VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

KLASIFIKACE DAT S VYUŽITÍM UMĚLÝCH NEURONOVÝCH SÍTÍ
DATA CLASSIFICATION USING ARTIFICIAL NEURAL NETWORKS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Hana Gurecká**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Radomil Matoušek, Ph.D.**
SUPERVISOR

BRNO 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

Ústav:	Ústav automatizace a informatiky
Studentka:	**Hana Gurecká**
Studijní program:	Strojírenství
Studijní obor:	Aplikovaná informatika a řízení
Vedoucí práce:	**doc. Ing. Radomil Matoušek, Ph.D.**
Akademický rok:	2015/16

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Klasifikace dat s využitím umělých neuronových sítí

Stručná charakteristika problematiky úkolu:

Úloha klasifikace dat je široce studována v oblastech umělé inteligence, výpočetní inteligence, strojového učení, získávání dat apod. Tato bakalářská práce bude prezentovat některé klasifikační algoritmy založené na umělých neuronových sítích. K realizaci bude využito prostředí Matlab.

Cíle bakalářské práce:

1) Stručné seznámení s metodami umělých neuronových sítí v kontextu dále prezentovaných řešení.
2) Tvorba a popis aplikací demonstrujících řešení úloh: klasifikace Iris data set pomocí dopředné neuronové sítě (úloha identifikace kosatců), rozpoznávání vzorů momocí Hopfieldovy sítě, klasifikace Iris data set pomocí sítě SOM.
3) Rozprava k dosaženým výsledkům.

Seznam literatury:

[online, 1. 11. 2013]
http://neuroph.sourceforge.net/tutorials/FaceRecognition/FaceRecognitionUsingNeuralNetwork.html

S. Lawrence, C. Lee Giles, A.Ch. Tsoi, A.D. Back, Face (2013): Recognition: A Convolutional Neural Network Approach, IEEE Transactions on Neural Network [online, 1. 11. 2013]
http://clgiles.ist.psu.edu/papers/IEEE.TNN.face recognition.hybrid.nn.pdf

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2015/16

V Brně, dne

L. S.

doc. Ing. Radomil Matoušek, Ph.D. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

ABSTRAKT

Práce se zabývá neuronovými sítěmi využívanými ke klasifikaci dat. Teoretickou náplní práce je představení tří základních typů neuronových sítí využitelných ke klasifikaci dat. Těmito sítěmi jsou dopředná neuronová síť se zpětným šířením chyby, Hopfieldova síť s minimalizací energetické funkce a Kohonenova metoda samoorganizačních map. Ve druhé části práce jsou tyto algoritmy naprogramovány a testovány v prostředí Matlab. Na konci testování každé sítě jsou diskutovány výsledky.

ABSTRACT

The thesis deals with neural networks used in data classification. The theoretical part presents the three basic types of neural networks used in data classification. These networks are feedforward neural network with backpropagation algorithm, the Hopfield network with minimization of energy function and the Kohonen's method of self-organizing maps. In the second part of the thesis these algorithms are programmed and tested in Matlab environment. At the end of each network testing results are discussed.

KLÍČOVÁ SLOVA

Neuronové sítě, zpětné šíření chyby, Hopfieldova síť, Kohonenovy samoorganizační mapy, klasifikace dat

KEYWORDS

Neural networks, Backpropagation, Hopfield net, Kohonen self-organizing maps, data classification

BIBLIOGRAFICKÁ CITACE

GURECKÁ, H. *Klasifikace dat na bázi umělých neuronových sítí.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2016. 67 s. Vedoucí bakalářské práce doc. Ing. Radomil Matoušek, Ph.D.

PODĚKOVÁNÍ

Ráda bych poděkovala doc. Ing. Radomilu Matouškovi, Ph.D. za odborné vedení mé bakalářské práce, ochotu, pomoc a cenné rady, které mi poskytl během zpracování práce.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato práce je mým původním dílem, zpracovala jsem ji samostatně pod vedením doc. Ing. Radomila Matouška, Ph.D. a s použitím literatury uvedené v seznamu.

V Brně dne 27. 5. 2016 ……

Gurecká Hana

OBSAH

ÚVOD

Práce se zabývá možnostmi klasifikace dat za pomoci umělých neuronových sítí. Neuronovým sítím se v rámci posledního půlstoletí dostává velké pozornosti díky jejich schopnosti učení se (adaptace). Schopnost učení je inspirována posledními poznatky z oblasti neurofyziologie, kdy se snažíme napodobit lidskou schopnost učení pomocí sítě umělých neuronů. Díky tomu spadají koncepty neuronových sítí pod obor umělé inteligence.

Neuronových sítí se využívá v řadě odvětví, především pak jde o rozhodování či předpovídání vývoje. Konkrétních aplikací je již dnes nesčetné množství, jako příklad nám zde poslouží řídicí systémy, lékařství, předpovědi chování systémů, rozpoznávání řeči a písma. Již dnes neuronové sítě zažívají obrovský rozmach a jejich význam do budoucna jistě poroste.

Text se bude dělit na dva větší celky, kdy první bude představovat teoretický úvod do problematiky a popisovat základní typy neuronových sítí, které tvoří bázi širokého spektra dalších modelů sítí. Druhá část práce bude mít pak praktický charakter, kdy bude každý typ sítě naprogramován v prostředí Matlab a budou zkoumány jeho charakteristiky na konkrétních datech.

První kapitola bude sestavena jako obecné představení konceptu neuronových sítí. Na začátku bude pro úplnost textu zařazena stručná historie neuronových sítí. Pokračovat se bude vysvětlením základních pojmů, jako je umělý neuron, topologie sítí, typy učení sítí. Obecně tedy budeme hovořit o možnostech organizační, aktivní a adaptivní dynamiky.

Další kapitola se zaměří na obecně nejpoužívanější adaptační algoritmus zpětného šíření chyby, který je popsán pomocí třech uvedených dynamik. Mimo jiné bude zmíněna problematika počtu neuronů ve skrytých vrstvách a užité optimalizační metody využívající gradientní přístup.

V pořadí třetí kapitola představí Hopfieldovu síť určenou pro klasifikaci vzorů, popíše adaptivní dynamiku realizovanou za pomoci minimalizace energetické funkce a neopomene rozebrat úskalí, která jsou s Hopfieldovou sítí spojena, jako je spontánní vznik nepravých vzorů či kapacita sítě.

Poslední kapitola teoretické části práce bude věnována Kohonenově samoorganizační mapě, která stručně popíše způsob učení sítí bez učitele, převádění vícedimenzionálních dat na dvojrozměrný prostor a představí i možnosti vektorového kvantování učením.

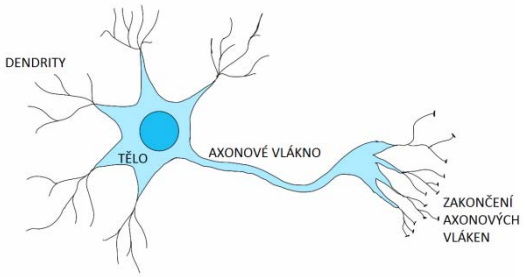
V praktické části budou na základě uvedených a popsaných skriptů vytvořeny sítě, které následně naučíme pomocí probíraných optimalizačních metod (dopředná neuronová síť se zpětným šířením chyby, Hopfieldova síť, Kohonenova samoorganizační mapa) a jejich výsledky zobrazíme.

Pro metodu zpětného šíření chyby a Kohonenovy mapy je v prostředí programu Matlab možné nastavit řadu parametrů, které mohou měnit vlastnosti sítě. V rámci praktické části proto budeme zkoumat i vlivy vybraných parametrů na klasifikační schopnosti sítě. Na závěr testování každé metody bude provedena diskuze nad získanými výsledky.

Všechny algoritmy, použitá data i obrázky výsledných grafů budou součástí elektronické přílohy.

1 ÚVOD DO NEURONOVÝCH SÍTÍ

Koncept umělých neuronových sítí byl inspirován a vytvořen na základě analogie s funkcí biologických neuronů nervového systému živočichů. Úkolem nervového systému, resp. centrální nervové soustavy je řídit organismus, přičemž tato činnost zahrnuje v technickém pohledu zpracování signálů pomocí základních výpočetních jednotek – neuronů (Obr. 1). Biologické neurony jsou mezi sebou propojeny vazbami, označenými jako tzv. synapse, čímž vytvářejí neuronovou síť. V tomto pohledu lze biologický neuron chápat jako výpočetní jednotku, která na základě vstupů (dendrity) realizuje výstupy (zakončení axonových vláken), přičemž v rámci neuronové sítě je míra propojení jednotlivých neuronů ovlivněna synapsemi. Snahou konceptu umělých neuronových sítí je napodobit funkcionalitu takovéto biologické neuronové sítě. Výsledkem jsou různé modely umělých neuronových sítí, které mohou více či méně korespondovat s kognitivními funkcemi mozku nebo realizovat rozličné matematické funkce a operace.



Obr. 1) Biologický neuron

1.1 Historie

Počátky umělých neuronových sítí, dále NN, se datují k roku 1943, kdy vědci Warren McCulloch a Walter Pitts publikovali článek "*A Logical Calculus of Ideas Immanent in Nervous Activity*". V tomto článku představili první umělý neuron s binárními vstupy a skokovou aktivační funkcí (binárním výstupem). Každý vstup neuronu nabýval hodnoty 0 (inhibiční), nebo 1 (excitační), následně byla vyhodnocena hodnota prahové funkce. Prvotní neuron tedy měl za úkol vyhodnocovat logické funkce [1].

