UMELÉ NEURÓNOVÉ SIETE

Matematické modelovanie

MARTIN TIMOTHY TIMKO

7.3. 2015 – 21.7 2017

1 LINEÁRNE MODELY NEURÓNOVÝCH SIETÍ

Keď som sa prvý krát dozvedel o umelých neurónových sieťach nevedel som si predstaviť ako to celé vôbec môže fungovať. Grafové algoritmy, graf ako matematická štrukúra, ma zaujali. Hneď som v nich videl veľký potenciál a bolo zrejmé, že úzko súvisia s neurónovými sieťami. Ako som hlbšie prenikal do tajov umelých neurónových sietí a čiastočne aj umelej inteligencie ako takej, stále viac som v nich videl pomerne komplikovanú matematiku, avšak v celej jej kráse, teda umelé neurónové siete zasahujú do širokej časti matematiky nielen do problematiky grafov.

Ľudia zaoberajúci sa umelými neurónovými sieťami alebo všeobecne umelou inteligenciou fascinuje namjä tajuplnosť až záhadnosť prepojenia biologicky orientovaných neurónových sietí s umelými. Nedá sa povedať, že by štúdium umelých neurónových sietí bolo výhradne pre matematikov, ale na druhej strane táto oblasť je s ňou veľmi úzko prepojená, ostatne tak ako skoro všetko, a bohužiaľ alebo chválabohu, jej možnosti sú ňou determinované. Na jednej strane sa môžte inšpirovať biológiou, teda v podstate fyzikou, na druhej strane ste plne obmedzení možnosťami matematiky. Čo nie je možné matematicky popísať, to nie je možné prakticky realizovať na počítači.

Lineárne neurónové siete sú najjednoduchšie umelé neurónové siete. Základom je formálny neurón s lineárnou aktivačnou funkciou f. Máme niekoľko vstupných kanálov, ktoré označíme vektorom $\vec{x}$:

$$\vec{x} = (x_1, x_2, ..., x_n) \tag{1}$$

Tieto vstupné kanály budú váhované pomocou váhového vektora w:

$$\vec{w}^T = (w_1, w_2, ..., w_n) \tag{2}$$

Váhový vektor priraďuje od každého vstupu po výstup určitú hodnotu, podobne ako pri grafoch. Potom výstupná aktivita lineárneho neurónu bude:

$$\vec{y} = f(\vec{w}^T \vec{x} + \xi) \tag{3}$$

Symbol ξ je prah aktivácie a najčastejšie je aktivačná funkcia identická, takže je zanedbateľná. Tu už je priamo paralela s biológiou, teda, biologický neurón obsahuje tzv. dendrity, čo predstavuje vstup impulzov do tela neurónu (soma), a napokon obsahuje tzv. axóny, výstup impulzov. Tieto zložky sa v celku nazývajú ako neurón. Samozrejme v mozgu prebiehanjú zložité biochemické deje, ktoré sú ale matematicky pomerne silne idealizované. Podobnosť s umelým neurónom je teda v impulzoch, ktoré spracovávame po technickej stránke ako digitalizované signály, aby ich bolo možné realizovať matematicky, na počítači. Takýto technický neurón predstavuje obyčajnú funkciu, ktorá má nejaké vstupy a nejaké výstupy. Jednotlivé vstupy/výstupy nemusia byť lineárne, ale môžu predstavovať celú škálu matematických funkcií, v závislosti od toho, čo chceme modelovať.

Celá veda v tomto smere spočíva v tom, že jednotlivé neuróny môžu spolu navzájom interagovať prostredníctvom tzv. synapsy (analógia z biológie). Je to miesto stretu neurónov, čo v preklade signálov bude znamenať, že jednotlivé výstupné charakteristiky daných neurónov sa v tomto mieste budú prekrývať, lepšie povedané meniť v závislosti od charakteru synapsy. Mieru zmeny signálu od jednotlivých neurónov reprezentujeme váhou, ktorá už bola spomenutá.

Takže zatiaľ máme nejaké vstupy, nejaký výstup a ováhované vstupy. Celý vtip spočíva vo váhovej matici, ktorá je zložená práve z ováhovaného vstupného vektora. Váhová matica je

matica, ktorá jednoducho popisuje vzťahy medzi vstupmi a výstupmi, ktorým priraďuje určitú hodnotu. Vzťah i-tého a j-tého neurónu označíme ako W_{ij} a celá váhová matica bude mať tvar:

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{21} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix} \tag{4}$$

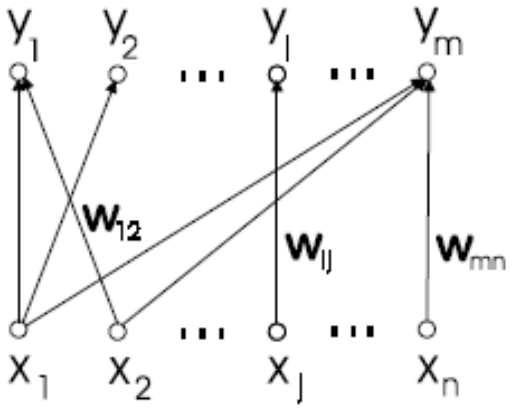
Ak na vstup priložíme vektor $\vec{x}$ dostaneme výstup $y = (y_1, y_2, ..., y_m)$, kde $y_1, y_2, ..., y_m$ sú aktivácie výstupných neurónov. Úlohou je nájsť váhovú maticu W. Táto matica bude uchovávať jednotlivé **vzory**. V procese jej vznikania (iteráciách) sa postupne menia tzv. **synaptické váhy** medzi neurónmi. Zmenu týchto váh považujeme za **"učenie"**. Ak sa v neurónovej sieti nemenia hodnoty týchto váh, sieť je "naučená".

Pri nejakom vstupe získame nejaký výstup, ktorý bude mapovať váhová matica. To znamená, že pri rovnakom vstupe by sme mali získať výstup prislúchajúci danému vstupu. Danú váhovú maticu potrebujeme natrénovať, aby takto reagovala. To sa vykoná tzv. trénovacou množinou.

Trénovacia množina je usporiadaná množina čísel, ktorou testujeme danú neurónovú sieť. Obsahuje vstupy a im prislúchajúce výstupy, ale charakter tejto množiny závisí na type neurónovej siete. V našom prípade budeme mať nasledujúcu trénovaciu množinu:

$$T = (\vec{x}^{(1)}, \vec{y}^{(1)}), (\vec{x}^{(2)}, \vec{y}^{(2)}), ..., (\vec{x}^{(n)}, \vec{y}^{(n)}) \tag{5}$$

Ide o usporiadanú dvojicu čísel, vstupu a výstupu. Jednotlivé prvky môžu obsahovať ľubovoľné reálne čísla, aj záporné. Dané označenie horných indexov vo vzorci trénovacej množiny neznamená mocninu čísla, ale označenie poradia. Lineárnu neurónovú sieť je možné vidieť na nasledujúcom obrázku.



