váhy např. [4, 5]:

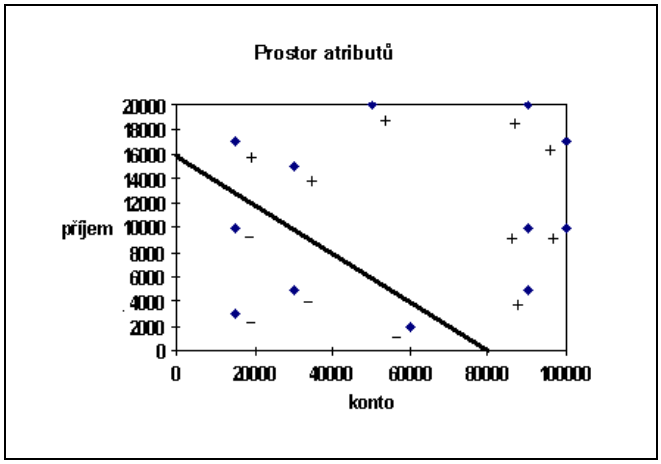
$$w_1 = 1$$
$$w_2 = 0.2$$
$$w_0 = 16000$$

Schopnost neuronu s takto nastavenými vahami klasifikovat naše data ukazuje Obr. 5. Rozdělovací přímka je definována rovnicí

$$\text{příjem} + 0.2\ \text{konto} - 16000 = 0.$$

Klient	příjem	konto	úvěr
K101	3000	15000	ne
K102	10000	15000	ne
K103	17000	15000	ano
K104	5000	30000	ne
K105	15000	30000	ano
K106	20000	50000	ano
K107	2000	60000	ne
K108	5000	90000	ano
K109	10000	90000	ano
K110	20000	90000	ano
K111	10000	100000	ano
K112	17000	100000	ano

Tab. 1 Numerická data



Obr. 5 Adaline jako lineární klasifikátor do dvou tříd

[4] Například zde znamená, že všechny váhy mohou být násobeny libovolným (stejným) číslem.

[5] Podobný výsledek bychom získali pro logistickou regresi.

Často se provádí normalizace vstupních hodnot (např. na interval [-1,1]). Tato normalizace snižuje rozdíly mezi váhami. Pokud pro naše data použijeme normalizované vstupy

norm_příjem = příjem/10000 - 1
norm_konto = konto/50000 - 1

Získáme tyto váhy neuronu [6]:

$w_1 = 5.8029$
$w_2 = 4.3746$
$w_0 = -2.8733$

Důležitou vlastností neuronů je jejich schopnost učit se. Učením se zde myslí (algoritmus) nastavení vah $\boldsymbol{w}$ na základě předložených příkladů $[\boldsymbol{x_i}, y_i]$ tak, aby systém co nejsprávněji zpracovával (např. klasifikoval) i neznámé příklady $\boldsymbol{x_k}$[7].

Mezi první způsoby učení patří *Hebbův zákon* z roku 1949. Byl formulován jako model učení na úrovni neuronů v mozku. Vychází z představy, že se posilují ty vazby které u daného neuronu způsobují jeho aktivaci. V (umělých) neuronech lze toto pravidlo formulovat takto:

$$\mathbf{w}_{i+1} = \mathbf{w}_i + y_i \ \mathbf{x}_i$$

kde $[\mathbf{x}_i, y_i]$ je trénovací příklad ($\boldsymbol{x_i}$ je vektor hodnot vstupních atributů a y_i je informace o zařazení příkladu $\boldsymbol{x_i}$ do třídy), $\boldsymbol{w_i}$ je váha před modifikací a $\boldsymbol{w_{i+1}}$ je váha po modifikaci. Váha $\boldsymbol{w}$ může růst nade všechny meze, což neodpovídá biologické realitě.

Adaline samotná používala jiný způsob učení, tzv. *gradientní metodu*. Zde se vycházelo z požadavku aby chování sítě bylo co nejvíce podobno celkovému chování učitele, který provádí klasifikaci vstupních příkladů trénovací množiny. Zavádí se tedy tzv. *střední kvadratická chyba*, která má pro n příkladů z trénovací množiny D_{TR} podobu

$$\mathrm{Err}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Tuto chybu se snažíme minimalizovat (y_i je hodnota cílového atributu a $\hat{y}_i$ je výsledek zařazení sítí). Z požadavku na minimum střední kvadratické chyby lze odvodit následující pravidlo pro modifikaci vah:

$$\mathbf{w} \leftarrow \mathbf{w} + \Delta\,\mathbf{w}\,,$$

kde

$$\Delta\,\mathbf{w} = -\eta\,\frac{\partial \mathrm{Err}}{\partial \mathbf{w}} = \eta \sum_{i=1}^{n}(y_i - \hat{y}_i)\,\mathbf{x}_i$$

Modifikace vah se tedy provádí až po zpracování celé trénovací množiny.

Odvození vztahu pro modifikaci vah $\boldsymbol{w}$ pro případ, že neuron realizuje pouze vážený součet vstupů, tedy že

[6] Pro výpočet byla použita implementace neuronových sítí ze systému Weka.

[7] Na počátku procesu učení jsou váhy $\boldsymbol{w}$ nastaveny náhodně (na nějaké hodnoty blízké 0).

$$\hat{y}_i = \mathbf{w} \cdot x_i$$

bylo již jednou uvedeno v podkapitole věnované učení jako aproximace funkcí. Připomeňme si, že

$$\frac{\partial Err}{\partial \mathbf{w}} = \frac{1}{2} \sum_{i=1}^{n} \frac{\partial}{\partial \mathbf{w}} (y_i - \hat{y}_i)^2 = \frac{1}{2} \sum_{i=1}^{n} 2 (y_i - \hat{y}_i) \frac{\partial}{\partial \mathbf{w}} (y_i - \hat{y}_i) =$$

$$= \sum_{i=1}^{n} (y_i - \hat{y}_i) \frac{\partial}{\partial \mathbf{w}} (y_i - \mathbf{w} \cdot \mathbf{x}_i) = \sum_{i=1}^{n} (y_i - \hat{y}_i) (- \mathbf{x}_i)$$