V roce 1949 kanadský neuropsycholog Donald Hebb vydal knihu „*The Organization of Behavior*", která dodala umělým neuronovým synapsím učicí pravidlo [3]. To spočívá v úpravě vah na vstupech tak, že vstupy vedoucí ke korektní excitaci neuronu jsou posilovány, zatímco pokud je neuron excitován nesprávně, jsou vstupní hodnoty, které k jeho nesprávné excitaci vedly, oslabeny [2].

V roce 1951 byl sestaven první neuropočítač zvaný *Snark*, jehož autorem byl Marvin Minsky. Snark dokázal automaticky přizpůsobovat váhy a technicky byl velkým úspěchem. Do budoucna inspiroval vědce k sestavení dalších neuropočítačů, nicméně se nedá označit za užitečný, jelikož nebyl použit k řešení žádných praktických problémů[1].

Následně v roce 1957 Frank Rosenblatt zobecnil Pittsův a McCullochův model neuronu na tzv. *perceptron*. Perceptron je pevná architektura jednovrstvé neuronové sítě

s *m* vstupy a *n* výstupy, která počítá s reálnými čísly. Rosenblatt také navrhl nový učicí algoritmus, který při náhodné konfiguraci vstupních vah dokáže v konečném čase najít korektní váhový vektor. Díky tomuto výzkumu mezi roky 1957 a 1958 Rosenblatt společně s Charlesem Wightmanem navrhl neuropočítač Mark I Perceptron, jež byl určen k detekci znaků. Mark I Percepron sklidil obrovský úspěch, díky čemuž za nedlouho vznikl další typ neuropočítače ADALINE [3].

ADALINE byla zkratkou pro ADaptivní LINeární Element, jehož autorem byl Bernard Widrow s kolektivem svých studentů. ADALINE se od Mark I Preceptronu lišil především tím, že jeho vstupy byly obecně reálné a jednotlivé adaptivní lineární elementy realizovaly lineární funkci [3]. Rozjetý výzkum v oblasti neuronových sítí však v roce 1969 zaznamenalo krizi, jelikož Marvin Minsky účelově zdiskreditoval neuronové sítě na základě faktu, že jednovrstvá neuronová síť není schopna vyčíslit logickou funkci XOR. Sám tehdy prohlásil, že řešením by byla vícevrstvá neuronová síť, avšak předpokládal, že vzhledem ke složitosti takové struktury nebude možné najít vhodný učicí algoritmus. Díky tomuto prohlášení se prostředky i zájem vědců zaměřily na jiné oblasti. V roce 1986 však byl odvozen algoritmus, který využívá při učení principu zpětného šíření chyby-Backpropagation. Tento algoritmus uměl vyřešit i problém XOR a umožnil znovunastartování intenzivního bádání v oblasti neuronových sítí, které tvá dodnes [4].

1.2 Koncept neuronových sítí

Neuronových sítí s výhodou využíváme pro jejich schopnost klasifikace, regresní analýzy, predikce časových řad či shlukování. Jedním z pilířů neuronových sítí jsou statistické metody a pojmy z teorie grafů.

Obecně, dle povahy problému, je neuronová síť tvořena až stovkami samostatných jednotek, které nazýváme umělými neurony, dále jen *neurony*. Tyto neurony jsou organizovány ve vrstvách a mezi sebou vzájemně propojeny. Orientované propojení jednotlivých neuronů nazýváme *topologickým uspořádáním sítě*. Každý neuron obsahuje vážené vstupy (tedy vstupní vektor hodnot násobený vektorem příslušných váhových koeficientů), prahovou hodnotu, přenosovou funkci a jeden výstup. Konkrétní síť je určena architekturou sítě, přenosovou funkcí svých neuronů a pravidlem učení, resp. *organizační, aktivní a adaptivní dynamikou*.

Neuronové sítě považujeme díky konfigurovatelnosti *vah* učením za parametrizovaný systém. Výpočet odezvy neuronu obecně probíhá tak, že vážený součet vstupů (vstupy jsou vynásobeny odpovídajícími vahami) v porovnání s *prahovou hodnotou* určuje, jakou hodnotou je neuron aktivován, případně jestli se vůbec zaktivuje. Tento *aktivační signál* je vyhodnocen *přenosovou funkcí* a výsledek je výstupem celého neuronu. Během tréninku jsou váhy na jednotlivých spojích optimalizovány dle učicího pravidla a povahy řešené úlohy[5].

1.3 Organizační dynamika

Organizační dynamika představuje topologii neuronové sítě, jejíž názvosloví a struktura odpovídají tradicím teorie grafů. Pod pojmem topologie si můžeme představit architekturu sítě skládající se z jednotlivých neuronů, tedy počty vrstev v neuronové síti, počty neuronů v těchto vrstvách a v některých případech i geometrické rozložení neuronů ve vrstvě.

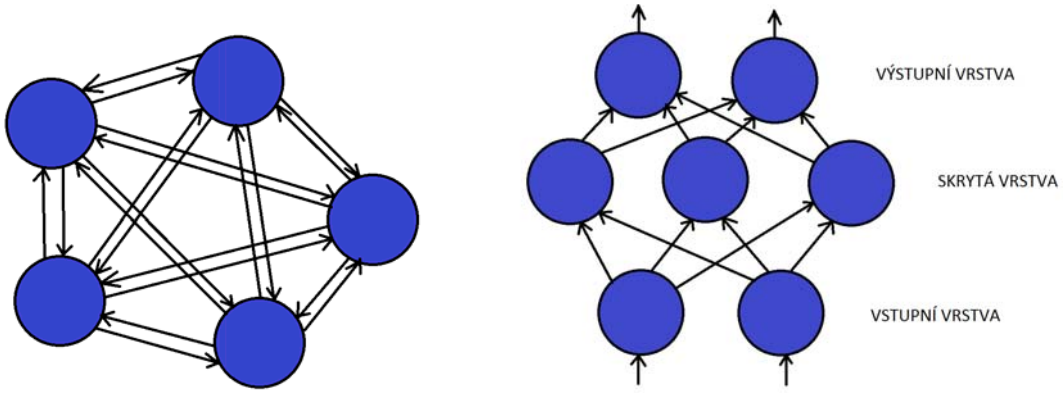
Topologie zůstává v naprosté většině případů neměnná. Než-li se však stanoví definitivní architektura, je třeba velmi pečlivě uvážit konkrétní účel sítě. V případě neměnných topologií můžeme sítě dělit na dva základní typy: *cyklické* (rekurentní) a *acyklické* (dopředné).

1.3.1 Cyklické (rekurentní) neuronové sítě

Za neuronovou síť s cyklickou topologií považujeme síť, ve které existuje skupina neuronů (nemusí se jednat o všechny neurony v síti), která je vzájemně *cyklicky propojena*. Cyklické spojení znamená, že jednotlivé neurony jsou vzájemně propojeny a výstup posledního neuronu je zároveň jedním ze vstupů vstupního (prvního) neuronu. Za elementární cyklus můžeme považovat zpětnou vazbu neuronu, kdy výstup daného neuronu je zároveň jeho vstupem. Za síť s *úplnou cyklickou topologií* můžeme označit síť, kde výstup každého neuronu je vstupem všech neuronů v síti (Obr.2a).

1.3.2 Acyklické (dopředné) neuronové sítě

Za acyklickou neuronovou síť povazujeme takovou síť, kde nenajdeme žádný cyklus. Jednotlivé spoje vedou pouze vpřed (odtud název dopředné). Typickým znakem dopředných sítí je, že vždy můžeme jednoznačně určit vrstvy, ze kterých se skládají. Za speciální případ považujeme vícevrstvou neuronovou síť, která obsahuje vstupní vrstvu, výstupní vrstvu a skrytou vrstvu (jednu nebo i více). Acyklickou neuronovou síť můžeme jednoznačně specifikovat počty neuronů v jednotlivých vrstvách. Na obrázku (Obr.2b) vidíme síť s dvěma vstupními neurony, dvěma neurony ve skryté vrstvě a jedním výstupním neuronem, můžeme ji tedy označit jako dopřednou síť 2-3-2 [6].



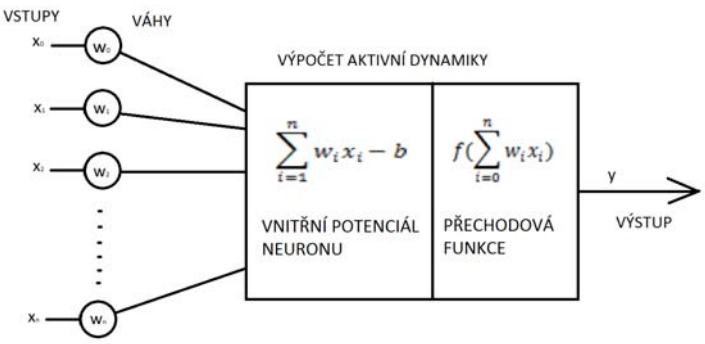
Obr. 2) a) Úplná cyklická topologie b) Topologie dopředné sítě

1.4 Aktivní dynamika

Aktivní dynamika popisuje funkci již naučené neuronové sítě o pevné topologii a konfiguraci váhových koeficientů. Nejdříve se nastavují hodnoty vstupních neuronů na vstupní hodnoty, které tvoří tzv. *vstupní prostor*. Následuje inicializace sítě, během které probíhá samotný výpočet. Obecně můžeme uvažovat spojitý průběh výpočtů, nicméně ve většině případů se jedná o taktovaný systém s diskrétním časem, kdy je vyhodnocení odezvy neuronů uskutečňováno v diskrétních krocích. Můžeme rozlišovat *sekvenční výpočet*, kdy je v každém časovém kroku aktualizován právě jeden neuron, nebo *paralelní výpočet*, kdy je v jednom čase zároveň *aktualizováno* více neuronů. Aktualizace představuje změnu stavu neuronů na základě hodnot na jejich vstupech. Ve vstupním prostoru jsou proto neurony aktualizovány

pouze na vstup, ovšem každý další navazující neuron je aktualizován na základě výstupu neuronů, které mu předchází. V aktivní dynamice rozlišujeme *synchronní* a *asynchronní* modely neuronových sítí podle toho, jestli je aktualizace řízena centrálně, nebo se jednotlivé neurony aktualizují nezávisle na jiných neuronech. Stav výstupních neuronů reprezentuje výstup celé sítě, který se ve většině případů ustálí, a je tedy pro dané vstupní hodnoty při dané topologii a konfiguraci konstantní [1].