Neurónová sieť bude fungovať, ak nájdeme k trénovacej množine T, váhovú maticu W podľa vzťahu:

$$y^{(i)} = Wx^{(i)} \tag{6}$$

Matematicky, nech x je matica X, ktorej stĺpce tvoria trénovacie vstupy $x(1), x(2), ..., x(n)$ a y je matica Y, ktorej stĺpce budú požadovať výstupy $y(1), y(2), ..., y(n)$. Potom môžme predchádzajúci vzorec prepísať do maticového tvaru:

$$Y = WX \tag{7}$$

Za predpokladu, že matica X je regulárna, je možné spočítať jej inverznú maticu a jednoducho získať požadované váhové prepojenia:

$$W = YX^{-1} \tag{8}$$

Lenže, čo to znamená, že matica X je regulárna ? To znamená, že je štvorcová a jej determinant je rôzny od nuly. Ale, ak je matica štvorcová, tak počet vzoriek v trénovacej množine je rovný dimenzii vstupných vzorov. Čiže to by znamenalo, že ak máme dimenziu vstupu 4, tak aj počet trénovacích vzoriek by musel byť 4. Dostali by sme tak maticu 4×4, čo je štvorcová matica a boli by sme schopní pri determinante rôznom od nuly spočítať inverznú maticu a nájsť váhovú maticu prepojení. Nejde o to, že by takýto prípad nemohol nikdy nastať, ale v praxi sa často nevyskytuje. Práve preto je potrebné modifikovať maticu X tak, aby bolo možné nájsť váhovú maticu W. V takomto prípade sa v pôvodnom vzorci X, modifikuje na tvar X^+:

$$W = YX^+ \tag{9}$$

kde

$$X^+ = X^T(XX^T)^{-1} \tag{10}$$

Toto platí, ak počet riadkov matice X je menší ako počet stĺpcov, čiže počet trénovacích vzoriek je väčší ako dimenzia vstupov (a hodnosť matice X je rovná n). Presne toto sme potrebovali. Matica X^+ sa nazýva **pseudoinverzná matica** matice X a platí:

$$XX^+X = X \tag{11}$$

Budeme uvažovať príklad, keď máme nájsť váhovú maticu prepojení W **lineárneho autoasociátora**, ktoré majú za úlohu zapamätať si určité vzory. Trénovacia množina bude mať tvar:

$$T = (\vec{x}^{(1)}, \vec{y}^{(1)}), (\vec{x}^{(2)}, \vec{y}^{(2)}), ..., (\vec{x}^{(n)}, \vec{y}^{(n)}) \tag{12}$$

Autoasociácia zjednodušene znamená, že to, čo dáme na vstup, chceme dostať na výstupe. Ak dáme na vstup vektor $\vec{x}^{(1)}$, ten istý vektor musíme dostať na výstupe. Čiže musíme nájsť takú váhovú maticu W, *ktorá bude umožňovať tento prepočet*. A tomuto hovoríme, že **neurónová sieť sa "naučí" rozpoznávať vzory**. Takže pre autoasociátor bude matica požadovaných výstupov $Y = X$. Z pôvodného vzťahu pre váhovú maticu W dostaneme nasledovný vzťah:

$$W = XX^+ \tag{13}$$

Ďalej budeme mať nasledovné 2 vstupy, vzory:

$$\vec{x}^{(1)} = \begin{pmatrix} -1 \\ 3 \\ 1 \\ 5 \end{pmatrix}; \vec{x}^{(2)} = \begin{pmatrix} 4 \\ -1 \\ 2 \\ 3 \end{pmatrix} \tag{14}$$

$$X = \begin{pmatrix} -1 & 3 & 1 & 5 \\ 4 & -1 & 2 & -3 \end{pmatrix} \sim \begin{pmatrix} -1 & 3 & 1 & 5 \\ 0 & 11 & 6 & 17 \end{pmatrix} \tag{15}$$

Zároveň sme vyšetrili, či sú dané vstupy(vektory), lineárne nezávislé, keď usudzujeme hodnosť $h = 2$, vektory sú teda lineárne nezávislé. Najskôr spočítame pseudoinverznú maticu X^+:

$$X = \begin{pmatrix} -1 & 3 & 1 & 5 \\ 4 & -1 & 2 & -3 \end{pmatrix} \sim \begin{pmatrix} -1 & 3 & 1 & 5 \\ 0 & 11 & 6 & 17 \end{pmatrix} \tag{16}$$

$$X^+ = (XX^T)^{-1}X \tag{17}$$

$$(XX^T)^{-1} = \begin{pmatrix} -1 & 3 & 1 & 5 \\ 4 & -1 & 2 & -3 \end{pmatrix} \begin{pmatrix} -1 & 4 \\ 3 & -1 \\ 1 & 2 \\ 5 & -3 \end{pmatrix} = \begin{pmatrix} 36 & -20 \\ -20 & 30 \end{pmatrix}^{-1} == \begin{pmatrix} 0.0441 & 0.0294 \\ 0.0294 & 0.0529 \end{pmatrix} \tag{18}$$

$$X^+ = \begin{pmatrix} 0.0441 & 0.0294 \\ 0.0294 & 0.0529 \end{pmatrix} \begin{pmatrix} -1 & 3 & 1 & 5 \\ 4 & -1 & 2 & -3 \end{pmatrix} = \begin{pmatrix} 0.0735 & 0.1029 & 0.1029 & 0.1324 \\ 0.1824 & 0.0353 & 0.1353 & -0.0118 \end{pmatrix} \tag{19}$$

Poznámka: Jednotlivé vzorce obsahujú vektor vstupov X v pôvodnom znení a niekde v transponovanom tvare. Záleží od toho ako zapíšeme vstupné vektory. Je to len vec zápisu.

Keď už máme spočítanú pseudoinverznú maticu, výsledná váhová matica vznikne súčinom pôvodných vstupov a pseudoinverznej matice:

$$W = X^T X^+ = \begin{pmatrix} -1 & 4 \\ 3 & -1 \\ 1 & 2 \\ 5 & -3 \end{pmatrix} \begin{pmatrix} 0.0735 & 0.1029 & 0.1029 & 0.1324 \\ 0.1824 & 0.0353 & 0.1353 & -0.0118 \end{pmatrix} = \begin{pmatrix} 0.6559 & 0.0382 & 0.4382 & -0.1794 \\ 0.0382 & 0.2735 & 0.1735 & 0.4088 \\ 0.4382 & 0.1735 & 0.3735 & 0.1088 \\ -0.1794 & 0.4088 & 0.1088 & 0.6971 \end{pmatrix} \tag{20}$$

Teraz, keď už máme váhovú maticu môžeme otestovať výstupy pre rôzne vstupy. Neurónová sieť je naučená na dané vstupy $\vec{x}^{(1)}$ a $\vec{x}^{(2)}$. Keďže sa jedná o autoasociačnú neurónovú sieť, tak to znamená, že ak zadáme na vstup $\vec{x}^{(1)}$, na výstupe musíme dostať ten istý vstup. Ale ak zadáme akýkoľvek iný vstup, nedostaneme na výstupe ten istý vstup, ale nejaký náhodný vektor, resp. vektor, ktorý sa blíži tým dvom vstupným vektorom, na ktorú sme sieť naučili rozpoznávať.

$$Y_1 = W\vec{x}^{(1)} = W \begin{pmatrix} -1 \\ 3 \\ 1 \\ 5 \end{pmatrix} = \begin{pmatrix} -1.0000 \\ 3.0000 \\ 1.0000 \\ 5.0000 \end{pmatrix} \tag{21}$$

$$Y_2 = W\vec{x}^{(2)} = W \begin{pmatrix} 4 \\ -1 \\ 2 \\ -3 \end{pmatrix} = \begin{pmatrix} 4.0000 \\ -1.0000 \\ 2.0000 \\ -3.0000 \end{pmatrix} \tag{22}$$

$$Y_3 = W\vec{x}^{(3)} = W \begin{pmatrix} 2 \\ 2 \\ 7 \\ -1 \end{pmatrix} = \begin{pmatrix} 4.6353 \\ 1.4294 \\ 3.7294 \\ 0.5235 \end{pmatrix} \tag{23}$$

Ak sa pokúsime zadať vektor blízky vektoru $\vec{x}^{(1)}$, sieť nám vráti podobný vektor k tomuto vektoru:

$$Y_4 = W\vec{x}^{(4)} = W \begin{pmatrix} -1 \\ 3.5 \\ 2 \\ 5.5 \end{pmatrix} = \begin{pmatrix} -0.6324 \\ 3.5147 \\ 1.5147 \\ 5.6618 \end{pmatrix} \tag{24}$$

V podstate sa jedná všeobecne o aproximáciu. Ak nenájdeme presný vzor, hľadáme približný. To je jedna z vlastností aj biologicky orientovaných systémov. A zároveň to je najzákladnejšia vlastnosť umelej neurónovej siete.