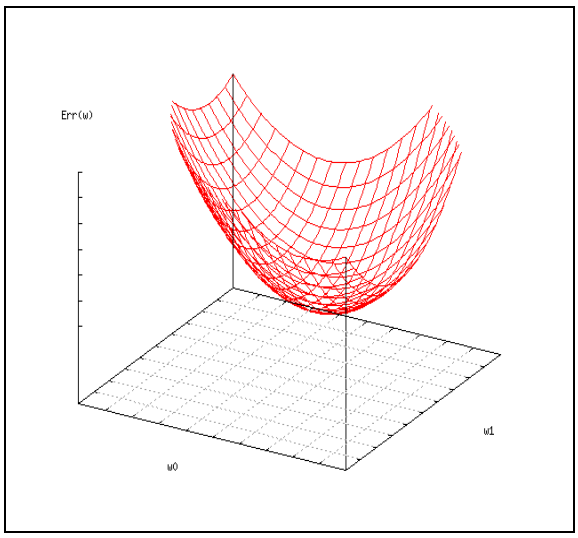
Chybová funkce *Err(**w**)* bude mít v tomto případě jedno globální minimum (Obr. 6)[8]. Uvedený postup výpočtu vah nalezne takové váhy ***w***, které budou odpovídat tomuto minimu. V případě lineárně separabilních tříd toto minimum odpovídá bezchybné klasifikaci příkladů, tedy nalezneme takové váhy ***w***, pro které

$$Err(\mathbf{w}) = 0 .$$

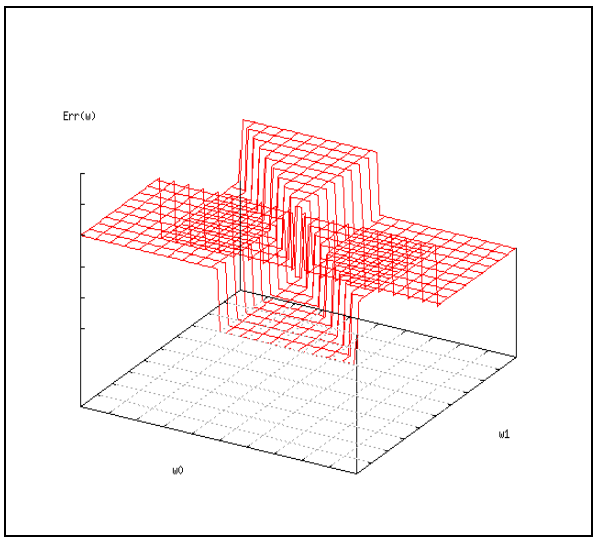
Vzhledem k tomu, že *Adaline* používá skokovou aktivační funkci

$$\hat{y}_i = \text{sign} (\mathbf{w} \cdot x_i),$$

je podoba chybové funkce poněkud složitější (Obr. 7)[9]. Přesto můžeme i pro Adaline použít výše odvozené pravidlo pro modifikaci vah. Nabývá-li totiž skutečný výstup y pouze hodnot ± 1, pak pro výstup neuronu $\hat{y} = \pm 1$ platí $\text{sign}(\mathbf{w}\, x) = \mathbf{w} \cdot \mathbf{x}$.



Obr. 6 Chybová funkce pro lineární aktivaci



Obr. 7 Chybová funkce pro skokovou aktivaci

[8] Uvedená chybová funkce se týká úlohy nonekvivalence uvedené na Obr. 9.

[9] Opět se jedná o nonekvivalenci z Obr. 9.

Používanější varianta, tzv. stochastická (inkrementální) aproximace předpokládá, že se váhový vektor $\boldsymbol{w}$ modifikuje okamžitě poté, kdy při klasifikaci i-tého příkladu došlo k odchylce mezi požadovaným a skutečným výstupem systému [10].

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \Delta \mathbf{w}_i ,$$

$$\Delta \mathbf{w}_i = \eta \, (y_i - \hat{y}_i) \, \mathbf{x}_i .$$

Ze vzorce je vidět, že při učení chybí zpětná vazba. V případě odchylky se váhový vektor $\boldsymbol{w}$ modifikuje, aniž by se zjišťovalo, zda provedená změna měla vliv na výstup systému $\hat{y}_i$. Trénovací množinou D_{TR} se tedy prochází *opakovaně* tak dlouho, dokud není neuron „naučen“ (střední kvadratická chyba je minimální resp. dostatečně malá) [11]. V případě *Adaline* je možno dosáhnout *nulovou* chybu pouze pro *lineárně separabilní třídy*. Pouze v tomto případě *Adaline* bezchybně klasifikuje všechny prvky trénovací množiny. Pokud třídy nejsou lineárně separabilní, učení se ustálí v okamžiku nalezení minima (ne nutně globálního). Parametr η, nazývaný *learning rate*, udává „krok“, kterým se mění váhový vektor $\boldsymbol{w}$. Je-li η příliš velké, můžeme při přibližování k minimu toto minimum minout, je-li η příliš malé, iterování trvá příliš dlouho. Obvykle se volí $\eta \in [0, 10]$ (často η je 0.1 nebo 0.05, navíc s rostoucím počtem iterací se η může zmenšovat).

5.4.2 Perceptron

Zatím jsme se zabývali jedním neuronem. Jak to vypadá s neuronovými sítěmi? První neuronová síť pochází z roku 1957. Rosenblattův *Perceptron* byl navržen jako model zrakové soustavy [12]. *Perceptron* je hierarchický systém tvořený třemi úrovněmi (viz Obr. 8). První z nich, nazývaná sítnice, slouží k přijímání informace z prostředí. Je tvořena receptory, prvky, jejichž výstup nabývá hodnoty 1 nebo 0 podle toho, zda jsou prostředím excitovány nebo ne. Výstupy receptorů jsou (přes náhodně zvolené vazby) přivedeny na asociativní elementy. Asociativní element připomíná výše popsaný adaptivní lineární neuron s tím, že všechny váhy w_i mají pevné hodnoty +1 nebo -1. Asociativní element se aktivuje (vydá hodnotu 1), pokud souhrn jeho vstupů překročí zadaný práh. Počet asociativních elementů je řádově desítky tisíc.Výstupy z asociativních elementů jsou náhodně zvolenými vazbami propojeny na reagující elementy, jejichž počet odpovídá počtu tříd, do kterých klasifikujeme. Reagující elementy realizují vážený součet

$$\sum_{i=1}^{m} w_i \, x_i$$

V bloku výběr maxima se vybere pro daný obraz ten reagující element, který má nejvyšší výstup, a který tedy odpovídá třídě, do které je obraz zařazen.

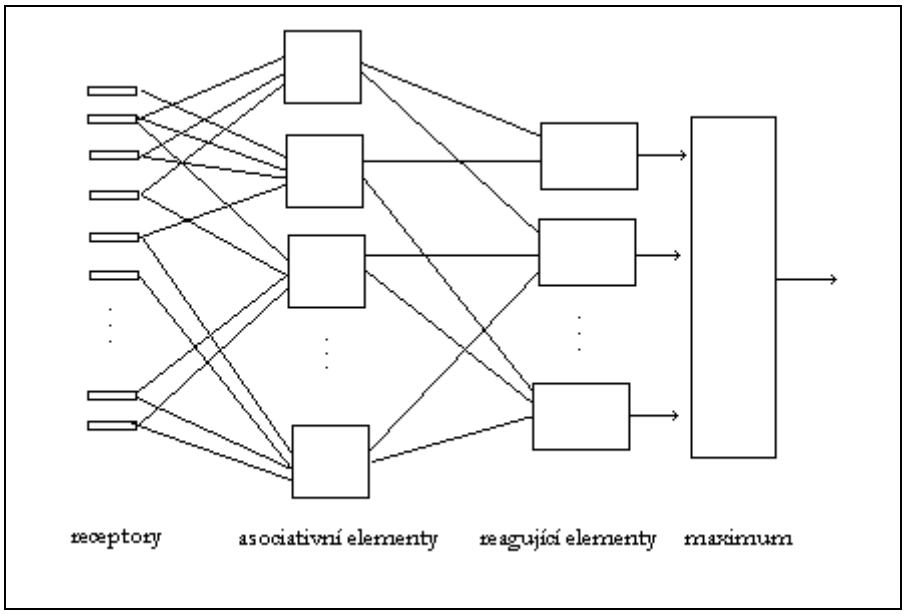
Učení *Perceptron*u probíhá na úrovni reagujících elementů. Pravidlo pro modifikaci vah odpovídá výše uvedené stochastické aproximaci

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \Delta \mathbf{w}_i , \qquad \Delta \mathbf{w}_i = \eta \, (y_i - \hat{y}_i) \, \mathbf{x}_i .$$

[10] Vzhledem k tomu, že jak $\hat{y}_i$ tak y_i nabývá hodnoty 1 nebo 0, je jejich rozdíl 0 nebo +1 nebo -1.