Jak již bylo uvedeno, umělý neuron považujeme za stavební jednotku neuronových sítí. Signály, které do neuronu přicházejí, označujeme jako *vstupy*. Vstupy neuronu jsou obecně reprezentovány reálnými čísly. V průběhu aktivní dynamiky jsou nejdříve násobeny příslušnými váhovými koeficienty (taktéž reálnými čísly), sečteny, porovnány s prahovou hodnotou (čímž dostáváme tzv. vnitřní potenciál neuronu ξ) a následně zhodnoceny přechodovou funkcí, jejíž výsledek je výstupem neuronu [5]. Schéma umělého neuronu, které znázorňuje postup signálu neuronem, vidíme na Obr. 3). Jelikož vstupy i váhy jsou z oboru reálných čísel, mohou generovat jak kladné tak záporné hodnoty vnitřního potenciálu. Práh, též někdy označovaný bias *b,* je konstanta, která je někdy považována za nultý vstup neuronu s hodnotou 1 a váhou odpovídající *b*, která podléhá adaptaci v rámci fáze učení.



Obr. 3) Schéma umělého neuronu

Vnitřní potenciál neuronu můžeme vyčíslit následovně [1]:

$$\xi = \sum_{i=1}^{n} w_i x_i - b, \tag{1}$$

kde x_i představuje příslušnou vstupní hodnotu a w_i hodnotu váhy pro daný vstup z vektoru n vstupů. Prahová hodnota se formalizuje jako nultý vstup, čímž získáme vztah:

$$\xi = \sum_{i=0}^{n} w_i x_i. \tag{2}$$

Prahovou hodnotu ve vzorci představuje nultý vstup s hodnotou pro $x_0 = 1, w_0 = -b$. Vyčíslená hodnota vnitřního potenciálu se následně stává argumentem aktivační přenosové funkce (2). S využitím tzv. aktivační funkce $f(\xi)$ dostáváme celkovou odezvu neuronu:

$$y = f(\xi) = f(\sum_{i=0}^{n} w_i x_i). \tag{3}$$

Poznamenejme, že model neuronu definovaný dle (3) je označován jako *perceptron*.

1.4.1 Aktivační (přenosová) funkce

Aktivační funkce mohou být obecně jak lineární, tak nelineární. Tato funkce realizuje zobrazení $R^n \rightarrow R$ a určuje, zda je neuron inhibován či excitován. Tyto funkce se liší dle typu neuronové sítě. Svou povahou mohou být diferencovatelné i nediferencovatelné a omezené či neomezené v oboru hodnot. Původní model neuronové sítě s perceptronovým učicím pravidlem využíval skokovou aktivační funkci (4), případně (5). Lineární model ADALINE funkci (6) a obecné modely neuronové sítě funkce (6), (7) a (8) [4]. Poznamenejme, že uvedené modely aktivačních funkcí jsou frekventovaně používané, ovšem definic těchto funkcí může existovat nespočetně mnoho. V případě vícevrstvých neuronových sítí se běžně využívá více typů přenosových funkcí v rámci jedné sítě.

$$f(\xi) = \begin{cases} 1 \; pro \; \xi \geq 0 \\ 0 \; pro \; \xi < 0 \end{cases} \tag{4}$$

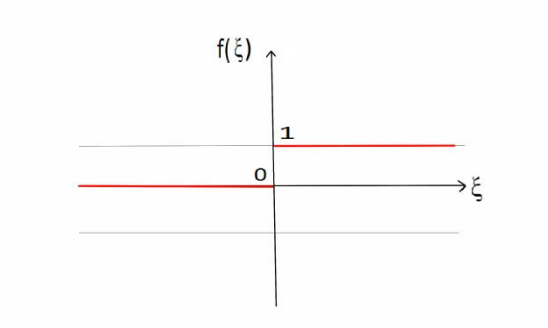
$$f(\xi) = \begin{cases} 1 \; pro \; \xi \geq 0 \\ -1 \; pro \; \xi < 0 \end{cases} \tag{5}$$

$$f(\xi) = \xi \tag{6}$$

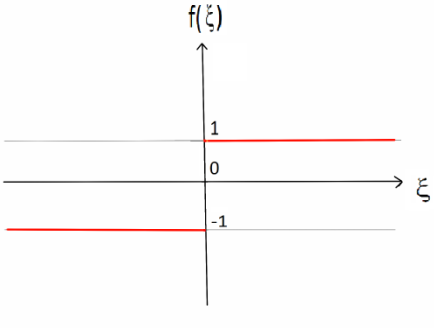
$$f(\xi) = \tanh\left(\frac{1}{2}\xi\right) = \frac{1 - e^{-\xi}}{1 + e^{-\xi}} \tag{7}$$

$$f(\xi) = \frac{1}{1 + e^{-\xi}} \tag{8}$$

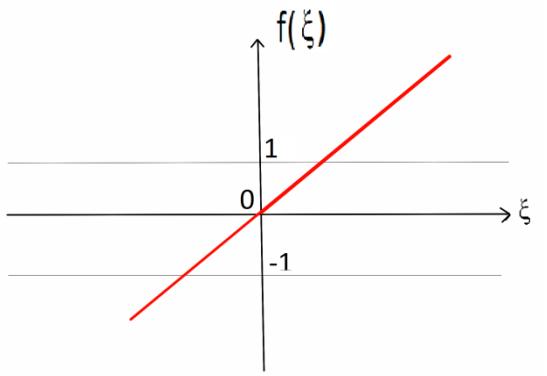
$$f(\xi) = \begin{cases} 1 \; pro \; \xi > 1 \\ \xi \; \text{pro} \; -1 \leq \xi \leq 1 \\ 0 \; pro \; \xi < -1 \end{cases} \tag{9}$$



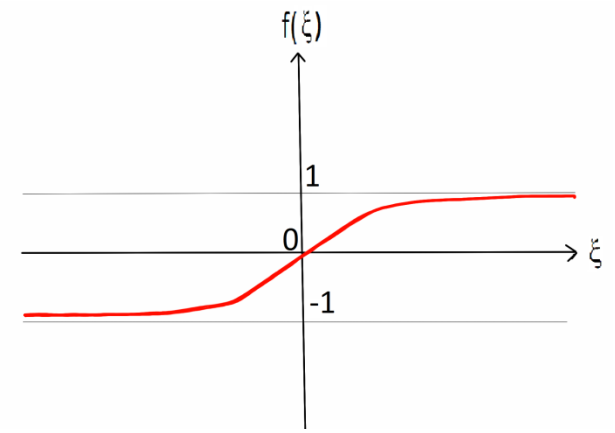
Obr. 4) Aktivační funkce typu: *jednotkový skok*



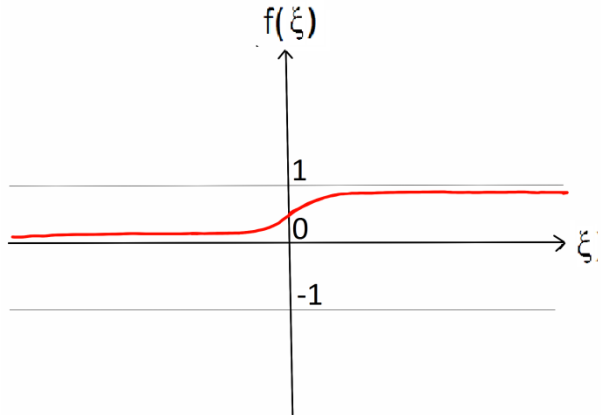
Obr. 5) Aktivační funkce typu: *signum*



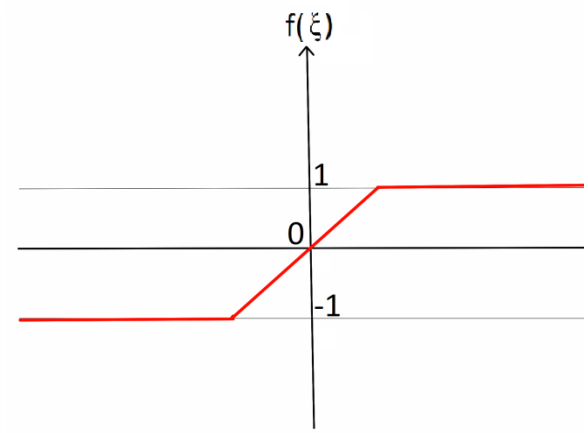
Obr. 6) Aktivační funkce typu: *linearita*



Obr. 7) Aktivační funkce typu: *hyperbolický tangens*



Obr. 8) Aktivační funkce typu: *logistická sigmoida*



Obr. 9) Aktivační funkce typu: *saturace* (*sat. linearita*)

1.5 Adaptivní dynamika - Učení neuronových sítí

Schopnost učit se je jedním ze základních znaků inteligence. Adaptivní dynamiku sítě představuje její učení, tedy úprava váhových koeficientů pomocí daného učicího pravidla. Na počátku mohou být váhy nastaveny náhodně, zpravidla na hodnoty blízké nule, následuje jejich úprava - učení. Adaptivní dynamika, obdobně jako u dynamiky aktivní, probíhá v posloupnosti diskrétních časových kroků [1].