Takže ako je možné vidieť umelá neurónová sieť typu lineárneho autoasociátora sa "naučila" rozpoznávať vzory. Konkrétne dva vzory $\vec{x}^{(1)}$ a $\vec{x}^{(2)}$. Ak by sme chceli ďalšie vzory, museli by sme prepočítať váhovú maticu W s pridaným vzorom. Zaujímavou vlastnosťou je **šum**. Sieť nám pri nesprávnom vzore nenájde presný vzor, ale tzv. zašumený vzor v podobe vektora, ktorý sa blíži vektoru jedného z naučených vstupov - jedná sa o **aproximáciu**. To je jedna z vlastností aj biologicky orientovaných systémov. A zároveň to je najzákladnejšia vlastnosť umelej neurónovej siete. Samozrejme v reálnej praxi sa autoasociátor často používa na *rozpoznávanie vzorov*, obrázkov, a preto by vstupné vektory neobsahovali matice rozmerov 4×1, ale matice podstatne väčších rozmerov. Ako je možné vidieť umelé neurónové siete sú z praktického hľadiska záležitosťou matematiky. V tomto prípade sme si vystačili so základmi lineárnej algebry, ale na dôkladné pochopenie podstaty "učiaceho" procesu (teda dôvodu prečo to funguje) by boli potrebné jej hlbšie znalosti. Ostatné umelé neurónové siete používajú okrem lineárnej algebry aj matematickú analýzu a diferenciálne rovnice, čo je už podstatne ťažšie na pochopenie. Ale v princípe uvedený postup vystihuje hlavnú podstatu umelých neurónových sietí.

2 HOPFIELDOVE NEURÓNOVÉ SIETE

John Hopfield bol americký vedec, fyzik, tvorca *Hopfieldových neurónových sietí*. Jeho model predstavuje interpretáciu *Sherringtonovho-Kirkpatrickovho* a *Isingovho* modelu magnetika. Ide o model, ktorý je založený na fyzikálnom podklade tzv. **spinových skiel**. Pre ich zaujímavé vlastnosti sa ich fyzici snažili využiť na vytvorenie idealizovaných neurónových sietí, ktorých správanie je možné interpretovať ako analógie rôznych mozgových funkcií, akými sú napr. asociatívne vyvolávanie z pamäte, spracovanie časových postupností stimulov, zabúdanie a podobne. Hopfieldov model a jeho modifikácie patria medzi teoreticky najlepšie preštudované modely neurónových sietí. Jedná sa o najjednoduchšie *rekurentné* neurónové siete. Zmena synaptických váh sa realizuje podľa *Hebbovho pravidla*. Hopfieldove neurónové siete sa najčastejšie používajú ako asociačné pamäte na uchovávanie a vyvolávanie vzorov. Avšak medzi nevýhody Hopfieldových sietí patrí možnosť uchovania malého množstva vzoriek (`0.15N` vzoriek zo všetkých možných vzoriek). Hopfieldove neurónové siete sa často označujú aj ako *atraktorové* alebo autoasociatívne neurónové siete.

Tieto neurónové siete sa okrem spomenutých aplikácií používajú aj na riešenie mnohých optimalizačných problémov. V tomto prípade sa už jedná o určité dynamické systémy, kde sa ústredným pojmom stáva stabilita takejto siete. Aj preto sa nazývajú ako atraktorové siete, pričom atraktor je určitý konečný stav systému, v ktorom sa systém stabilizuje. V podstate stav systému reprezentujeme ako energiu, a keďže sa mení stav systému, mení sa aj energia a úlohou je hľadať rôzne príznaky siete v takto sa meniacom systéme. Táto problematika bude neskôr objasnená osobitným článkom.

V týchto sieťach môže byť neurón v jednom z dvoch stavov, t.j. $\vec{S_i} = \{-1, +1\}$. Nech N je celkový počet neurónov v sieti. Vstupný a zároveň aj výstupný stav siete je vyjadrený N-rozmerným binárnym vektorom $\vec{S} = (S_1, S_2, ..., S_N)$.

Váha synapsy J_{ij} *(Junction)* je tvorená j-tým neurónom na i-tom neuróne. Pre excitačnú synapsu platí, $J_{ij} > 0$ a pre inhibičnú synapsu $J_{ij} < 0$. Neurón na sebe samom synapsy netvorí, t.j. $J_{ii} = 0$. Suma príspevkov od jednotlivých neurónov $\vec{S_j}$, pričom $j = 1, ..., N$, váhovaná prostredníctvom synaptických váh J_{ij}, vyjadruje postsynaptický potenciál, ktorý je mierou *excitácie* i-teho neurónu. Postsynaptický potenciál sa zvykne označovať ako h_i^{int}:

$$h_i^{int} = \sum_{j=1}^{N} J_{ij} \vec{S_j} \tag{25}$$

Opäť sú to pojmy z biológie, ktoré sú v tomto zmysle idealizované už len preto, že v reálnych neurónových sieťach existujú elektrické ako aj chemické synapsie, pričom sa navzájom ovplyvňujú, kdežto v tomto prípade je to len diskrétny model postavený na tomto základe. **Synapsia** nie je nič iné ako kontakt medzi neurónmi a jej funkcia je taká istá ako pri živých organizmoch, a teda, *prenos nervového vzruchu*, v našom prípade zmena číselných hodnôt. Táto zmena sa môže udiať v kontakte na dvoch miestach tak, že ak si predstavíme kontakt ako štrbinu tak na jej začiatku (presynaptickej časti) je možná jedna zmena hodnoty a na konci (postsynaptickej časti) druhá zmena hodnoty.

Neurón sa aktivuje, t.j. bude generovať na svojom výstupe tzv. **akčný potenciál**, ak postsynaptický potenciál h_i^{int} prekročí istú hodnotu prahového napätia, tzv. **prah excitácie** neurónu. Akčný potenciál ako už z názvu vyplýva nie je nič iné ako realizácia akcie na základe určitého potenciálu. Táto veličina sa zvykne označovať ako h_i^{ext}. Celkový efektívny postsynaptický potenciál neurónu je potom $hi = h_i^{int} - h_i^{ext}$.

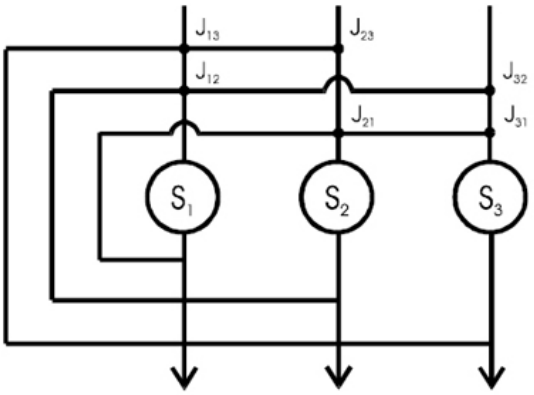
Pravidlo pre zmenu stavu *i*-tého neurónu, tzv. *prechodové pravidlo*, je dané vzťahom:

$$\vec{S_i} \to \vec{S_i}' = \mathrm{sign}(h_i) = \mathrm{sign}(\sum_{J=1}^{N} J_{ij}\vec{S_j} - h_i^{ext}) \tag{26}$$

pričom funkcia $\mathrm{sign}(x)$ je definovaná nasledovne:

$$\mathrm{sign}(x) = \begin{cases} +1 & , x > 0 \\ -1 & , x < 0 \end{cases} \tag{27}$$

Schéma Hopfieldovej neurónovej siete s dvojstavovými prvkami je na obrázku nižšie.



Stavy neurónov môžu byť $\vec{S_i} = \{-1, +1\}$ ako bolo spomenuté, vonkajšie pole kladieme obyčajne rovné nule, takže $h_i \neq 0$. Aktualizovanie stavu neurónov podľa vzťahu môže prebiehať dvoma spôsobmi. Prvým je synchrónna (paralelná) dynamika, keď všetky neuróny menia svoj stav naraz, t.j. v čase t platí:

$$\vec{S_i}(t) = \mathrm{sign}(\sum_{j=1, j\neq i}^{N} J_{ij}\vec{S_j}(t-1) - h_i^{ext}(t)) \tag{28}$$

Jeden cyklus relaxácie (prechod) odpovedá aktualizácii stavu všetkých N neurónov.