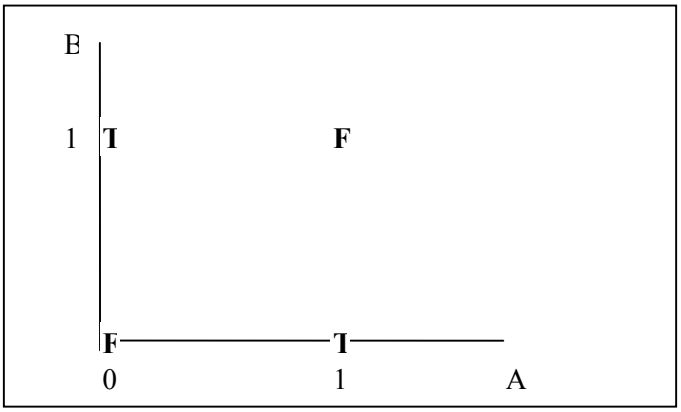
[11] Počet těchto průchodů trénovací množinou je obvykle značný. Počet takzvaných iterací se pohybuje v řádech tisíců. Učení neuronů (a neuronových sítí) je tedy poněkud zdlouhavý proces.

[12] Zde uvedený popis Perceptronu je převzat z [Kotek a kol., 1980]. Původní Rosenblattův popis byl publikován v Rosenblatt,F.: The Perceptron, A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, Vol. 65, No. 6, 1958.



Obr. 8 Perceptron (podle [Kotek a kol., 1980])

Sám Rosenblatt zamýšlel perceptron jako model mozku. Myšlenka perceptronu však inspirovala mnoho techniků a rychle pronikla do aplikací jako učící se klasifikátor. Po počátečním nadšení se v roce 1969 v knize Perceptrons od M.Minského a S.Paperta objevila kritika klasifikačních schopností perceptronu. Minsky a Papert ukázali, že se perceptron jako lineární klasifikátor nedokáže vyrovnat s tak jednoduchým pojmem jako je nonekvivalence (viz Obr. 9). Tato oprávněná námitka ale byla neprávem chápána jako kritika neuronových sítí jako takových.



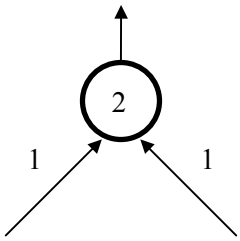
Obr. 9 Nonekvivalence

Výzkum neuronových sítí pak stagnoval až do poloviny 80. let, kdy dostal nový impuls. O renesanci neuronových sítí se zasloužili především Hopfield, Hecht-Nielsen a Kohonen. V té době se podařilo ukázat, že libovolnou spojitou funkci $f:[0,1]^n \rightarrow R^m$, $f(\mathbf{x}) = y$ lze s libovolnou přesností aproximovat pomocí třívrstvé sítě, která je tvořena n neurony ve vstupní vrstvě, $2n+1$ neurony v prostřední vrstvě a m neurony ve výstupní vrstvě[13].

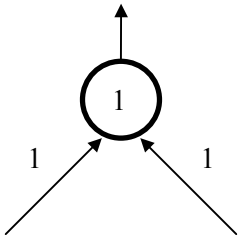
[13] Jde o modifikaci Kolmogorovova teorému z roku 1957, který ukázal, že libovolnou funkci z n-rozměrné krychle $[0,1]^n$ do množiny reálných čísel R lze vyjádřit jako funkci jedné proměnné.

Zde uvádíme konstruktivní důkaz toho, že třívrstvou sítí lze sestrojit libovolnou logickou funkci: Pomocí jednoho lineárního neuronu lze realizovat

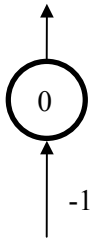
- konjunkci (uzel and) [14],



- disjunkci (uzel or),



- negaci (uzel not).



Spojením těchto elementů lze (v rámci třívrstvé sítě) modelovat libovolnou logickou funkci vyjádřenou v disjunktivní normální formě. Např. nonekvivalenci

$$A \neq B$$

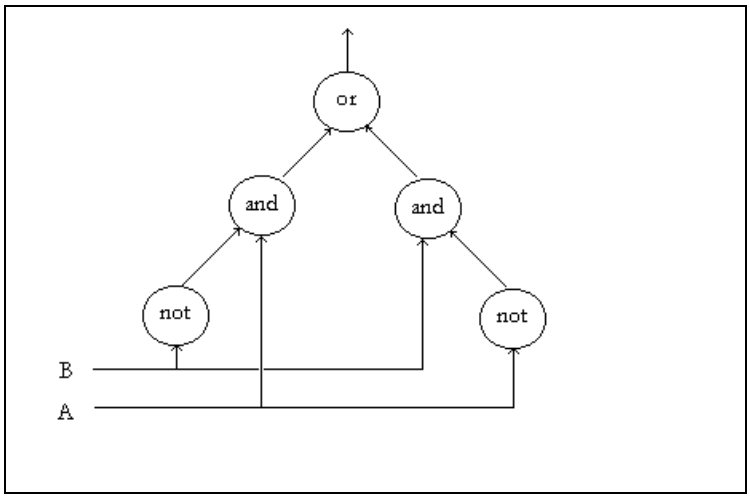
můžeme zapsat jako

$$(A \wedge \neg B) \vee (\neg A \wedge B)$$

čemuž odpovídá neuronová síť uvedená na Obr. 10.

Výpočetní síla neuronových sítí je tedy stejná jako výpočetní síla logických obvodů. Tím byla vyvrácena námitka Minského a Paperta platící pro Perceptron.

[14] Schéma konjunkce (i dále uvedené disjunkce a negace) je zjednodušením dříve uvedeného schématu *Adaline*, kde (v případě konjunkce) $w_0 = 2$, $w_1 = 1$, $w_2 = 1$; tedy číslo v kroužku označuje práh a čísla u šipek jsou váhy vstupů. Jak vstupy x_1, x_2 tak výstup $\hat{y}$ nabývají pouze hodnot 0 nebo 1.