Existují dva základní způsoby učení neuronových sítí. Jedná se o *učení s učitelem* a *učení bez učitele*. Při daných způsobech učení se vždy zabýváme parametry, které souvisí s učením ze vzorů [7]:

- Kapacita (*capacity*): kolik různých výstupů může síť rozeznávat, jaké funkce, rozhodnutí a v jakých rozmezích může síť vyhodnocovat.

- Obsáhlost vzorů (*sample comlpexity*): kolik tréninkových vzorků je potřeba, aby bylo zajištěno odpovídající zobecnění, příliš málo vzorků může způsobit problém zvaný „over-fitting“, kdy má síť dobré výsledky na tréninkové množině, ale špatné výsledky na ověřovací množině.
- Výpočetní náročnost (*Computational complexity*): kolik času je potřeba k naučení sítě. Řada učicích pravidel, která jsou používána v současné době má vysokou výpočetní náročnost, proto je hledání efektivnějších učicích pravidel i dnes velmi živé téma.

Učení s učitelem

Při učení s učitelem (*supervised learning*) se síť adaptuje za pomoci *tréninkových dat*. Tréninková data obsahují jak vstupní hodnoty, tak i požadované výstupy při daných vstupních hodnotách. Tréninková data se ještě před samotným učením rozdělí na tři skupiny: *tréninkovou*, *validační* a *testovací* v poměru běžně 70:15:15. Učení s učitelem se aplikuje na sítě, kde jsou neurony uspořádány ve vrstvách, a mezi jednotlivými vrstvami je každý neuron z předcházející vrstvy propojen s každým neuronem z vrstvy následující. Síť můžeme jednoznačně rozdělit na vstupní vrstvu, výstupní vrstvu a tzv. skryté vrstvy, ležící mezi vstupní a výstupní vrstvou. Učení s učitelem probíhá tak, že je skutečná odezva sítě porovnána s požadovanou odezvou dle trénovací množiny. Následně jsou váhy dle užitého učicího pravidla změněny tak, aby výstup sítě odpovídal, nebo konvergoval k požadovanému výstupu, přičemž se jedná o iterační proces. Obecně se síť učí generalizovat [1].

Učení bez učitele

O učení bez učitele (*unsupervised learning*, *self-organized learning*) hovoříme v případě, že máme k dispozici vzorové vstupy, ale nemáme k nim přiřazeny výstupy jako u učení s učitelem. Síť hledá opakující se vzor, za pomoci metrik příbuznosti, na základě kterého lze zadaná data třídit do kategorií.

2 ALGORITMUS BACKPROPAGATION

Backpropagation, neboli algoritmus zpětného šíření chyby, byl navržen Rumelhartem a jeho kolektivem pracovníků. Jedná se gradientní algoritmus, za pomoci kterého jsou adaptovány vícevrstvé dopředné sítě. Backpropagation probíhá jako učení s učitelem, kdy je vypočtený výstup sítě porovnán s požadovaným výstupem. Následně se zpětným šířením signálu upravují váhy tak, aby síť na vzor reagovala požadovaným výstupem. Bylo dokázáno, že takovéto sítě jsou schopny aproximovat s požadovanou přesností jakoukoliv spojitou funkci, a mají proto velké využití například pro regresní analýzu [8]. Obecně je princip backpropagation nejužívanějším algoritmem v oblasti neuronových sítí vůbec. Je použit v asi 80-90% aplikací [3]. Optimalizační postup backpropagation je využíván u dopředných neuronových sítí se skrytými vrstvami a je určen pro klasifikaci dat, která obecně nejsou lineárně separovatelná (v případě lineárně separovatelných funkcí není neuronů ve skrytých vrstvách potřeba) [9].

2.1 Organizační dynamika

Organizační dynamika u zpětného šíření chyby odpovídá vícevrstvému perceptronu. Síť je tvořena pevnou acyklickou topologií se vstupní a výstupní vrstvou a jednou nebo dvěma vnitřními *skrytými vrstvami* neuronů. U skrytých vrstev zavádíme *bias*, tedy nultý vstup roven vždy 1. Mezi jednotlivými vrstvami je úplné propojení, což znamená, že každý neuron je spojen s každým neuronem vrstvy následující. Příklad architektury neuronové sítě se zpětným šířením chyby jsme mohli vidět na Obr. 2b jakožto příklad acyklické sítě.

Speciálním tématem je volba, zda použijeme k řešení problému jednu nebo dvě skryté vrstvy a kolik tyto vrstvy budou obsahovat neuronů. Použijeme-li příliš málo neuronů, síť nemusí být dostatečně přesná pro komplexní problém. Nedostatek neuronů ve skryté vrstvě pro adekvátní klasifikaci nazýváme *nedostatečnou konvergencí* (underfitting). Použijeme-li naopak neuronů příliš mnoho, síť ztratí schopnost generalizace tréninkových vzorů, a nebude tedy schopna dobře klasifikovat vstupy nové, které nebyly zahrnuty v trénovací množině. Tento problém je označován jako *přeučení sítě* (overfitting) [8].

Hned na začátku je potřeba zmínit, že řada výzkumníků se snažila přijít na optimální vzorec pro určování příslušných hodnot, avšak univerzální řešení pro počet neuronů ve skryté vrstvě tak, aby byl výpočetní čas co nejnižší a poskytovaný výsledek co nejpřesnější, nebylo nalezeno [9]. Jedna skrytá vrstva se zpravidla používá při spojitém zobrazení z jednoho konečného prostoru do druhého. Dvě skryté vrstvy pak mohou představovat libovolnou křivku s racionální přenosovou funkcí a mohou tedy aproximovat libovolně hladké křivky s požadovanou přesností. Pokud je naším požadavkem maximální přesnost, pak můžeme v síti použít i více skrytých vrstev, ovšem za cenu velké výpočtové náročnosti. Proto se obecně síť s více než dvěma skrytými vrstvami v praxi moc neuplatňuje.

Zcela orientačními pravidly pro určování počtu neuronů ve skrytých vrstvách jsou dle různých publikací například následující doporučení:

- počet skrytých neuronů jsou 2/3 (nebo 70-90%) počtu neuronů ve vstupní vrstvě,
- počet neuronů ve skrytých vrstvách by měl být menší než dvojnásobek vstupních neuronů,

- počet neuronů ve skryté vrstvě by se měl pohybovat v rozmezí počtu neuronů vstupní a výstupní vrstvy.

Nicméně jak již bylo zmíněno, pravidla jsou pouze orientační a nemusí platit vždy. Ve vhodném počtu neuronů ve skryté vrstvě hrají roli také komplexnost aktivační funkce, učicí pravidlo, ale především povaha a množství tréninkových vzorů, podle kterých jsou upravovány váhy [9]. Proto je třeba pro každý problém řešený metodou backpropagation potřeba zvlášť uvážit všechny výše zmíněné faktory a přizpůsobit jim počet neuronů ve skrytých vrstvách.

2.2 Aktivní dynamika

Aktivní dynamika probíhá v diskrétním čase a odezvu sítě generuje na základě funkce $y(w)$, která je dána maticí w váhových koeficientů a realizuje zobrazení $\mathbb{R}^n \to (0,1)^m$. Pro algoritmus backpropagation je klíčové, aby funkce sítě, tedy i aktivační funkce jednotlivých neuronů, byly diferencovatelné. Na základě inicializačních hodnot přivedených na vstupní neurony počítáme vnitřní potenciály jednotlivých neuronů:

$$\xi_j = \sum_{i \in j_{\leftarrow}} w_{ij} y_i, \tag{10}$$

kde $j_{\leftarrow}$ značí výstupy neuronů z předchozí vrstvy, tedy vstupy j-tého neuronu. Pro vstupní neurony odpovídá $j_{\leftarrow}$ vstupním hodnotám sítě. Na základě vnitřního potenciálu je za pomoci sigmoidální přenosové funkce (8), která spojitě aproximuje ostrou nelinearitu (5), stanoven výstup neuronu:

$$y_j = f_j(\xi_j), \quad \text{kde } f_j(\xi) = \frac{1}{1+e^{-\lambda\xi}}. \tag{11}$$

Parametr $\lambda \epsilon \mathbb{R}$ nazýváme *strmost* (gain), která zaručuje nelineární růst či pokles sigmoidální funkce v bodech blízkých nule (pro $\lambda > 0$ růst, pro $\lambda < 0$ pokles). Běžně je volena hodnota $\lambda = 1$, avšak je možné volit i jiné hodnoty, ve speciálních případech může λ dokonce nabývat různých hodnot pro jednotlivé neurony λ_j.

Díky takto získanému výstupu y_j neuronu můžeme počítat další vrstvu sítě. Tento proces opakujeme přes všechny vnitřní vrstvy až do dosažení výstupní vrstvy, kdy jsou výstupy jednotlivých neuronů odezvou celé sítě.