Druhou možnosťou je asynchrónna (sekvenčná) dynamika, keď v každom časovom momente t mení svoj stav len jeden náhodne vybraný neurón i:

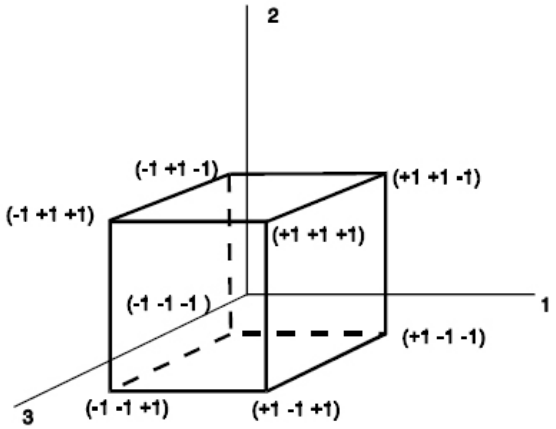
$$\vec{S_i}(t) = \mathrm{sign}(\sum_{j=1, j\neq i}^{N} J_{ij}\vec{S_j}(t) - h_i^{ext}(t)) \tag{29}$$

Energia danej konfigurácie aktivity S (hamiltonián systému) je definovaná ako:

$$E(\vec{S}) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} J_{ij}\vec{S_i}\vec{S_j} - \sum_{i=1}^{N}\vec{S_i}h_i^{ext} \tag{30}$$

Ak chceme, aby Hopfieldova sieť dospela ku globálnej stabilite tak energia musí byť záporná a musí prejavovať klesajúci charakter. Podrobnejšie to bude uvedené v jednom z budúcich

článkov. Na ilustráciu stavového priestoru pre Hopfieldovú sieť s tromi diskrétnymi neurónmi je na nasledujúcom obrázku.



Sieť sa môže nachádzať v jednom z 8 stavov, vrcholoch kocky.

Tieto modely sa už údajne v praxi veľmi nepoužívajú. Uplatňujú sa viac vo výskumných oblastiach. Najrozšírenejšou aplikáciou je oblasť modelovania neurobiologických a psychických javov, konkrétne ide o simuláciu nasledovných kognitívnych procesov:

- *rozpoznávanie* — vnímaný objekt už bol raz kedysi vnímaný, je v pamäti
- *asociatívne* vybavenie z pamäte — rekonštrukcia kompletnej položky (reprezentácie objektu) v pamäti, iniciovaná prítomnosťou fragmentu známeho vzoru na vstupe siete
- *klasifikácia* — konkrétny atraktor (resp. okolie atraktora) reprezentuje spracovaný a zapamätaný vzor

Používajú sa rôzne modifikácie týchto sietí. Ako už bolo spomenuté v úvode najčastejšie sa používaú ako **asociačné pamäte** pre uchovávanie a vyvolávanie vzorov.

3 LINEÁRNE MODELY NEURÓNOVÝCH SIETÍ

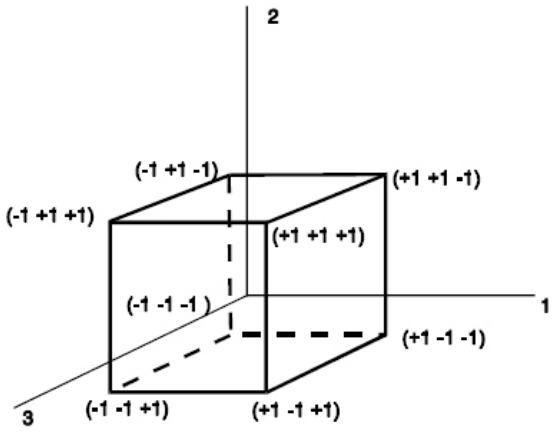
Asociačná pamäť neurónovej siete je sieť, v ktorej sú váhové hodnoty nastavené takým spôsobom, aby si bola schopná zapamätať množinu asociačných vzorov (vektorov). Ako bolo uvedené v predošlých článkoch o neurónových sieťach, jedná sa o váhovú maticu, ktorej vhodnými matematickými úpravami získami maticu, ktorá uchováva určitý počet vzorov. Je známe, že počet týchto vzorov je ohraničený, teda nie je možné uložiť do váhovej matice veľmi veľký počet vzorov.

Atraktor je stav siete, ktorý sa opakuje, je to teda tzv. **stacionárny stav siete**. Stacionárny stav siete predstavujú *pamäťové stavy siete*. To, ktorý stav siete je atraktorom, čiže jej pamäťovým stavom, je dané váhovou maticou, v tomto prípade označené ako J (J-junction). Teda vo váhovej matici sú uložené atraktory, pamäťové stavy alebo stabilné stavy siete. Atraktory môžu byť *pravdivé* a *falošné*. **Pravdivý atraktor** je taký, ktorý skutočne existuje vo váhovej matici. Ak sa jedná o autoasociačnú neurónovú sieť, tak to, čo máme na vstupe sa musí objaviť na výstupe. Vstup sa môže nachádzať aj v zašumenej podobe, neúplnej, pretože to je podstata autoasociácie, kedy sieť nájde na základe neúplneho vzoru jej úplny vzor. Takže, ak dáme na vstup hoci aj neúplný vzor, sieť by mala poskytnúť pamäťový vzor, ktorý je skutočne uložený vo váhovej matici. Ak ho poskytne, ide o pravdivý atraktor, v opačnom prípade, ide o **falošný (klamlivý) atraktor**. Pri Hopfieldových sieťach bolo dôležité, že sa dajú navrhnúť tak, aby sme apriori poznali všetky pamäťové stavy siete. Vzorec na konštrukciu váhovej matice je nasledujúci:

$$J_{ij} = \frac{1}{N} \sum_{\mu=1}^{p} \vec{\xi}_i^{\mu} \vec{\xi}_j^{\mu} \tag{31}$$

Vektory $\vec{\xi}$ predstavujú jednotlivé pamäťové stavy, pričom ich celkový počet je p. J_{ij} môže nadobudnúť $2p + 1$ rôznych hodnôt z intervalu $\langle \frac{-p}{N}, \frac{+p}{N} \rangle$. Tento predpis je vyjadrením jednej varianty tzv. **Hebbovho pravidla** pre zmenu synaptických váh. Hebbovo pravidlo hovorí, že váha synapsy rastie, ak oba neuróny spojené touto synapsou sú zároveň aktívne, a naopak váha synapsy klesá, ak je aktivita týchto dvoch neurónov nekorelovaná (nie je synchrónna).

Ak máme sieť s tromi diskrétnymi neurónmi, sieť sa môže nachádzať v jednom z 8 stavov, vrcholoch kocky, tak ako je znázornené na obrázku nižšie (a ako už bolo uvedené pri Hopfieldovej neurónovej sieti).



Na to, aby sme vyšetrili všetky stabilné stavy siete potrebujeme mať najskôr vopred dané pamäťové vzory a odpovedajúcu váhovú maticu. Vzorec pre výpočet váhovej matice poznáme, takže pre 2 ľubovoľné pamäťové vzory bude váhová matica vyzerať nasledovne:

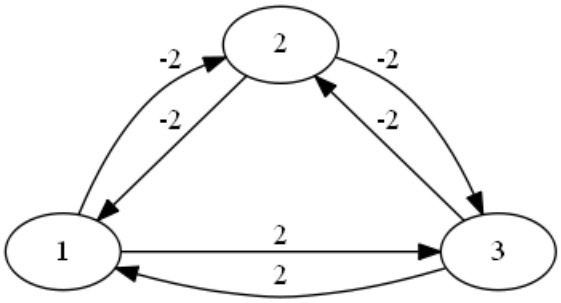
$$\vec{\xi}_1 = (1,-1,1)^T; \vec{\xi}_2 = (-1,1,-1)^T$$

$$J = \frac{1}{N}\sum_{i=1}^{p} \vec{\xi}_i^{\mu}\vec{\xi}_j^{\mu} - \frac{p}{N}I$$