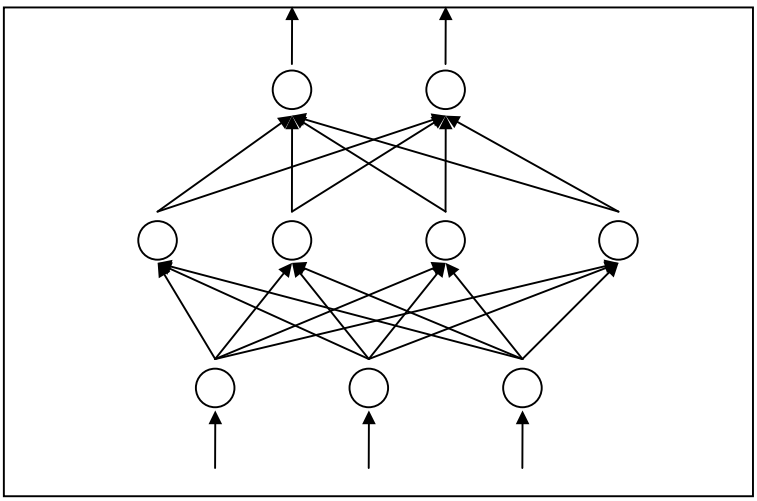


Obr. 10 Nonekvivalence vyjádřená neuronovou sítí

5.4.3 Topologie soudobých sítí

Kromě toho, že byly navrženy nové topologie neuronových sítí, podařilo se v osmdesátých letech rovněž nalézt algoritmy, které umožňovaly učení v těchto složitějších sítích [15]. Teprve tyto algoritmy umožnily návrat neuronových sítí na výsluní popularity. V současné době je známa (a používána) řada typů sítí. Podrobněji se seznámíme se dvěmi z nich: vícevrstvým perceptronem a Kohonenovou mapou.

Vrstevná síť (vícevrstvý perceptron, multi-layer perceptron) je síť složená z vrstev neuronů, kde v rámci jedné vrstvy nejsou mezi neurony žádné vazby, ale neuron z jedné vrstvy je propojen se všemi neurony vrstvy sousední. Nejpoužívanější topologií je síť s jednou skrytou vrstvou (Obr. 11). Jedná se o zobecnění jednoduchého perceptronu (případně adaptivního lineárního neuronu), které má schopnost aproximovat libovolnou spojitou funkci.



Obr. 11 Vícevrstvý perceptron s jednou skrytou vrstvou

[15] Viz např. práce Rumelhart, D.E. – Hinton,G.E. – Williams,R.J. Learning Internal Representations by Error Propagation z roku 1986.

Pro učení takovéto sítě se nejčastěji používá zobecněná gradientní metoda zvaná zpětné šíření chyby (*error backpropagation)*. Na Obr. 12 je uvedena inkremetální (stochastická) podoba tohoto algoritmu pro síť s jednou skrytou vrstvou ([Mitchell, 1997]). Opět je naším cílem minimalizovat chybu založenou na druhé mocnině rozdílu mezi skutečným a očekávaným výstupem sítě pro příklad $\boldsymbol{x}_i$ [16]

$$\mathrm{Err}(\mathbf{w}_i) = \frac{1}{2} \sum_{v \in \text{výstupy}} (y_{i,v} - \hat{y}_{i,v})^2$$

Změna váhy vazby vedoucí od neuronu *j* k neuronu *k* se tedy bude řídit gradientem funkce *Err*

$$\Delta w_{j,k} = -\eta \frac{\partial \mathrm{Err}}{\partial w_{j,k}} = -\eta \frac{\partial \mathrm{Err}}{\partial \mathrm{SUM}_k} \frac{\partial \mathrm{SUM}_k}{\partial w_{j,k}} = -\eta \frac{\partial \mathrm{Err}}{\partial \mathrm{SUM}_k} x_{j,k} ,$$

kde SUM_k je vážený součet vstupů do neuronu *k*.

Základem algoritmu je výpočet chyby na výstupech jednotlivých neuronů. Nejprve se počítá chyba pro neurony ve výstupní vrstvě (vztah 2.1.2), to pak (zpětně) umožní spočítat chybu pro neurony ve skryté vrstvě (vztah 2.1.3). Odvození těchto klíčových vztahů je uvedeno na Obr. 13. Tuto pasáž může čtenář přeskočit bez ztráty souvislostí.

Backpropagation algoritmus

1. inicializuj váhy sítě malými náhodnými čísly (např. z intervalu [-0.05,0.05])
2. dokud není splněno kritérium pro zastavení
   2.1. pro každý příklad [**x**,y] z trénovacích dat
      2.1.1. spočítej výstup out_u pro každý neuron *u* v síti
      2.1.2. pro každý neuron *v* ve výstupní vrstvě spočítej chybu
         $\mathrm{error}_v = \mathrm{out}_v (1 - \mathrm{out}_v)(y_v - \mathrm{out}_v)$
      2.1.3. pro každý neuron *s* ve skryté vrstvě spočítej chybu
         $\mathrm{error}_s = \mathrm{out}_s (1 - \mathrm{out}_s) \sum_{v \in \text{výstup}} (w_{s,v} \mathrm{error}_v)$
      2.1.4. pro každou vazbu vedoucí od neuronu *j* do neuronu *k* modifikuj váhu vazby
         $w_{j,k} = w_{j,k} + \Delta w_{j,k}$, kde $\Delta w_{j,k} = \eta \mathrm{error}_k x_{j,k}$

Obr. 12 Algoritmus Backpropagation [17]

Při zpětném šíření se v kroku práce sítě informace (výsledky) šíří od vstupní vrstvy k vrstvě výstupní, v kroku učení se váhy modifikují po vrstvách od výstupu k vstupu. Pro zastavení procesu učení se používají tato kritéria:

- ustálení chybové funkce (v minimu),
- dosažení předem zadaného počtu iterací,
- pokles pod předem zadanou hodnotu chybové funkce.

Vícevrstvá síť potřebuje pro učení oklasifikované příklady. Pracuje tedy v režimu učení s učitelem.

[16] V neinkrementálním případě minimalizujeme střední chybu $\mathrm{Err}(\mathbf{w}) = \sum_i \mathrm{Err}(\mathbf{w}_i)$.

[17] Výstup z neuronu *u* je označen symbolem out_u (output), $x_{j,k}$ označuje *j*-tý vstup do *k*-tého neuronu $w_{j,k}$ označuje váhu *j*-tého vstupu do *k*-tého neuronu, neurony používají sigmoidální nelinearitu.

Nejprve modifikace vah pro výstupní vrstvu. Pro neuron v ve výstupní vrstvě

$$\frac{\partial Err}{\partial SUM_v} = \frac{\partial Err}{\partial out_v} \frac{\partial out_v}{\partial SUM_v}$$

Přitom

$$\frac{\partial Err}{\partial out_v} = \frac{\partial}{\partial out_v} \frac{1}{2} \sum_{l \in \text{výstupy}} (y_l - out_l)^2 = \frac{\partial}{\partial out_v} \frac{1}{2} (y_v - out_v)^2 = \frac{1}{2} 2 \ (y_v - out_v) \frac{\partial (y_v - out_v)}{\partial out_v} = -(y_v - out_v)$$

a

$$\frac{\partial out_v}{\partial SUM_v} = \frac{\partial f(SUM_v)}{\partial SUM_v},$$

kde pro sigmoidální aktivační funkci

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

je

$$\frac{\partial f(SUM_v)}{\partial SUM_v} = f(SUM_v)\ (1 - f(SUM_v)) = out_v\ (1 - out_v)$$

tedy

$$\frac{\partial Err}{\partial SUM_v} = -(y_v - out_v)\ out_v\ (1 - out_v)$$

Při výpočtu změny vah nějakého neuronu ve skryté vrstvě vycházíme z toho, že tento neuron ovlivňuje chybu sítě prostřednictvím výstupních neuronů, se kterými je spojen. V případě plně propojené vrstevné sítě musíme brát do úvahy všechny neurony ve výstupní vrstvě. Pro neuron s ve skryté vrstvě tedy

$$\frac{\partial Err}{\partial SUM_s} = \sum_{v \in \text{výstupy}} \frac{\partial Err}{\partial SUM_v} \frac{\partial SUM_v}{\partial SUM_s}$$