2.3 Adaptivní dynamika

Adaptivní dynamika nám popisuje proces nastavení vah tak, aby síť generovala správný výsledek. Optimalizační funkce backpropagation představuje učení s učitelem, což znamená, že při procesu učení síti předkládáme nejen vstupní hodnoty, které bude síť klasifikovat, ale i požadované výstupní hodnoty. Trénovací množina o p vzorech T tedy vypadá následovně:

$$T_s = \left\{\{I_1, O_1\}\{I_2, O_2\} \dots \{I_p, O_p\}\right\}.$$

Vidíme, že jde o soubor uspořádaných dvojic, kde $I_i = [i_1, i_2, \dots, i_k]$ je vektor trénovacích vzorů a $O_i = [o_1, o_2, \dots, o_m]$ je výstupní vektor požadovaných hodnot, příslušný i-tému vzoru z trénovací množiny o p prvcích. Jednotlivé vstupy i výstupy nabývají hodnot <0,1> [10].

Jednoduše lze algoritmus zpětného šíření chyby popsat tak, že je vypočtena odezva sítě pro jeden ze vstupů tréninkové množiny za náhodného nastavení vah. Tato odezva je následně porovnána s příslušným požadovaným výstupem. Na základě rozdílu mezi vypočtenou a požadovanou odezvou sítě je vypočtena *parciální chyba sítě* (pro konkrétní vzor). Součet všech parciálních chyb sítě (pro celou tréninkovou množinu) udává celkovou chybu sítě. Vypočtená parciální chyba je vynásobena tzv. *rychlostí učení* (learning rate) a vrácena zpět do sítě, kde šíří zpět od výstupu ke vstupům za úpravy jednotlivých váhových koeficientů tak, aby při dalším výpočtu odezvy byla chyba sítě menší. Po vypočítání všech parciálních chyb sítě od všech prvků tréninkové množiny jsou tyto chyby sečteny, přičemž dostáváme celkovou chybu sítě. Na závěr celkovou chybu sítě porovnáme s požadovanou maximální chybou sítě. Pokud je celková chyba větší než požadovaná maximální chyba, pak se celý postup opakuje.

Princip algoritmu zpětného šíření chyby spočívá v minimalizaci chybové funkce, která je definovaná jako [3]:

$$E(w) = \sum_{k=1}^{p} E_k(w), \tag{12}$$

kde p je počet tréninkových vzorů a $E_k(w)$ představuje parciální chybu sítě pro konkrétní tréninkový vzor [10]:

$$E_k(w) = \frac{1}{2}\sum_{j=1}^{m}(y_j - o_j)^2. \tag{13}$$

Kde je m počet neuronů ve výstupní vrstvě, y_j je vypočtenou odezvou j-tého neuronu a o_j je požadovaná odezva j-tého neuronu, která je určená tréninkovou množinou.

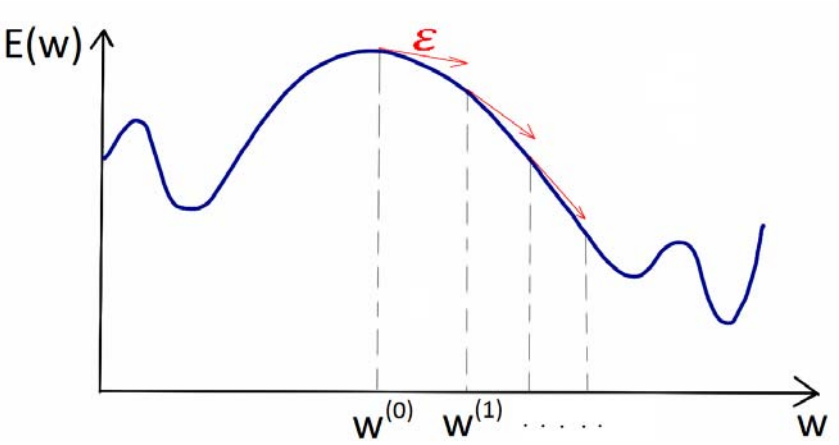
Minimalizaci chybové funkce provádíme pomocí *gradientní metody*. Abychom gradientní metodu mohli k řešení tohoto problému použít, musí být chybová funkce diferencovatelná. Pokud je podmínka diferencovatelnosti splněna, můžeme upravovat váhy dle následujícího učicího pravidla [3]:

$$w_{ij}^{(t)} = w_{ij}^{(t-1)} + \Delta w_{ij}^{(t)}, \tag{14}$$

kde $\Delta w_{ij}^{(t)}$ představuje změnu váhových koeficientů v čase t a je úměrná zápornému gradientu chybové funkce v bodě $w^{(t-1)}$ [3] :

$$\Delta w_{ij}^{t} = -\varepsilon \frac{\partial E}{\partial w_{ij}}(w^{(t-1)}), \tag{15}$$

kde $\varepsilon \epsilon (0,1)$ je *koeficient učení* a $\frac{\partial E}{\partial w_{ij}}$ představuje gradient chybové funkce. Pro lepší představu hledání minima chybové funkce nám poslouží obrázek Obr.10. Na obrázku vidíme vývoj chybové funkce v závislosti na váhové konfiguraci. Jak již bylo zmíněno, hledáme minimum funkce tak, že k příslušnému nastavení vah (v čase t=0 je konfigurace náhodná), tedy bodu na křivce funkce, vypočteme gradient (tečný vektor), jemuž pomocí koeficientu učení přiřadíme velikost. O takto orientovaný vektor se posuneme v rámci váhového prostoru tak, abychom postupně zmenšovali celkovou chybu. Postup opakujeme, dokud se na chybové funkci limitně neblížíme lokálnímu minimu. Je nutné poznamenat, že gradientní metoda za vhodné volby koeficientu učení vždy konverguje k lokálnímu minimu chybové funkce, nicméně nemusí k tomu pro velkou výpočetní náročnost dojít v reálném čase [3].



Obr. 10) Gradientní metoda

Výpočet gradientu chybové funkce počítáme tak, že jej vyjádříme jako součin parciálních chybových funkcí. Uvědomíme-li si, že jednotlivé chybové funkce závisí na funkcích jednotlivých neuronů sítě, pak můžeme využít derivace složené funkce takto [3]:

$$\frac{\partial E}{\partial w_{ij}} = \sum_{k=1}^{p} \frac{\partial E_k}{\partial w_{ij}} = \sum_{k=1}^{p} \frac{\partial E_k}{\partial y_j} \frac{\partial y_j}{\partial \xi_j} \frac{\partial \xi_j}{\partial w_{ij}} \tag{16}$$

$$\frac{\partial \xi_j}{\partial w_{ij}} = y_i \tag{17}$$

$$\frac{\partial y_j}{\partial \xi_j} = \lambda_j y_j (1 - y_j) \tag{18}$$

$$\frac{\partial E_k}{\partial y_j} = y_j - o_j. \tag{19}$$

Pro výstupní neurony lze tedy parciální derivaci počítat a zpětným šířením jsme schopni aktualizovat váhové koeficienty i napříč sítí. Jak již bylo zmíněno, po každé úpravě vah je přepočítána celková chyba sítě a porovnána s požadovanou maximální chybou sítě. V případě, že je vypočtená hodnota menší nebo rovna požadované maximální chybě, proces učení končí. V opačném případě se celý postup opakuje až k dosažení žádané přesnosti.

3 HOPFIELDOVA SÍŤ

Autorem tohoto modelu je John Hopfield, který svou prací navázal na model McCullocha a Pittse. Model byl analyzován dalšími vědci, nicméně až právě Hopfield model uvedl do obecného povědomí svou analogií s fyzikální teorií magnetických materiálů za pomoci *energetické funkce*. Výhodami Hopfieldovy sítě je schopnost rozpoznávat vzory inverzní k vzorům naučeným, jednorázové učení pomocí Hebbova zákona a případná možnost přiučení dalších vzorů atd. [11]. Je nutné podotknout, že existuje více variant Hopfieldova modelu, například spojitá Hopfieldova síť, kdy se využívá spojité přechodové funkce. V následujícím textu se však budeme zabývat pouze základnímu modelu sítě, kdy síť pracuje v diskrétním čase.

Síť funguje jako *asociativní paměť*. Zatímco klasická počítačová paměť hledá informace podle příslušné adresy, asociativní paměť dokáže vybavit požadovanou informaci na základě její neúplné znalosti (tedy asociace). Asociativní paměť můžeme dále dělit na *autoasociativní* a *heteroasociativní*. Autoasociativní paměť, na kterou se zaměříme, dokáže doplnit vstupní informaci. Například když se podíváme oprýskanou ceduli, na které se již místy odloupla barva, stále si můžeme domyslet, jak vypadala, když byla nová, případně co na ní bylo napsáno. Heteroasociativní paměť vybavuje informaci, která se vstupní informací nějakým způsobem souvisí. Příkladem by tedy mohla být cedule s obrázkem města, kdy heteroasociativní paměť vybaví název daného města.

3.1 Organizační dynamika

Hopfieldova síť je pevná cyklická síť s N neurony. Každý neuron v síti je vstupem i výstupem sítě, přičemž všechny neurony jsou vzájemně ohodnoceny vahami a propojeny (každý s každým), avšak žádný neuron není spojen sám se sebou. Příklad organizační dynamiky Hopfieldovy sítě můžeme vidět na Obr. 2a.

Váhy sítě jsou vždy celočíselné a symetrické, tedy $w_{ij} = w_{ji}$. Proto můžeme dvojici orientovaných vah mezi jednotlivými neurony považovat za jedno neorientované spojení. Díky této vlastnosti je též Hopfieldova síť nazývána jako *síť symetrická*. Jednotlivé neurony sítě mají neměnný práh a skokovou přechodovou funkci. Neurony mohou nabývat stavu {0,1}, jak je tomu u klasické skokové funkce podle (4), nebo bipolárních hodnot{-1,1} dle (5). Dále bude síť popisována za pomoci bipolárních hodnot, jelikož vzorce jednotlivých funkcí jsou pro tyto hodnoty přehlednější.