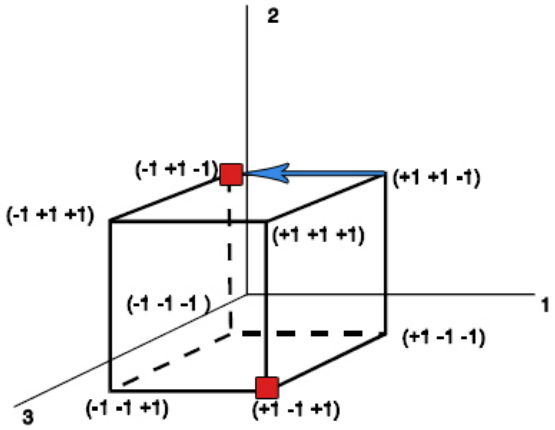
$$J = \frac{1}{3}\begin{pmatrix}1\\-1\\1\end{pmatrix}\begin{pmatrix}1 & -1 & 1\end{pmatrix} + \frac{1}{3}\begin{pmatrix}-1\\1\\-1\end{pmatrix}\begin{pmatrix}-1 & 1 & -1\end{pmatrix} - \frac{2}{3}\begin{pmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{pmatrix}$$

$$J = \begin{pmatrix}0 & -2 & 2\\-2 & 0 & -2\\0 & -2 & 0\end{pmatrix}$$

Vo vzorci pre váhovú maticu pribudol oproti pôvodnému vzorcu pre výpočet J určitý výraz v tvare $\left(-\frac{p}{N}\right)I$. Do tohto výrazu dosadzujeme počet pamäťových vzorov (p) a celkový počet neurónov (N) (odtiaľ $\frac{2}{3}$). Tento pomer násobíme jednotkovou maticou I. Takto napokon získame konečnú váhovú maticu J s dvomi pamäťovými stavmi $\vec{\xi}_1$ a $\vec{\xi}_2$. Túto váhovú maticu je možné nakresliť vo forme grafu nasledovne:



Na základe tejto váhovej matice môžme hľadať stabilné stavy siete - atraktory. Ak by nám niekto zadal len váhovú maticu a nepovedal by nám aké pamäťové vzory obsahuje, museli by sme vyšetrovať všetky stavy a nájsť stabilné stavy siete (v prípade, že by vôbec nejaké existovali). Všetky stavy siete pre $N = 3$ sú nasledovné:



Červeným štvorčekom sú vyznačené stabilné stavy siete, sú to pamäťové stavy $\vec{\xi}_1$ a $\vec{\xi}_2$. O atraktoroch sa zvykne hovoriť, že priťahujú určité stavy. V tomto prípade, ak dáme vektor $\vec{x}^{(0)} = (1, 1, -1)$, bude pritiahnutý atraktorom v smere modrej šípky k atraktoru $(-1, 1, -1)$, čo je možné spočítať nasledovným spôsobom:

$$\vec{x}^{(0)} = (1, 1, -1)$$

$$\vec{x}^{(2)} = \operatorname{sign}\left(\sum_{j=1}^{3} J_{ij}\vec{x}^{(0)}\right) = \operatorname{sign}\left(\frac{1}{3}\begin{pmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 0 & -2 & 0 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix}\right) =$$

$$= \operatorname{sign}\left(\frac{1}{3}\begin{pmatrix} -4 \\ 0 \\ 0 \end{pmatrix}\right) = \begin{pmatrix} -1 \\ 1 \\ -1 \end{pmatrix}$$

$$\vec{x}^{(2)} = \operatorname{sign}\left(\sum_{j=1}^{3} J_{ij}\vec{x}^{(1)}\right) = \operatorname{sign}\left(\frac{1}{3}\begin{pmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 0 & -2 & 0 \end{pmatrix}\begin{pmatrix} -1 \\ 1 \\ -1 \end{pmatrix}\right) =$$

$$= \operatorname{sign}\left(\frac{1}{3}\begin{pmatrix} -4 \\ 4 \\ -4 \end{pmatrix}\right) = \begin{pmatrix} -1 \\ 1 \\ -1 \end{pmatrix}$$

Poznámka: Pri vyhodnocovaní funkcie $\operatorname{sign}(x)$ *treba uvažovať aj prípad, ak* $x = 0$. *V tomto prípade nevykonáme žiadnu zmenu, ponecháme pôvodný stav.*

Mohli sme začať s ľubovoľným vektorom. Zvolili sme si $\vec{x}^{(0)} = (1, 1, -1)$. V prvej iterácii sme sa dostali do stavu $\vec{x}^{(1)} = (-1, 1, -1)$. Vieme, že tento stav je atraktor, to znamená, že v ďalšej interácii by sa stav nemal zmeniť. Ako je možné vidieť stav $\vec{x}^{(2)} = (-1, 1, -1)$, teda naozaj sa nezmenil. Ak by sme si zvolili iný počiatočný vektor, stav, dostali by sme sa do iného stavu. Z každého stavu sa vieme dostať buď do iného stavu, alebo do toho istého stavu. Ak sa dostaneme po iterácii do toho istého stavu, môže ísť o klamlivý atraktor. *Ale nie vždy*. Pretože, ak vieme, že v nejakom stave je pamäťový stav a my začneme v tomto stave, tak sa dostaneme do toho istého stavu. Aj v našom prípade, ak by sme ako počiatočný stav zadali priamo jeden z atraktorov, dostaneme sa hneď v prvej iterácii do toho istého stavu, čo zodpovedá ozajstným

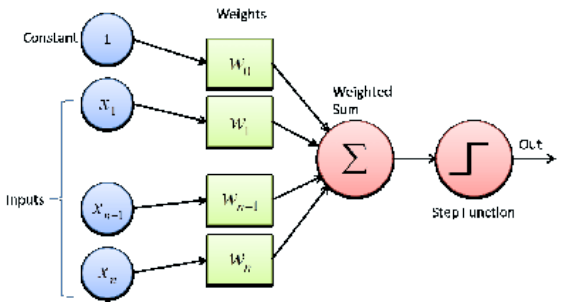
atraktorom. Jedným zo spôsobov ako zistiť, či ide o skutočný atraktor je ten, že atraktor by mal priťahovať viac stavov, určite nie len jeden, alebo žiadny. Pri vyšetrovaní stavov sa teda môže stať, že z niektorého stavu sa nedostaneme do žiadného iného stavu - tzv. **izolovaný stav**.

Ako pamäťový stav si je možné predstaviť nejaký vzor, obrázok. Na vstup prikladáme nejaké vzory, často neúplné, a sieť sa ich snaží rozpoznať. Akýkoľvek vzor bude pozostávať z vektorov, teda obrázok by pozostával z matice, kde každý bod (pixel) obrázku by predstavoval jeden neurón v neurónovej sieti. Je zrejmé, že obrázky obsahujú veľký počet bodov, takže výpočtová náročnosť takýchto neurónových sietí môže byť značná. Práve preto sa považujú umelé neurónové siete za jedny z výpočtovo najnáročnejších systémov. Konkrétna implementácia niektorých typov umelých neurónových sietí bude uvedená v nasledujúcich článkoch.

4 IMPLEMENTÁCIA NEURÓNOVÝCH SIETÍ: PERCEPTRÓN

Perceptrón je najjednoduchšia umelá neurónová sieť, ktorú navrhol v roku 1957 americký psychológ Frank Rosenblatt.

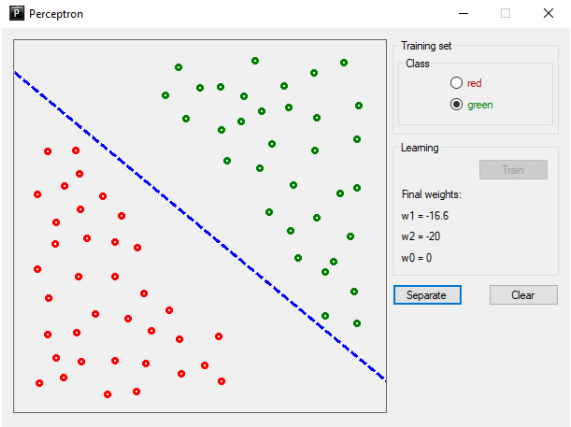
Jedná sa o **jednovrstvovú neurónovú sieť**, kde jedna vrstva pozostáva z n-vstupov $(x_1, x_2, ..., x_n)$ a druhá vrstva z jedného výstupu (y). Výstup je spojený s každým vstupom a pomyselne je takéto spojenie známe z grafových štruktúr z matematiky, kde každé spojenie môže podliehať tzv. **váhovému označeniu**, ktoré predstavuje váhovú hodnotu $(w_1, w_2, ..., w_n)$. Schematické znázorenie na obrázku nižšie reprezentuje jednoduchý perceptrón, pričom jednotlivé spojenia znázornené úsečkami (v našom prípade predstavujúce jednotlivé váhové prepojenia) sú aplikovateľné aj v iných vedných a technických disciplínach, napr. elektrotechnike, kde takéto prepojenia ilustrujú signály vo vodičoch a podobne.