Přitom $\frac{\partial Err}{\partial SUM_v}$ pro výstupní neuron jsme již spočítali výše, a

$$\frac{\partial SUM_v}{\partial SUM_s} = \frac{\partial SUM_v}{\partial out_s} \frac{\partial out_s}{\partial SUM_s} = w_{s,v} \frac{\partial out_s}{\partial SUM_s} = w_{s,v}\ out_s\ (1 - out_s)$$

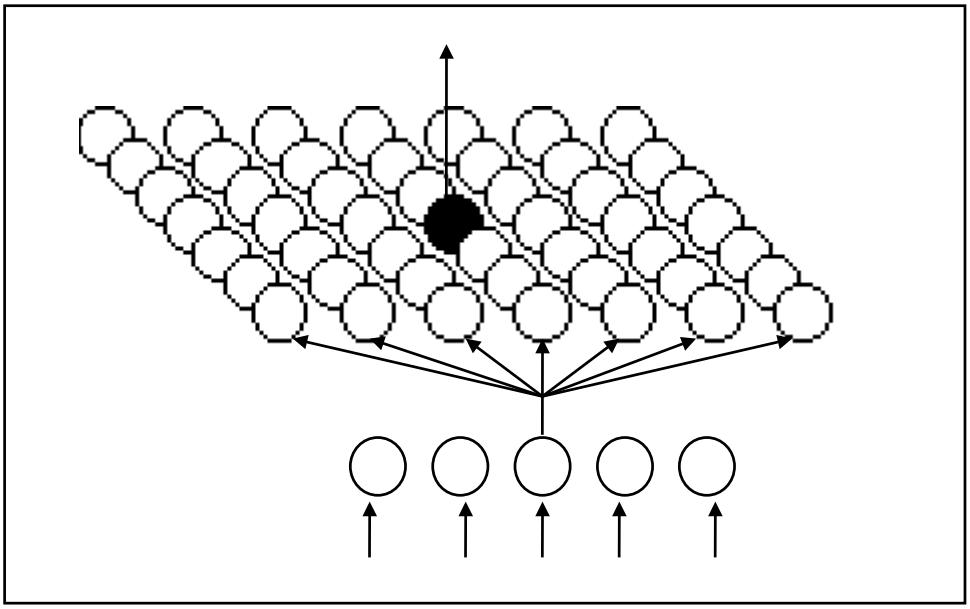
tedy

$$\frac{\partial Err}{\partial SUM_s} = out_s\ (1 - out_s) \sum_{v \in \text{výstupy}} w_{s,v} \frac{\partial Err}{\partial SUM_v}$$

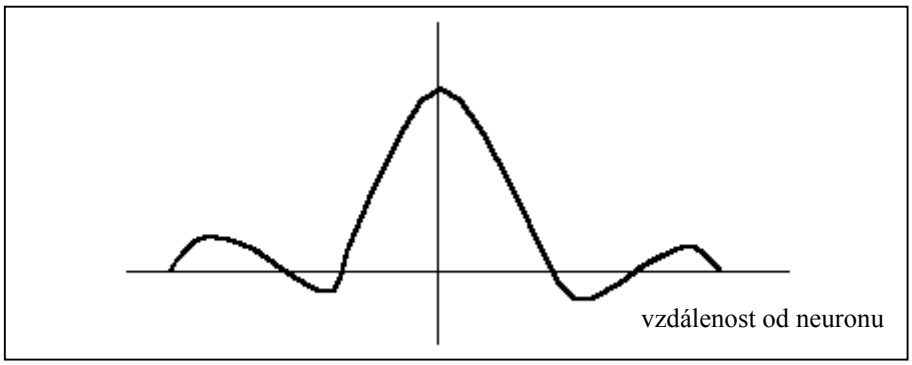
Obr. 13 Odvození modifikace vah pro algoritmus backpropagation

Kohonenova mapa (self-organizing map, SOM) je síť tvořená dvěma vrstvami; vstupní vrstvou a vrstvou neuronů uspořádaných do čtvercové matice (Kohonenovy mřížky) navzájem spojených

neuronů. Vstupní vrstva je propojena na každý neuron v mřížce (Obr. 14). Vazby mezi sousedními neurony v mřížce jsou excitační, vazby ke vzdáleným neuronům jsou inhibiční - toto vzájemné ovlivňování se nazývá laterální inhibice. Tyto vazby jsou pevné (závislost vazby na vzdálenosti od daného neuronu má podobu tzv. „mexického klobouku“ tak jak je znázorněno na Obr. 15).



Obr. 14 Kohonenova mapa



Obr. 15 Síla vazeb mezi neurony

V kroku klasifikace probíhá *kompetice* mezi neurony. Podle zásady „vítěz bere všechno“ je aktivován (výstup $\hat{y} = 1$) pouze ten neuron, který má váhy co nejblíže vstupnímu příkladu $\boldsymbol{x}$. Vítěz potlačí výstupy ostatních neuronů, které pak dají hodnotu $\hat{y} = 0$. V kroku učení se mění váhy pouze vítězného neuronu, popřípadě se váženě modifikují váhy nejbližších sousedů vítěze. Cílem učení je přiřadit příklady k jednotlivým neuronům mřížky. Na rozdíl od perceptronů, kde se minimalizuje odchylka mezi skutečným a požadovaným výstupem, u Kohonenovy sítě se minimalizuje ztrátová funkce tvaru

$$\mathrm{Err}(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{n} (\mathbf{x}_i - \mathbf{w})^2$$

Z této funkce plyne pravidlo pro modifikaci vah

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \eta\, (\mathbf{x}_i - \mathbf{w}_i)\, \hat{y}_i$$

Vzhledem k tomu, výstup $\hat{y}$ nabývá hodnot 0 nebo 1, modifikuje se pouze váha vítězného neuronu

$$\mathbf{w}_{i+1} = (1 - \eta)\, \mathbf{w}_i + \eta\, \mathbf{x}_i$$

váhy ostatních neuronů se nezmění [18].