3.2 Adaptivní dynamika

Adaptivní dynamika se u Hopfieldovy sítě řídí *Hebbovým učicím pravidlem*. To bylo v roce 1949 definováno Donaldem Hebbem jako analogie k neurofyziologickému osvojování podmíněných reflexů [4]. Pravidlo je určeno pro neurony s binárními vstupy i výstupy a je založeno na myšlence, že spojení mezi dvěma neurony je zesíleno v případě, že jsou tyto neurony souhlasně aktivovány. V případě, že stavy neuronů jsou opačné, spojení mezi nimi je utlumeno.

Hebbovo učení je příkladem učení bez učitele, což znamená, že síti dodáváme vstupní hodnoty, nikoliv však požadované výstupy. Síti tedy zadáváme p vzorů, přičemž každý vzor

odpovídá vektoru *n* hodnot {-1,1} vstupních a výstupních neuronů, které si u autoasociativní paměti odpovídají. Učení probíhá tak, že se na jednotlivé neurony přivedou hodnoty, které chceme síť naučit. Následně jsou aktualizovány váhové koeficienty podle vzorce [8]:

$$w_{ij}(t) = w_{ij}(t-1) + f(\xi)_i f(\xi)_j, \tag{20}$$

kde *t* odpovídá hodnotě diskrétního kroku. Vzorec udává, že v případě souhlasných stavů neuronů {1,1} nebo {-1,-1} bude váha posílena o 1 a naopak při nesouhlasných stavech neuronů {-1,1} nebo {1,-1} bude váha o 1 oslabena.Výslednou konfiguraci pro *p* vzorů můžeme tedy vyjádřit následovně [8]:

$$w_{ij} = \sum_{k=1}^{p} f(\xi)_{ki} f(\xi)_{kj} \quad 1 \leq j \neq i \leq n \tag{21}$$

Odtud plyne i symetričnost vah $w_{ij} = w_{ji}$, kterou jsme zmínili na začátku kapitoly. Váhy jsou při každé aktualizaci buď zvýšeny, nebo sníženy o hodnotu 1, podle toho, kolikrát byly příslušné neurony souhlasné, resp. nesouhlasné. Na základě toho si můžeme uvědomit, že výsledné znaménko vah vypovídá o tom, zda celkově převyšovaly opačně nebo souhlasně orientované spoje [3]. Z uvedeného vyplývá, že vzory, které odpovídají i požadovaným hodnotám, nejsou v síti reprezentovány přímo, jak je tomu například u učení s učitelem, nýbrž jsou zakomponovány do vah na spojení jednotlivých neuronů.

3.3 Aktivní dynamika

Nejdříve se v aktivním režimu nastavují stavy jednotlivých neuronů, na vstupy sítě. V tuto chvíli se tedy vstupy rovnají výstupům příslušných neuronů. Následně jsou sekvenčně synchronně aktualizovány jednotlivé neurony. Dalším krokem je vyčíslení vnitřního potenciálu neuronu, který vždy spadá do oboru celých čísel, a jeho výsledné znaménko indikuje jeho novou hodnotu. Potenciál neuronu *i* vyčíslíme následovně [3]:

$$\xi_i^{(t)} = \sum_{j=1}^{n} w_{ij} \xi_j^{(t-1)}, \tag{22}$$

kde $\xi_i^{(t)}$ – je potenciál neuronu v čase *t* a *n* je počet neuronů sítě. Síť takto aktualizujeme tak dlouho, dokud nedosáhne *stabilního stavu*, tedy:

$$\xi_i^{(t+1)} = \xi_i^{(t)} \tag{23}$$

pro všechny neurony v síti. Dobu před ustálením sítě do stabilního stavu, tedy kdy se síť opakovaně aktualizuje, nazýváme *relaxace*. Výstupní neurony ve stabilním stavu určují výstup sítě.

Dá se dokázat, že podle této aktivní dynamiky se symetrickými vahami je sekvenční výpočet Hopfieldovy sítě pro jakoukoliv kombinaci vstupních hodnot vždy konečný a generuje zobrazení f(w): $f(w): \{-1,1\}^n \rightarrow \{-1,1\}^n$ [3]. Funkce f(w) je kromě příslušných hodnot vah závislá na pořadí aktualizace neuronů. Výpočet lze provádět také paralelně, nicméně v takovém případě se síť nemusí ustálit a výsledek může po určité době střídat dva stavy [3].

3.4 Energetická funkce

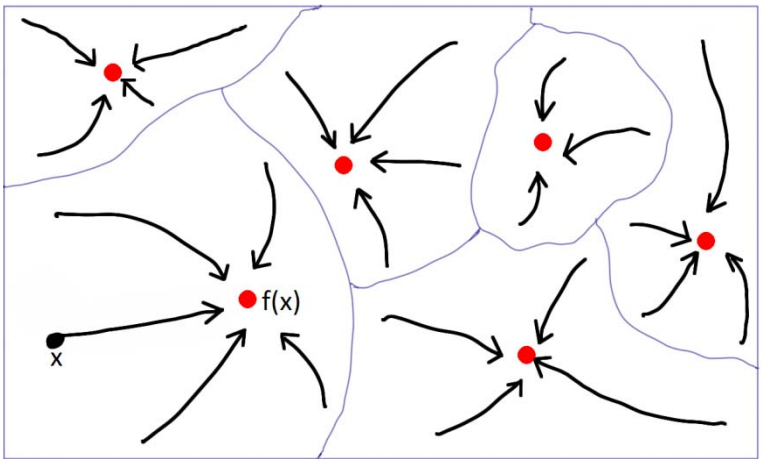
Jak již bylo zmíněno, Hopfield se ve své práci inspiroval fyzikální teorií magnetismu. Magnetické materiály jsou v rámci teorie považovány za soubor elementárních magnetů na úrovni atomů, tzv. *spinů,* uspořádaných do krystalové mřížky [3]. Za předpokladu, že jednotlivé spiny mohou nabývat kladné a záporné hodnoty, dostáváme analogii s Hopfieldovou sítí, kdy jednotlivé neurony mohou nabývat hodnot {-1,1}. Spiny v magnetických materiálech jsou ovlivňovány jednak externími magnetickými vlivy a jednak interním polem vlastního magnetického materiálu. Každý atom přináší do vnitřního magnetického pole příspěvek úměrný spinu. V Hopfieldově síti můžeme k externím vlivům přirovnat vstupní hodnoty a neurony ke spinům, které za pomoci vah přinášejí svůj příspěvek do vnitřního pole Hopfieldovy sítě, které zpětně ovlivňuje neurony.

Vzhledem k výše uvedenému zavedl Hopfield tzv. e*nergetickou funkci*, která přiřazuje *potenciální energii* každému stavu neuronu a celé síti takto [10]:

$$E_e(y^t) = -\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ji} y_i^{(t-1)} y_j^{(t-1)} \qquad j \neq i. \tag{24}$$

Čím má síť nižší hodnotu energetické funkce, tím je stabilnější. Nejstabilnější je ve chvíli, kdy jsou hodnoty $w_{ji}x_ix_j$ maximální, tedy znaménko součinu výstupů odpovídá znaménku vah.

Při procesu učení Hopfieldovy sítě je každým vzorem generováno lokální minimum energetické funkce. Okolí těchto lokálních minim nazýváme *oblastí atrakce,* kde nacházíme výstupy podobné příslušnému vzoru. Při procesu minimalizace energetické funkce jsou tyto body z oblasti atrakce posouvány do lokálního minima funkce, které je pro energetickou funkci stabilním bodem. Geometricky si tedy můžeme síť pomyslně rozdělit na podoblasti jednotlivých vzorů, ke kterým je každý z výstupů sítě přitahován, viz Obr.11.



Obr. 11) Plocha energetické funkce

Na ploše energetické funkce však mohou vznikat tzv. *nepravé vzory* (*fantomy*). Fantomy jsou spontánně vzniklá lokální minima energetické funkce, která však nemají odpovídající vzor. Pakliže zadáme síti vstup spadající do oblasti atrakce nepravého vzoru, síť vybaví nesmyslný výstup. Existují však způsoby, jak v rámci adaptivní dynamiky fantomy

dodatečně vyloučit. Například u sítě s vahami w_{ij} můžeme nepravý vzor se vstupními hodnotami x'_i vyloučit úpravou vah tak, že [3]:

$$w'_{ij} = w_{ij} - x'_i x'_j. \tag{25}$$

V neurofyziologické analogii je eliminace fantomů vlastně léčení neuróz. Existují i teorie, na základě kterých jsou v lidském mozku fantomy odučovány během spánku a tím dochází ke zlepšení paměti [3].

3.5 Kapacita sítě

Počet vzorů, které si je Hopfieldova síť schopna zapamatovat, není neomezený. Schopnost reprodukce (zapamatování) je předpokladem asociativní paměti, což znamená, že množina vzorů představuje lokální minima na ploše energetické funkce. Kapacita asociativní paměti pro Hopfieldovu síť je závislá na poměru počtu vzorů *p*, které chceme síť naučit, a počtu neuronů sítě *n*. Přesněji, pro reprodukci všech vzorů, které chceme ukládat do paměti Hopfieldovy sítě, musí být počet těchto vzorů maximálně [3]

$$p_{max} = \frac{n}{\log n}. \tag{26}$$

Avšak naším cílem není pouze uložení vzorů do paměti, nýbrž i jejich zpětného vyvolání, a to i na základě jejich neúplné znalosti. Z analýzy vyplývá, že Hopfieldovu síť lze použít jako autoasociativní paměť v případě, že

$$p \leq 0.138n, \tag{27}$$

kdy vzory odpovídají lokálním minimům energetické funkce [3]. V případě, že kapacita sítě (*p/n*) převyšuje hodnotu 0.138, lokální minima energetické funkce příslušející daným vzorům zanikají. Naopak pokud

$$p \leq 0.05n, \tag{28}$$

z lokálních minim odpovídajících vzorům se stávají globální minima energetické funkce [3]. Globální minima jsou hlubší než lokální minima fantomů. Z výše uvedeného plyne, že Hopfieldova síť s dobrými vlastnostmi je velmi paměťově náročná. Například pro naučení 10 různých vzorů je třeba 200 neuronů, tedy pro cyklickou síť 40000 spojů s vahami z oboru celých čísel. Z tohoto důvodu základní model Hopfieldovy sítě nenachází praktické uplatnění a je používán spíše k teoretickému pochopení problematiky, nicméně existuje řada modifikací základního modelu, které se pokoušejí o eliminaci problému paměťové náročnosti sítě.