Signál prenášaný vstupnými neurónmi je buď *binárny* $(0, 1)$ alebo *bipolárny* $(-1, 0|1)$. Výstup z perceptrónu je potom jednoduchá funkcia: $y = f(y_{in})$, pričom funkcia sa nazýva aktivačná funkcia, ktorá môže mať rôzne tvary (akákoľvek diferencovateľná funkcia, najčastejšie *sigmoidálna* (logistická) funkcia).

Aktivačná funkcia je v zásade určitá brána, alebo hranica, kde dochádza k nejakej zmene, v našom prípade k *zmene* váhových hodnôt. Samotné váhové hodnoty sa menia v čase, teda neurónová sieť sa týmto procesom adaptuje a "učí". Za naučenú neurónovú sieť je možné považovať také hodnoty váhových prepojení, kedy nám neurónová sieť poskytuje požadovaný výstup.Zmeny váhových stavov sa realizujú v čase, postupne, podľa určitého algoritmu. Pre perceptrón existuje tzv. *perceptrónový algoritmus*.

Výsledná aplikácia perceptrónu, ktorá separuje dve dátové množiny vyzerá nasledovne:



Jedná sa o aplikáciu naprogramovanú v jazyku .NET C#, typu WinForms. V nasledujúcej časti budú popísané niektoré dôležité časti programu.

Hlavná časť programu pozostáva z výpočtu váhových prepojení, tzn. *výpočet trénovacej množniny*:

```
double output;

for (int i = 0; i < num_weights; i++)
{
        weights[i] = 0.0;
}

while (stop)
{
        stop = false;

        for (int i = 0; i < train_exp.Count; i++)
        {
                output = result(i);

                if (train_exp[i] != (int)output)
                {
                        weights[0] += ro * train_exp[i] * train_x[i];
                        weights[1] += ro * train_exp[i] * train_y[i];
                        weights[2] += ro * train_exp[i];

                        stop = false;
                }
        }
}

lblW1.Text = "w1 = " + weights[0];
lblW2.Text = "w2 = " + weights[1];
lblW0.Text = "w0 = " + weights[2];
```

Funkcia `Result()`:

```
double result(int set)
{
        double res;

        res = ((train_x[set] * weights[0]) + (train_y[set] * weights[1]) + (1.0 * wei

        if (res > 0.0)
        {
                res = 1.0;
        }
        else
        {
                res = -1.0;
        }

        return res;
}
```

Pri definovaní jednotlivých bodov, červenej a zelenej množiny sa vykreslí separačná priamka:

```
Pen p = new Pen(Color.Blue, 3);
p.DashStyle = System.Drawing.Drawing2D.DashStyle.Dash;

btnTrain.Enabled = false;

float y1 = -((float)weights[0] / (float)weights[1]) * (-200) - ((float)weights[2] / (
float y2 = -((float)weights[0] / (float)weights[1]) * (200) - ((float)weights[2] / (fl

if (y1 < 0 && y2 < 0)
{
        g.DrawLine(p, 0, 200 + Math.Abs(y1), 400, 200 + Math.Abs(y2));
}
if (y1 < 0 && y2 > 0)
{
        g.DrawLine(p, 0, 200 + Math.Abs(y1), 400, 200 - y2);
}
if (y1 > 0 && y2 < 0)
{
        g.DrawLine(p, 0, 200 - y1, 400, 200 + Math.Abs(y2));
}
if (y1 > 0 && y2 > 0)
{
        g.DrawLine(p, 0, 200 - y1, 400, 200 - y2);
}

this.Controls.Add(pctBox);
pctBox.Image = bmp;
```

Na záver časť programu, ktorá slúži na interakciu kresliacej plochy a definovanie jednotlivých množín:

```
//Red class
if (rbtnRed.Checked == true)
{
        if (e.X >= 0 && e.X <= 200)
        {
                //I. kvadrant
                if (e.Y >= 0 && e.Y <= 200)
                {
                        train_x.Add(e.X - 200);
                        train_y.Add(200 - e.Y);
                        train_exp.Add(1);

                        g.DrawEllipse(new Pen(Color.Red, 3), e.X, e.Y, 5, 5);

                        this.Controls.Add(pctBox);
                        pctBox.Image = bmp;
                }

                //III. kvadrant
                if (e.Y >= 200 && e.Y <= 400)
                {
                        train_x.Add(e.X - 200);
                        train_y.Add(e.Y - 200);
                        train_exp.Add(1);

                        g.DrawEllipse(new Pen(Color.Red, 3), e.X, e.Y, 5, 5);

                        this.Controls.Add(pctBox);
                        pctBox.Image = bmp;
                }
        }

        if (e.X >= 200 && e.X <= 400)
        {
                //II. kvadrant
                if (e.Y >= 0 && e.Y <= 200)
                {
                        train_x.Add(e.X - 200);
                        train_y.Add(200 - e.Y);
                        train_exp.Add(1);

                        g.DrawEllipse(new Pen(Color.Red, 3), e.X, e.Y, 5, 5);

                        this.Controls.Add(pctBox);
```

```
                pctBox.Image = bmp;
            }

            //IV.kvadrant
            if (e.Y >= 200 && e.Y <= 400)
            {
                train_x.Add(e.X - 200);
                train_y.Add(e.Y - 200);
                train_exp.Add(1);

                g.DrawEllipse(new Pen(Color.Red, 3), e.X, e.Y, 5, 5);

                this.Controls.Add(pctBox);
                pctBox.Image = bmp;
            }
        }
    }

    //Green class
    if (rbtnGreen.Checked == true)
    {
        if (e.X >= 0 && e.X <= 200)
        {
            //I.kvadrant
            if (e.Y >= 0 && e.Y <= 200)
            {
                train_x.Add(e.X - 200);
                train_y.Add(200 - e.Y);
                train_exp.Add(-1);

                g.DrawEllipse(new Pen(Color.Green, 3), e.X, e.Y, 5, 5);

                this.Controls.Add(pctBox);
                pctBox.Image = bmp;
            }

            //III.kvadrant
            if (e.Y >= 200 && e.Y <= 400)
            {
                train_x.Add(e.X - 200);
                train_y.Add(e.Y - 200);
                train_exp.Add(-1);

                g.DrawEllipse(new Pen(Color.Green, 3), e.X, e.Y, 5, 5);

                this.Controls.Add(pctBox);
                pctBox.Image = bmp;
```

```
                }
        }

        if (e.X >= 200 && e.X <= 400)
        {
                //II. kvadrant
                if (e.Y >= 0 && e.Y <= 200)
                {
                        train_x.Add(e.X - 200);
                        train_y.Add(200 - e.Y);
                        train_exp.Add(-1);

                        g.DrawEllipse(new Pen(Color.Green, 3), e.X, e.Y, 5, 5);

                        this.Controls.Add(pctBox);
                        pctBox.Image = bmp;
                }

                //IV. kvadrant
                if (e.Y >= 200 && e.Y <= 400)
                {
                        train_x.Add(e.X - 200);
                        train_y.Add(e.Y - 200);
                        train_exp.Add(-1);

                        g.DrawEllipse(new Pen(Color.Green, 3), e.X, e.Y, 5, 5);

                        this.Controls.Add(pctBox);
                        pctBox.Image = bmp;
                }
        }
}

btnTrain.Enabled = true;
```

Týmto spôsobom bola prezentovaná jednoduchá neurónová sieť, ktorá slúži na separovanie dvoch množín jednou priamkou. Z toho vyplýva, že nie je možné separovať zložitejšie množiny, napr. množiny pre ktoré je nutné použiť viac priamok jednotlivo, resp. spoločne vo forme polygónu alebo iné geometricé útvary. Jedná sa primárne o tzv. *XOR problém*, kedy na separovanie je potrebné použiť ešte jednu neurónovú vrstvu, tzv. **skrytú vrstvu**.