Kohonenova mapa má schopnost samoorganizace (self-organization); nevyžaduje přítomnost učitele. Lze ji tedy použít pro shlukování prvků v trénovací množině [19]. Existuje ale i varianta této sítě pro učení s učitelem. Síť *LVQ* (Linear Vector Quantization) má stejnou topologii i režim práce při zařazování příkladů ke shlukům. Ve fázi učení ale používá informaci o tom, ke které třídě (shluku) příklad skutečně patří. Pokud je vítězný neuron reprezentantem správné třídy, modifikují se jeho váhy stejně jako u Kohonenovy mapy směrem „k příkladu“

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \eta\, (\mathbf{x}_i - \mathbf{w}_i)\, \hat{y}_i$$

v případě, že zvítězil „špatný“ neuron, jeho váhy se modifikuji směrem „od příkladu“

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \eta\, (\mathbf{x}_i - \mathbf{w}_i)\, \hat{y}_i\,.$$

5.4.4 Metoda SVM

SVM (Support Vector Machine) je metoda klasifikace, která explicitně formalizuje to, co neuronové sítě řeší implicitně. Jedná se relativně novou metodu (např. [Cortes, Vapnik, 1995]), jejíž obliba v současnosti stále stoupá.

Hlavní myšlenkou metody je, převést pomocí vhodné datové transformace úlohu klasifikace do tříd, které nejsou lineárně separabilní na úlohu klasifikace do tříd lineárně separabilních. Místo v prostoru původních atributů se tedy pohybujeme v prostoru transformovaných atributů (příznaků). Datová transformace sice představuje nárůst dimenzionality (oproti původním atributům je počet příznaků větší), řešíme ale jednodušší úlohu. V tomto novém prostoru hledáme rozdělující nadrovinu, která má největší odstup od transformovaných příkladů z trénovací množiny (tzv. maximal margin hyperplane). Rozhodující pro nalezení rozdělující nadroviny jsou samozřejmě příklady, které leží nejblíže hledané hranici mezi třídami. Tyto takzvané „podpůrné“ vektory (support vectors) daly jméno celé metodě.

Metodu SVM si přiblížíme na často citované úloze [Schölkopf, Smola, 2001]. Původní příklady jsou popsány pomocí dvou atributů x_1 a x_2. Tyto příklady jsou rozděleny do dvou tříd pomocí elipsy (Obr. 16 vlevo). Použitím datové transformace

$$\Phi(\mathbf{x}) = \Phi(x_1, x_2) = (x_1^2, \sqrt{2}x_1x_2, x_2^2) = \mathbf{z}$$

převedeme úlohu na hledání nadroviny v třírozměrném prostoru z_1, z_2, z_3 (Obr. 16 vpravo). Hledáme tedy lineární diskriminační funkci[20]

$$f(\mathbf{x}) = \mathbf{w} \cdot \Phi(\mathbf{x}) + w_0\,.$$

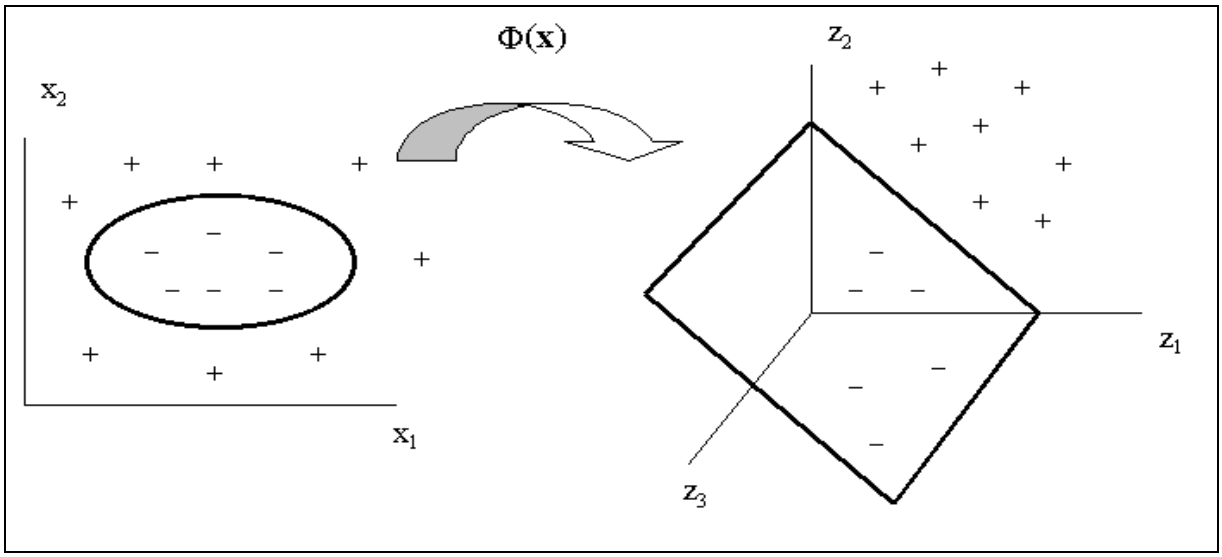
[18] V jiné variantě tohoto algoritmu se modifikují váhy i ostatních neuronů, bere se při tom do úvahy i vazba mezi neurony (Obr. 15). Váha od *j*-tého ke *k*-tému neuronu se pak modifikuje podle vzorce

$$w_{j,k} = w_{j,k} + \eta\ h(w_{vítěz}, w_{j,k})\ \ ||x_i - w_{j,k}||\ \hat{y}_i$$

kde $h(w_{vítěz}, w_{j,k})$ je funkce vyjadřující vazbu mezi vítězným neuronem a neuronem, pro který modifikujeme váhu.

[19] Lze ukázat, že tato síť odpovídá algoritmu pro shlukování podle *k* středů (*k*-means clustering) známému ze statistiky.

[20] Analogie s lineárním neuronem je zřejmá.



Obr. 16 Ilustrace metody SVM

Z požadavku na nadrovinu s maximálním odstupem od trénovacích příkladů lze odvodit podobu vah[21]

$$\mathbf{w} = \sum_i \lambda_i y_i \Phi(\mathbf{x}_i),$$

kde $[\mathbf{x}_i, y_i]$ jsou trénovací příklady, $y_i \in \{-1,1\}$. Tedy po dosazení

$$f(\mathbf{x}) = \sum_i \lambda_i y_i \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}) + w_0 .$$

Skalární součin v prostoru příznaků

$$\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x})$$

lze spočítat, aniž bychom potřebovali znát tvar transformační funkce Φ. Můžeme totiž využít výpočetní trik založený na speciálních funkcích zvaných kernel functions. Pro výše uvedený příklad bude touto funkcí polynom[22]

$$K(\mathbf{x}_i,\mathbf{x}) = (\mathbf{x}_i \cdot \mathbf{x})^2.$$

Platí totiž

$$\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}) = (x_{i1}^2, \sqrt{2}x_{i1}x_{i2}, x_{i2}^2)(x_1^2, \sqrt{2}x_1x_2, x_2^2) = x_{i1}^2x_1^2 + 2x_{i1}x_{i2}x_1x_2 + x_2^2 = (x_{i1}x_1 + x_{i2}x_2)^2 = (\mathbf{x}_i \cdot \mathbf{x})^2$$

Hledaná nadrovina pak má podobu

$$f(\mathbf{x}) = \sum_i \lambda_i y_i K(\mathbf{x}_i, \mathbf{x}) + w_0 .$$

Vracíme se tedy při výpočtu nazpátek do původního (méněrozměrného) prostoru atributů. Do výpočtu navíc vstupují pouze ty příklady $[\mathbf{x}_i, y_i]$, které leží nejblíže hranici mezi třídami. Nalezení příslušných parametrů je úlohou kvadratického programování.