4 KOHONENOVY SAMOORGANIZAČNÍ MAPY

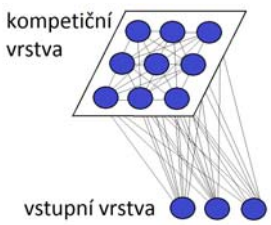
Samoorganizační algoritmy (self organizing map, SOM, self organizing feature map, SOFM) se *učí bez učitele* pomocí analýzy dat, přičemž využívají tzv. *soutěžní strategii* (*competitive learning*). Při této strategii je ve výsledku aktivován vždy právě jeden (vítězný) neuron ve výstupní (Kohonenově) vrstvě. Specifickým znakem SOM je realizace zobrazení $\mathbb{R}^n \rightarrow \{0,1\}^m$, které zachovává topologii a zobrazuje tak charakteristické rysy dat z trénovací množiny. U zobrazení je velmi významný fakt, že hodnoty vstupů, které jsou vzájemně blízko, jsou zobrazeny na výstupní prostor taktéž blízko sebe. Pomocí těchto typů sítí jsme schopni zjednodušit mnohorozměrné datové struktury vstupů na jednoduchý dvourozměrný výstup. Samoorganizační sítě jsou opět inspirovány činností mozku, přičemž fenomén topologického zobrazení příznaků je významně zastoupen v mozcích jak vyšších savců, tak i člověka [8].

Soutěžnímu učení se intenzivně věnovalo více vědců zaráz. Například v 70. letech 20. století to byli von der Malsburg, Grossberg, Willsaw či Fukushima [3]. Nejvýznamnějším přínosem této doby však byla Kohonenova samoorganizační mapa popsaná roku 1982. Kohonen, ačkoliv síť nese jeho jméno, nebyl prvním objevitelem algoritmu, nicméně do něj zanesl koncept množiny reprezentantů, majících stejné pravděpodobnosti výběru, který významně zjednodušil předchozí výpočetní postupy [8].

4.1 Organizační dynamika

Samoorganizační mapy jsou zpravidla tvořeny dvěma vrstvami neuronů. Jedná se o vstupní vrstvu s n neurony a výstupní vrstvu s m neurony, která navíc bývá uspořádána do topologické struktury, zpravidla mřížky pro dvojrozměrný prostor či řady pro jednorozměrný prostor. Schéma organizační dynamiky vidíme na Obr.12. Jak již bylo zmíněno, hodnoty, které jsou si blízké na vstupu, se na výstupní mřížku promítají taktéž s malou vzdáleností, proto můžeme definovat pojem *okolí neuronu*, což je vzdálenost s, do které spadají neurony blízké příslušnému neuronu. Okolí má zásadní význam v adaptivní dynamice sítě, kterou si přiblížíme později.

Vrstvy Kohonenovy sítě jsou zcela propojeny, tedy každý neuron vstupní vrstvy je spojen s každým neuronem výstupní vrstvy, přičemž každý spoj je ohodnocen vahami. Specifikem kompetitivních sítí je vzájemné propojení neuronů ve výstupní (*kompetiční*) vrstvě způsobující tzv. *laterální inhibici*. Jednotlivé neurony mají zpravidla sebeexcitující vazbu (+) a inhibiční vazbu vůči okolním neuronům (-). Ve výsledku je vítězný neuron, tedy ten, který je nejvíce excitovaný, ještě více excitován, zatímco okolní neurony jsou inhibovány na minimum [10].



Obr. 12) Kohonenova samoorganizační mapa

4.2 Aktivní dynamika

Výstup klasifikace Kohonenovy samoorganizační mapy je takový, že neuron s největší excitační hodnotou (tedy s nejmenší vzdáleností od vzoru) má hodnotu 1, přičemž všechny ostatní výstupní neurony mají hodnotu 0. Výpočet odezvy probíhá následovně [8]:

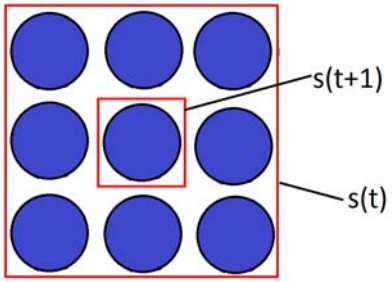
$$y_j^{(t)} = \begin{cases} 1 & \text{pro } j = \arg min_{l=1,\dots,h}\{\|x^{(t)} - w_{ij}\|\} \\ 0 & \text{pro ostatní hodnoty} \end{cases}, \tag{29}$$

kde $\|x^t - w_l\|$ představuje euklidovskou normu a w_{ij} jsou váhy příslušející danému spoji. Váhy zde reprezentují polohu výstupního neuronu ve vstupním prostoru. Hledáme tedy minimální vzdálenost mezi vstupním vektorem hodnot a výstupními neurony. Neuron, který je vstupu nejblíže, má nejvyšší potenciál, mající význam v kompetiční vrstvě, a to tak, že velikostí přímo úměrnou svému potenciálu zeslabuje neurony ve svém okolí [3]. Nakonec tedy nejvíce excitovaný neuron utlumí zbytek výstupů, přičemž je sám stále aktivní.

4.3 Adaptivní dynamika

Učení u Kohonenových samoorganizačních map probíhá předkládáním vzorů síti, přičemž je vyhodnocen vítězný neuron výstupní sítě tak, jak je popsáno v aktivní dynamice. Počáteční nastavení vah odpovídá náhodným malým hodnotám, většinou z intervalu (0,1). Následuje úprava vah aktivního neuronu tak, aby se výstupní neuron přiblížil zadanému vzoru tréninkové množiny. Při projití celé tréninkové množiny tedy dosáhneme přesunu výstupního neuronu do oblasti s největší koncentrací jemu podobných vstupů- shlukových oblastí [10].

Díky topologii výstupní vrstvy, kdy malá vzdálenost hodnot ve vstupním prostoru znamená blízkost výstupních neuronů, může adaptivní dynamika Kohonenových samoorganizačních map pracovat s pojmem okolí neuronu. V čase hodnota okolí neuronu klesá, viz Obr. 13, což nám umožňuje posouvat v rámci výstupní sítě nejen vítězný neuron kompetice, ale i neurony, které patří do jeho okolí.



Obr. 13) Okolí neuronu v závislosti na čase

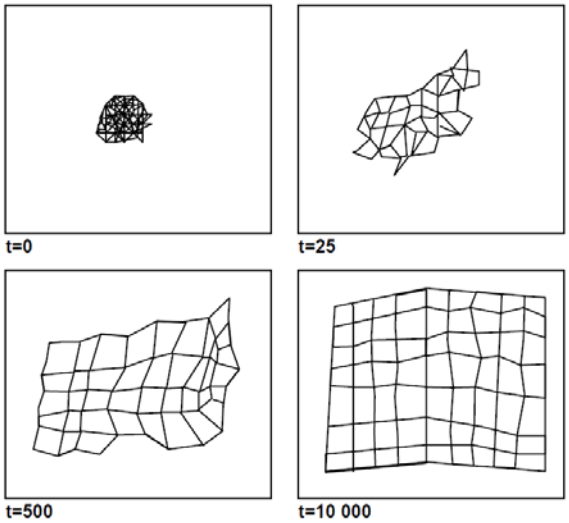
Učicí algoritmus samoorganizační mapy definujeme následovně [2]:

$$w_{ij}^{(t)} = w_{ij}^{(t-1)} + \theta\left(x_i^{(t)} - w_{ij}^{(t-1)}\right) \; pro \; j \epsilon \; s(c), \tag{30}$$

kde $c = \arg min_{ij=1,\dots,h}\{\|\mathbf{x}^{(t)} - \boldsymbol{w}\|\}$, $\theta \in \mathbb{R}$ je koeficient učení, $s \epsilon \mathbb{N}$ je okolí vítězného neuronu a představuje množinu neuronů náležející aktuálnímu okolí, která je závislá na čase. Koeficient učení nabývá hodnot (0,1) přičemž v případě SOM je na počátku blízký 1

a rychlost učení v závislosti na (diskrétním) čase klesá k 0. Taktéž okolí zasažené vítězným neuronem se při procesu učení zmenšuje. Obecně rozměr okolí na začátku adaptace odpovídá celé síti a zmenšuje se až na hodnotu 0, kdy jsou adaptovány pouze váhy vítězného neuronu. Grafické znázornění procesu učení Kohonenovy mapy je na Obr.14.

Shlukování si můžeme představit také jako funkci hustoty pravděpodobnosti, která definuje rozložení dat v rámci prostoru. Za pomoci této funkce můžeme počítat pravděpodobnost výskytu vektoru pro každý bod uvažovaného prostoru. Pokud jsou tedy vstupní hodnoty rozloženy na základě distribuční funkce, očekáváme totéž rozdělení i na výstupním prostoru[10].