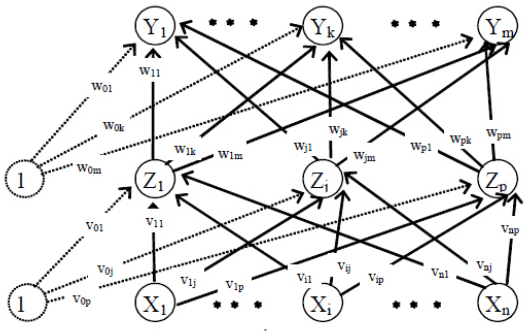
Neurónové siete obsahujúcu skrytú vrstvu je možné považovať za zložitejšie typy neurónových sietí, ktoré budú predmetom ďalšieho článku pojednávajúceho o učení typu *Backpropagation*.

5 IMPLEMENTÁCIA NEURÓNOVÝCH SIETÍ: BACKPROPAGATION

Jednovrstvové neurónové siete nachádzajú svoje uplatnenie v rôznych problémoch, najmä však v problematike rozpoznávania vzorov. Perceptrón slúži ako jednoduchá neurónová sieť, ktorá dokáže separovať dve a (po modifikácii) viac množín. Nemusí sa pritom jednať priamo o množiny uvedeného typu a principiálne ani o separovateľné problémy. Je možné to aplikovať na podobné úlohy rozpoznávania aké boli popísané v predchádzajúcich článkoch ohľadne neurónových sietí.

V každom prípade, jednovrstvové neurónové siete vykazujú pomerne nízku efektivitu a v spojitosti s tým sa okruh realizovateľných úloh rapídne zmenšuje. Viacvrstvové neurónové siete (vrátane neurónovej siete typu Perceptrón so skrytou vrstvou) je možné aplikovať oveľa efektívnejšie na rôzne druhy úloh, avšak najčastejšie sa používajú na riešenie rozpoznávania vzorov (*Pattern Recognition*).

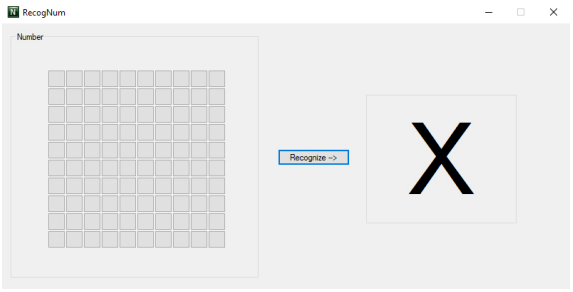
Existuje množstvo rôznych typov viacvrstvových neurónových sietí, avšak v našom prípade sa budeme zaoberať viacvrstvovou neurónovou sieťou s adaptačným algoritmom *Backpropagation*, tzn. spätné šírenie chyby. Keďže sa jedná o viacvrstvovú neurónovú sieť, tak pochopiteľne takáto sieť musí obsahovať minimálne dve vrstvy. Konkrétne pre túto sieť budeme potrebovať dokonca až tri vrstvy. Čo sa týka vrstiev ako takých, tak všeobecne sa používajú väčšinou tri až štyri vrstvy, tzn. jedna je vstupná vrstva, jedna je výstupná vrstva a jedna alebo dve vrstvy sú tzv. **skryté vrstvy** (hidden layers) (na obrázku označenie $(z_1, z_2, ..., z_n)$).



V zásade sa jedná o takú istú vrstvu ako ostatné vrstvy len s tým rozdielom, že skrytá vrstva musí byť s ktoroukoľvek ďalšou vrstvou spojená spôsobom každý s každým. Skrytá vrstva slúži na efektívnejší adaptačný proces, tzn. neurónová sieť sa bude efektívnejšie učiť a poskytovať tak lepšie výsledky. Navyše komplikovanejšie úlohy si priamo vyžadujú tieto skyrté vrstvy, pretože veľakrát bez takýchto vrstiev nie je možné daný problém vyriešiť. Neurónové siete ako také sú do veľkej miery rýdzo matematický problém, konkrétne sa jedná o oblasť numerickej a štatistickej matematiky, takže všetky výpočty poukazujú na pravdepodobnostné problémy. Z toho vyplýva, že matematický popis neurónových sietí je možné klasifikovať ako problém minimalizácie nejakej funkcie. Ak si predstavíme učenie vo všeobecnosti, tak človek sa učí do určitej miery formou pokusu a omylu a z toho vyplýva, že pri takomto type učenia vznikajú chyby učenia a následne tieto chyby sú častejším opakovaním minimalizované. Podobný princíp je založený na celých neurónových sieťach a konkrétne v našom prípade sa bude jednať o sofistikované riešenie minimalizácie chýb.

Keďže sme spomínali, že neurónové siete sú vhodné na rozpoznávanie obrazov (vzorov), tak v tomto prípade si popíšeme metódu *Backpropagation* pri riešení rozpoznávania celých čísel.

Výsledný program bol napísaný v jazyku .NET C#, WinForms a vyzerá nasledovne:



Na obrázku, na ľavej strane zadávame vzor číslice, ktorú chceme rozpoznať a na pravej strane máme uvedenú rozpoznanú číslicu.

Vstupná vrstva neurónov bude pozostávať zo `100` neurónov, pretože vstupná matica pre rozpoznávanie číslic obsahuje pole `10x10` bodov, teda `1` neurón odpovedá `1` políčku.

Výstupná vrstva neurónov pozostáva z `10` neurónov, pretože sieť je naučená rozpoznávať `10` číslic, od `0` až `9`.

Skrytá vrstva obsahuje `50` neurónov. Počet týchto neurónov nie je striktne daný. Ako už bolo spomínané, závisí od daného prípadu a väčšinou sa určuje odhadom alebo testovaním. Existujú empirické vzorce podľa ktorých je možné vypočítať počet vrstiev topológie siete ako aj počet neurónov skrytej vrstvy, avšak jedná sa iba o predbežné odhady a doposiaľ prakticky neexistuje nejaký platný osvedčený predpis, vzorec, podľa ktorého by bolo možné presne spočítať počet požadovaných neurónov.

V nasledujúcej časti budú popísané niektoré dôležité časti.

Výstupná vrstva je jednoznačne daná a teda predstavuje už spomenuté číslice. To je možné napísať ako dvojrozmerné pole vektorov od `0` až `9`

```
//0
output[0, 0] = 1;
output[0, 1] = 0;
output[0, 2] = 0;
output[0, 3] = 0;
output[0, 4] = 0;
output[0, 5] = 0;
output[0, 6] = 0;
output[0, 7] = 0;
output[0, 8] = 0;
output[0, 9] = 0;

//1
output[1, 0] = 0;
output[1, 1] = 1;
output[1, 2] = 0;
output[1, 3] = 0;
output[1, 4] = 0;
```

```
output[1, 5] = 0;
output[1, 6] = 0;
output[1, 7] = 0;
output[1, 8] = 0;
output[1, 9] = 0;

.
.
.

//9
output[9, 0] = 0;
output[9, 1] = 0;
output[9, 2] = 0;
output[9, 3] = 0;
output[9, 4] = 0;
output[9, 5] = 0;
output[9, 6] = 0;
output[9, 7] = 0;
output[9, 8] = 0;
output[9, 9] = 1;
```

Vstupná vrstva a skrytá vrstva sa na začiatok naplní náhodnými číslami:

```
void initNetwork()
{
        int i, j;

        Random rand = new Random();

        inputs[inputNeurons] = 1.0;
        hidden[hiddenNeurons] = 1.0;

        for (j = 0; j < hiddenNeurons; j++)
        {
                for (i = 0; i < inputNeurons + 1; i++)
                {
                        w_h_i[j, i] = (rand.NextDouble() - 0.5);
                }
        }

        for (j = 0; j < outputNeurons; j++)
        {
                for (i = 0; i < hiddenNeurons + 1; i++)
                {
                        w_o_h[j, i] = (rand.NextDouble() - 0.5);
                }
        }
```

```
}
```

Pole $w_{h_i}[j,i]$ reprezentuje váhové hodnoty medzi skrytou a vstupnou vrstvou a pole $w_{o_h}[j,i]$ reprezentuje váhové hodnoty medzi výstupnou a skrytou vrstvou.