[21] Parametry λ_i jsou takzvané Lagrangeovy multiplikátory.

[22] Obvykle se jako funkce jádra používají polynomy $(\mathbf{x}_i \cdot \mathbf{x})^k$ nebo gausovská funkce.

5.4.5 Neuronové sítě a dobývání znalostí z databází

Z hlediska dobývání znalostí z databází představují neuronové sítě jeden z nejpoužívanějších nástrojů pro tvorbu automatických systémů pro klasifikaci nebo predikci. Neuronové sítě jsou vhodnou alternativou k rozhodovacím stromům a pravidlům v situacích, kdy netrváme na srozumitelnosti nalezených znalostí.

Jiným dobrým důvodem pro použití neuronových sítí je převaha numerických atributů. Zatímco symbolické metody (stromy, pravidla) jsou šity na míru kategoriálním datům a numerická data vyžadují speciální zacházení (diskretizaci), nyní je situace opačná. Neuronové sítě jsou vhodnější pro data numerická[23] a problémy působí data kategoriální. Standardním řešením tohoto problému je tzv. *binarizace*. Pro jeden kategoriální atribut vytvoříme tolik nových binárních atritbutů, kolik měl původní atribut různých hodnot. Hodnota 1 u nového atributu A'_k vyjadřuje, že původní atribut A měl pro daný objekt hodnotu v_k.; ostatní atributy A'_q, $q \neq k$ pak mají hodnotu 0. Má-li původní atribut pouze dvě možné hodnoty, binarizaci nemusíme provádět; stačí když jedné z hodnot přiřadíme novou hodnotu 1 a druhé z hodnot novou hodnotu 0. Tato binarizace se často provádí v rámci algoritmu pro učení neuronové sítě, uživatel tedy nebývá zatěžován nutností data transformovat ručně.

Vezměme k ruce opět náš příklad bankovní aplikace. Pro vytvoření klasifikačního modelu můžeme použít vícevrstvý perceptron s jednou skrytou vrstvou. Topologii sítě bude určovat zvolená úloha i podoba trénovacích dat. Počet neuronů ve vstupní vrstvě bude vycházet z počtu vstupních atributů. Vzhedem k tomu, že všechny atributy jsou kategoriální, musíme provést binarizaci. Získáme tak 6 binárních atributů *příjem_vysoký*, *konto_vysoké*, *konto_střední*, *konto_nízké*, *pohlaví_muž*, *nezaměstnaný*, *úvěr* ; ve vstupní vrstvě tedy bude 6 neuronů. Podobu dat po binarizaci ukazuje Tab. 2. Výstupní vrstva bude tvořena jedním neuronem reprezentujícím závěr *úvěr(ano)*. Jeho výstup (v intervalu [0, 1]) bude interpretován jako doporučení zda půjčit ($\acute{y}>0.5$) nebo nepůjčit ($\acute{y}<0.5$)[24].

Problémem bývá určení počtu neuronů ve skryté vrstvě. Zdálo by se, že čím více, tím lépe. S rostoucím počtem neuronů ve skryté vrstvě (resp. s růstem počtu skrytých vrstev) se zvyšuje nelinearita chování sítě, rostou ale nároky na proces učení (potřebný počet příkladů, doba učení). Příliš rozsáhlá síť má rovněž tendenci k přeučení (overfitting); může se příliš zaměřit na nevýznamné podrobnosti vyskytující se v trénovacích datech, které ale nemají význam z hlediska řešené úlohy. Neexistuje obecný návod jak zvolit počet neuronů (jedna z používaných heurisitk radí, aby neuronů ve skryté vrstvě bylo dvakrát tolik jako je neuronů ve vstupní vrstvě). Přidržíme se této heuristiky a zvolíme 10 neuronů. V praxi je obvyklé zkusit více sítí s různými parametry (topologiemi, přechodovými funkcemi ...) a na základě jejich chování na testovacích datech vybrat tu nejlepší.

Pro výše popsanou topologii 6-10-1 získáme pomocí algoritmu zpětného šíření[25] pro naše data takové nastavení vah, že síť bude bezchybně klasifikovat všechny trénovací příklady (úplný popis sítě je uveden v příloze).

Naše data z Tab. 2 jsou lineárně separabilní (což jsme dopředu nemuseli vědět), stačila by tedy i jednodušší síť tvořená jedním neuronem[26]. Obecně ale platí, že vícevrstvý perceptron nalezne libovolně složitý popis tříd, tedy v prostoru atributů libovolně složitou „hranici“ mezi třídami.

---

[23] I zde se ale často provádí transformace – normalizace dat.

[24] Obecně platí, že ve výstupní vrstvě je tolik neuronů, kolik je tříd mezi kterými se systém rozhoduje. V případě dvou tříd ale stačí jeden neuron.

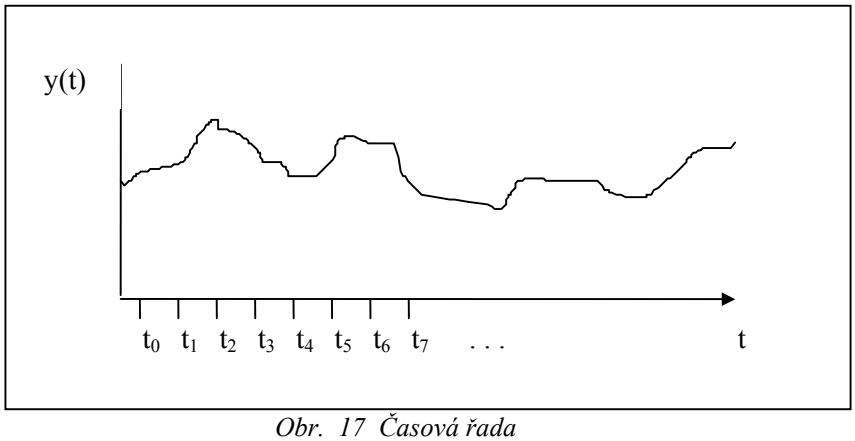
[25] Opět byla použita implementace ze systému Weka.

[26] Rovnice odpovídající rozdělující nadroviny je 6.7898 příjem_vysoký + 2.8184 konto_vysoké - 1.9507 konto_střední - 5.1480 konto_nízké - 0.5172 pohlaví_muž - 3.7801 nezaměstnaný - 2.1758 = 0.

klient	příjem_ vysoký	konto_ vysoké	konto_ střední	konto_ nízké	pohlaví_ muž	nezaměst naný	úvěr
k1	1	1	0	0	0	0	1
k2	1	1	0	0	1	0	1
k3	0	0	0	1	1	0	0
k4	0	1	0	0	0	1	1
k5	0	1	0	0	1	1	1
k6	0	0	0	1	0	1	0
k7	1	0	0	1	1	0	1
k8	1	0	0	1	0	1	1
k9	0	0	1	0	1	1	0
k10	1	0	1	0	0	0	1
k11	0	0	1	0	0	1	0
k12	0	0	1	0	1	0	1