Výpočet jednotlivých váhových prepojení trénovacej množiny je nasledovný:

```
void feedForward()
{
        int i, j;

        for (i = 0; i < hiddenNeurons; i++)
        {
                hidden[i] = 0.0;

                for (j = 0; j < inputNeurons + 1; j++)
                {
                        hidden[i] += (w_h_i[i, j] * inputs[j]);
                }
                hidden[i] = sigmoid(hidden[i]);
        }


        for (i = 0; i < outputNeurons; i++)
        {
                outputs[i] = 0.0;

                for (j = 0; j < hiddenNeurons + 1; j++)
                {
                        outputs[i] += (w_o_h[i, j] * hidden[j]);
                }
                outputs[i] = sigmoid(outputs[i]);
        }
}
```

Pričom sigmoidálna funkcia má tvar:

```
double sigmoid(double x)
{
        return (1.0 / (1.0 + Math.Exp(-x)));
}
```

Samotný prepočet váhových prepojení sa uskutočňuje počas celého procesu viac-krát v závislosti od veľkosti chybovej odchýlky. Práve túto časť má na starosti programový kód po stlačení tlačidla `Recognize`:

```
Random randTrain = new Random();

double mse;
int test;
int zero = 0;
```

```
generateMap();
initNetwork();

do
{
        test = randTrain.Next(0, 10);

        setNetworkInputs(test, 0.0);
        feedForward();
        backpropagateError(test);
        mse = calculateMSE(test);

} while (mse > 0.001);

setNetworkInputsPattern();
feedForward();

for (int z = 0; z < inputNeurons; z++)
{
        zero += map[z];
}

if (zero > 0)
{
        lblOutNum.Text = classifier().ToString();
}
else
{
        lblOutNum.Text = "X";
}
```

V uvedenom kóde tvorí podstatnú časť cyklus, ktorým kontrolujeme danú chybovú odchýlku. Odchýlka sa prepočitáva nasledujúcim programovým kódom:

```
void backpropagateError(int test)
{
        int outE;
        int hidE;
        int inpE;
        double[] errOut = new double[outputNeurons];
        double[] errHid = new double[hiddenNeurons];

        for (outE = 0; outE < outputNeurons; outE++)
        {
                errOut[outE] = ((double)output[test, outE] - outputs[outE]) * sigmoid_
        }
```

```
        for (hidE = 0; hidE < hiddenNeurons; hidE++)
        {
                errHid[hidE] = 0.0;

                for (outE = 0; outE < outputNeurons; outE++)
                {
                        errHid[hidE] += errOut[outE] * w_o_h[outE, hidE];
                }
                errHid[hidE] *= sigmoid_d(hidden[hidE]);
        }

        for (outE = 0; outE < outputNeurons; outE++)
        {
                for (hidE = 0; hidE < hiddenNeurons; hidE++)
                {
                        w_o_h[outE, hidE] += RO * errOut[outE] * hidden[hidE];
                }
        }

        for (hidE = 0; hidE < hiddenNeurons; hidE++)
        {
                for (inpE = 0; inpE < inputNeurons + 1; inpE++)
                {
                        w_h_i[hidE, inpE] += RO * errHid[hidE] * inputs[inpE];
                }
        }
}
```

Výpočet *MSE*, tzv. **stredná štvorcová chyba** je daná vzťahom, $MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$, pričom $\hat{Y}_i$ je vektor predpokladaných hodnôt (v našom prípade jednotlivé výstupy číslic) a Y_i je vektor aktuálne pozorovaných výstupných hodnôt a tomu zodpovedná programový kód:

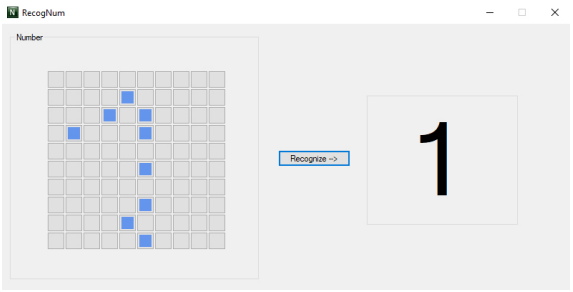
```
double calculateMSE(int test)
{
        double mse = 0.0;
        int i;

        for (i = 0; i < outputNeurons; i++)
        {
                mse += sqr(output[test, i] - outputs[i]);
        }

        return (mse / (double)i);
}
```

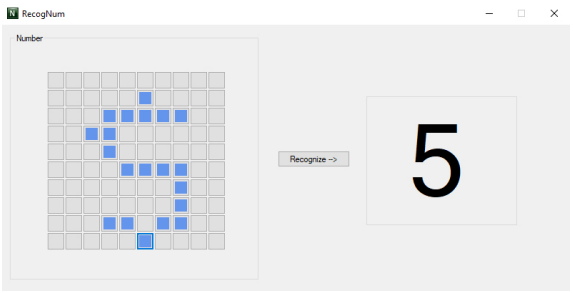
A pre koncový klasifikátor nasledovný kód:

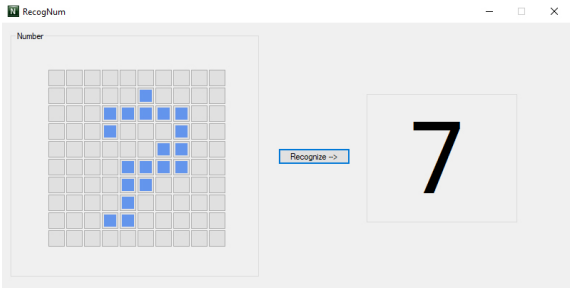
```
int classifier()
{
```

```
        int best = 0;
        double max;

        max = outputs[0];

        for (int i = 1; i < outputNeurons; i++)
        {
                if (outputs[i] > max)
                {
                        max = outputs[i];
                        best = i;
                }
        }
        return best;
}
```

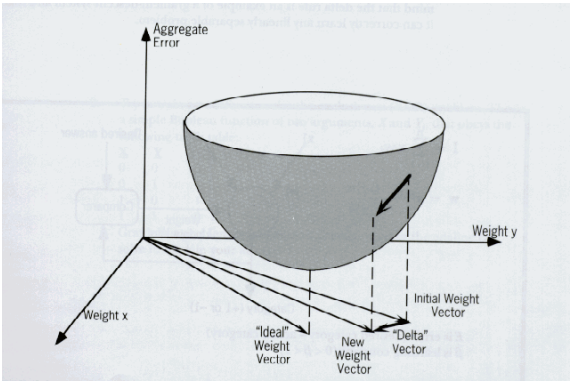
Niektoré ukážky výstupov pre jednotlivé číslice:







Chybovú funkciu je možné si predstaviť ako 3D povrch váhovými hodnotami na osi x a y a hodnotou chybovej funkcie na osi z. Týmto vznikne (ideálne) teleso zobrazené na obrázku nižšie. Postupne ako klesá hodnota chybovej funkcie prostredníctvom vektora, približujeme sa skutočnému, ideálnemu váhovému vektoru. Cieľom je, aby v každom kroku výpočtu vektor klesal smerom k tzv. **sedlu** (*sedlový bod*), čo predstavuje **lokálne minimum**.



Práve z toho dôvodu je nutné počítať so *skrytou vrstvou* neurónov a mala by byť navrhnutá tak, aby sa algoritmus nezacyklil, alebo aby váhový vektor nesmeroval mimo požadovaný sedlový bod. Existujú rôzne modifikácie tohto výpočtu, optimalizované na strmosť a rýchlosť vypočítaných hodnôt. Podrobne sa tým zaoberá už spomínaná *numerická matematika*, *optimalizačné metódy*, alebo *teória diferenciálnych rovníc*.