Tab. 2 Binarizovaná data

Naše bankovní data ilustrují jeden typ použití neuronových sítí, klasifikační úlohy. Typičtějším příkladem použití neuronových sítí je úloha predikce časových řad (cen akcií, směnných kurzů měn, spotřeby elektrické energie nebo meteorologické situace). Vícevrstvý perceptron může predikovat jak kvalitativní vývoj (vzrůst, pokles), tak i konkrétní hodnoty. S výhodou se v tomto typu aplikací použije skutečnost, že neuronové sítě bez problémů zpracovávají numerické atributy. Časová řada má obvykle podobu hodnot dané veličiny získané v po sobě jdoucích časových okamžicích (Obr. 17)[27]. Z této řady je třeba získat trénovací příklady pro neuronovou síť. Řekněme, že nás bude zajímat predikce na jeden časový okamžik dopředu; výstupní vrstva bude tedy tvořena jediným neuronem. Počet neuronů ve vstupní vrstvě závisí na tom, jak dlouhý historický úsek chceme brát při predikci v úvahu. Toto silně závisí na predikované veličině; o chování veličiny můžeme mít nějaké další dodatečné informace (předpokládané periodické nebo sezónní chování apod.). Budeme-li např. chtít predikovat ze čtyř posledních hodnot, bude mít vstupní vrstva čtyři neurony. Počet neuronů ve skryté vrstvě bude opět předmětem experimentování. Takto navržené topologii odpovídá podoba trénovacích dat uvedená na Obr. 18 (pro případ, že chceme predikovat přímo hodnotu veličiny), resp. na Obr. 19 (pro případ, že chceme predikovat jen změnu veličiny). V obou případech vytváříme trénovací příklady ze sekvencí po sobě jdoucích hodnot časové řady.



Obr. 17 Časová řada

[27] Obvykle se hodnoty veličiny zaznamenávají po uplynutí stejného časového úseku (ekvidistantní vzorkování), některé modernější přístupy pro predikci např. směnných kurzů ale používají vzorkování, které závisí na chování dané veličiny - tzv. tiková data se zaznamenávají v okamžiku, kdy změna kurzu překročí zadaný práh.

vstupy				výstup
$y(t_0)$	$y(t_1)$	$y(t_2)$	$y(t_3)$	$y(t_4)$
$y(t_1)$	$y(t_2)$	$y(t_3)$	$y(t_4)$	$y(t_5)$
$y(t_2)$	$y(t_3)$	$y(t_4)$	$y(t_5)$	$y(t_6)$
	. . .			

Obr. 18 Trénovací data pro predikci hodnot

vstupy				výstup
$sign(y(t_1) - y(t_0))$	$sign(y(t_2) - y(t_1))$	$sign(y(t_3) - y(t_2))$	$sign(y(t_4) - y(t_3))$	$sign(y(t_5) - y(t_4))$
$sign(y(t_2) - y(t_1))$	$sign(y(t_3) - y(t_2))$	$sign(y(t_4) - y(t_3))$	$sign(y(t_5) - y(t_4))$	$sign(y(t_6) - y(t_5))$
	. . .			

Obr. 19 Trénovací data pro predikci změny

Použití neuronových sítí s sebou nese problém interpretace. Znalosti získané neuronovými sítěmi jsou zcela nesrozumitelné pro uživatele (znalosti jsou dány topologií sítě a váhami vazeb mezi neurony). S tím souvisí i neschopnost sítí podávat vysvětlení. Z neuronových sítí se tedy stává černá skříňka, do které není vidět. Objevují se ale pokusy převést znalosti uložené v neuronových sítích do srozumitelnější podoby [Shavlik, 1992]. Obdobně se zkoumá možnost využít doménové znalosti při inicializaci neuronových sítí (např. systém KBANN popsaný v [Towell, Shavlik 1994]). V [Berka, Sláma, 1998] lze pak nalézt postup, kdy se nejprve (na základě znalostí) nastaví topologie sítě, ta se naučí z dat váhy vazeb a pak je zpět převedena do vhodnější reprezentace. Ve všech uvedených přístupech se vychází z toho, že vazby mezi neurony lze interpretovat jako pravidla. Tak např. pro neuron na Obr. 2 by měla takováto pravidla podobu:

IF příjem(vyhovuje) THEN úvěr(ano) (w_1)

IF konto(vyhovuje) THEN úvěr(ano) (w_2)

IF příjem(nevyhovuje) THEN úvěr(ano) ($-w_1$)

IF konto(nevyhovuje) THEN úvěr(ano) ($-w_2$)

úvěr(ano) ($-w_0$)

Při klasifikaci nového případu se použijí všechna aplikovatelná pravidla způsobem známým z kompozicionálních expertních systémů[28].

Na závěr této části ještě jedna poznámka. Neuronové sítě se výrazně liší od tradiční von Neumannovy koncepce počítače. Není v nich striktně oddělen procesor a paměť a informace v nich není lokalizována na nějakém pevném místě (adrese) ale je „rozprostřena“ ve vahách po celé síti. To umožňuje fungování sítě i v případě částečného poškození sítě, při neúplných nebo zašuměných datech apod. Mají v tomto smyslu blíže k (lidskému) mozku než klasické počítače.

[28] Kompozicionální expertní systémy jsou založeny na myšlence, že závěr konzultace se odvodí ze všech aplikovatelných pravidel tak, že se zkombinují dílčí příspěvky těchto pravidel.

Literatura:

[Berka, Sláma, 1998] Berka,P. - Sláma,M.: Using neural nets to learn weights of rules for compositional expert systems. In: (Mira, Pobil, Ali eds.) Methodology and Tools in Knowledge-Based Systems. Proc. 11th Int. Conf. on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems IEA-98-AIE, LNAI 1415, Springer, 1998, 511-519.

[Cortes, Vapnik, 1995] Cortes,C. – Vapnik,V.: Support vextor networks. *Machine Learning*, Vol 20, 1995, 273-297.

[Hecht-Nielasen, 1991] Hecht-Nielsen,R.: Neurocomputing. Addison Wesley, 1991, ISBN 0-201-09355-3.

[Kohonen, 1995] Kohonen,T.: Self-Organizing Maps, Springer, 1995.

[Kotek a kol., 1980] Kotek,Z. - Chalupa,V. - Brůha,I. - Jelínek,J.: Adaptivní a učící se systémy. SNTL, Praha, 1980.

[Mitchell, 1997] Mitchell,T.: Machine learning. McGraw-Hill. 1997. ISBN 0-07-042807-7

[Novák a kol.,1992] Novák,M. - Faber,J. - Kufudaki,O.: Neuronové sítě a informační systémy živých organizmů. Grada, Praha, 1992.

[Schölkopf, Smola, 2001] Schölkopf,B. - Smola,A.J.: Learning with Kernels. MIT Press, 2001.

[Shavlik, 1992] Shavlik,J.: A framework for combining symbolic and neural learning. Technical Report no. 1123, Comp. Sci. Dept., Uni. of Wisconsin, 1992.

[Šíma, Neruda, 1996] Šíma,J. – Neruda,R.: Teoretické otázky neuronových sítí. Matfyzpress, 1996.

[Towell, Shavlik, 1994] Towell,G. - Shavlik,J.: Knowledge-based artificial neural networks. *Artificial Intelligence,* 70(1-2), 1994, 119-165.