Obr. 14) Vizualizace procesu adaptace Kohonenovy mapy [10]

4.4 Kohonenova mapa a učení s učitelem

Jak jsme si ukázali výše, samoorganizační mapy jsou schopny na základě hustoty pravděpodobnosti upravit výstupní síť samy, můžeme je však použít také ke klasifikaci dat do předem stanovených tříd, což odpovídá typu učení s učitelem nebo též *kvantizaci dat učením* (learning vector quantization, LVQ). Pro každý shluk bude existovat požadovaný výstup, tedy neuron kompetiční vrstvy, z čehož vyplývá, že počet neuronů ve výstupní vrstvě bude odpovídat počtu tříd, do kterých chceme data rozdělit.

První krok adaptace pro učení s učitelem probíhá bez požadovaných výstupů, tedy stejně jako adaptace bez učitele. Síti jsou tedy předloženy pouze trénovací vzory a probíhá vektorová kvantizace. Dalším krokem je pak *doučení sítě*, které adaptuje váhy na naše požadované výstupy, nicméně na rozdíl od samoorganizace nepracujeme s celým okolím, nýbrž pouze s jedním, resp. dvěma neurony dle použitého algoritmu. Kohonen v průběhu života popsal tři způsoby kvantizace dat učením.

První algoritmus (LVQ1) funguje na takovém principu, že při správné klasifikaci hodnot dle požadovaného výstupu je příslušný výstupní neuron posunut směrem ke vstupu stejně, jak je tomu při učení bez učitele. Naopak v případě, že síť vstup klasifikuje jinak, než požadovaným výstupem, je výstupní neuron posunut do větší vzdálenosti od vstupu. Úprava vah, a tedy i posouvání neuronů v rámci výstupní mřížky, probíhá následovně [12]:

$$w_{ij}^{(t)} = \begin{cases} w_{ij}^{(t-1)} + \alpha\left(x^{(t)} - w_{ij}^{(t-1)}\right) \text{ pro } y^{(t)} = o_i \\ w_{ij}^{(t-1)} - \alpha\left(x^{(t)} - w_{ij}^{(t-1)}\right) \text{ pro } y^{(t)} \neq o_i \end{cases}, \tag{31}$$

kde $\alpha \epsilon \mathbb{R}$ je koeficient posunutí, který se po několika desítkách až stech tisících cyklů přiblíží nule, nicméně na rozdíl od koeficientu učení u samoorganizace se α inicializuje obecně hodnotami 0,01 až 0,02, což prakticky znamená, že i posunutí příslušného neuronu od/k vstupní hodnoty je velmi malé [3]. Posuvy se vytvoří shluky, přičemž okolí hranice klasifikovaných tříd se vyprázdní. Bylo zjištěno, že takto vzniklá hranice leží na středu spojnic výstupních neuronů a aproximuje k tzv. *bayesovské rozhodovací hranici*.

Druhý algoritmus LVQ2 navazoval na poznatky LVQ1 již s cílem záměrně posouvat svou rozhodovací hranici k bayesovské hranici. Toho mělo být docíleno posuvem dvou výstupních neuronů. Výběr těchto neuronů probíhá tak, že pro konkrétní vstupní vektor jsou vybrány dva jeho nejbližší výstupní neurony, pro které platí, že jeden klasifikuje správně a druhý špatně. Tyto neurony však nesmí být vstupu příliš blízko, proto se vstup zpravidla nachází v pomyslném pásmu na středu vzdálenosti mezi vybranými neurony[12]. Šířka pásma je volena zpravidla mezi 10-30% vzdálenosti neuronů [3]. Menší hodnotu volíme pro co nejpřesnější hranici, zatímco vyšší hodnota nám umožňuje zahrnout statisticky významné množství dat. Výpočet hodnot posunutí je totožný s LVQ1, to znamená, že správně hodnotící neuron posouváme směrem ke vstupu, zatímco nesprávně hodnotící opačným směrem. Praktickými experimenty bylo prokázáno, že LVQ2 je použitelný pouze pro nízký počet učících cyklů (kolem 10000), pro více cyklů se totiž jeho klasifikační schopnost opět začíná zhoršovat [3].

Třetí algoritmus učení s učitelem LVQ3 opět rozšiřuje svého předchůdce. Tentokráte je docíleno toho, aby byl algoritmus stabilní, tedy aby byl aplikovatelný i na velké množství cyklů bez ztráty na přesnosti. Toho bylo docíleno přidáním dalšího pravidla, kdy kromě posuvu výstupních neuronů dle LVQ2 jsou dodatečně upraveny váhy správně hodnotícího výstupního neuronu tak, aby byl ještě více posunut ke vstupu[3]:

$$w_{ij}^{(t)} = w_{ij}^{(t-1)} + \omega\alpha\left(x^{(t)} - w_{ij}^{(t-1)}\right), \tag{32}$$

kde parametr ω je v čase konstantní a měl by se nacházet v rozmezí <0,1;0,5>, přičemž nižší hodnota se volí pro užší pásma a vyšší pro širší pásma.

5 PRAKTICKÁ ČÁST

V praktické části se budeme věnovat jednoduchým aplikacím, které klasifikují data na základě algoritmů popsaných v předchozích kapitolách. Řešení úloh je realizováno v prostředí Matlab.

Matlab (z anglického matrix laboratory) je interaktivní prostředí společnosti MathWorks s vlastním programovacím jazykem pro rychlé a jednoduché vědecké výpočty, vývoj algoritmů, analýzu a vizualizaci dat. Nachází uplatnění v širokém spektru aplikací jako je komunikace, testování a měření, zpracování signálu a obrazu, finančního modelování a analýzy a mnoha dalších. Velkou výhodou Matlabu je možnost importu, exportu a zpracování dat různých formátů. Součástí tohoto nástroje je i velmi podrobná dokumentace popisující všechny funkce, jejich argumenty a ve většině případů i výstižně podané vysvětlení problematiky příslušného tématu, doplněné o schémata a konkrétní příklady.

Prostředí Matlab nabízí mimo výpočtové funkce také obrovské spektrum knihoven, ozn. jako *toolbox*, zaměřených na specifické oblasti. Jednou z těchto knihoven je Neural *Network toolbox* určený speciálně pro práci s neuronovými sítěmi, konkrétně pro navrhování, trénování, simulaci a vizualizaci. Nabízejí se zde možnosti práce s daty: klasifikace, shlukování, regrese, předpovídání časových řad, redukce dimenzí či modelování a řízení dynamických systémů. Práce se sítěmi, především návrh a trénování, je možná jak v rámci interaktivního příkazového řádku, tak i speciálního grafického rozhraní.

5.1 Iris data set

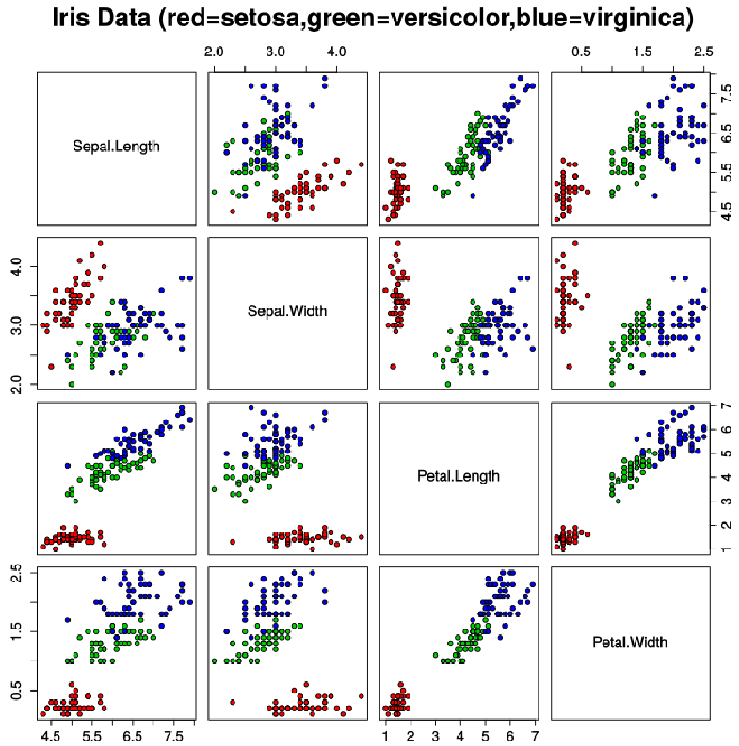
Pro aplikace metody backpropagation a Kohonenovy samoorganizační mapy bude využita databáze kosatců *Iris data set.* Informace do této databáze byly v roce 1935 shromážděny Edgarem Andersonem a následně v roce 1936 publikovány Ronaldem Fisherem v článku „The Use Of Multiple Measurements In Taxonomic Problems" [13]. Databáze je v oblasti neuronových sítí velmi známá a asi nejvíce využívaná. Můžeme ji najít např. na UCI Machine Learning Repository [14] nebo v přiloženém souboru jako puvodni_dataset.txt.

Data obsažená v databázi se váží ke třem třídám kosatců: Iris Setosa, Iris Versicolour a Iris Virginica. Tato data jsou znázorněna na Obr.15 a Obr.16. Druhý z obrázků je generován pomocí K-means funkce v Matlabu, která nalezne centrum dat příslušné třídy velmi podobným způsobem, jako Kohonenova samoorganizační mapa. Jelikož jen těžko zobrazíme čtyřdimenzionální prostor, jsou vygenerovány dva grafy, přičemž se mění jedna z os grafu určená pro délku/šířku okvětního lístku.

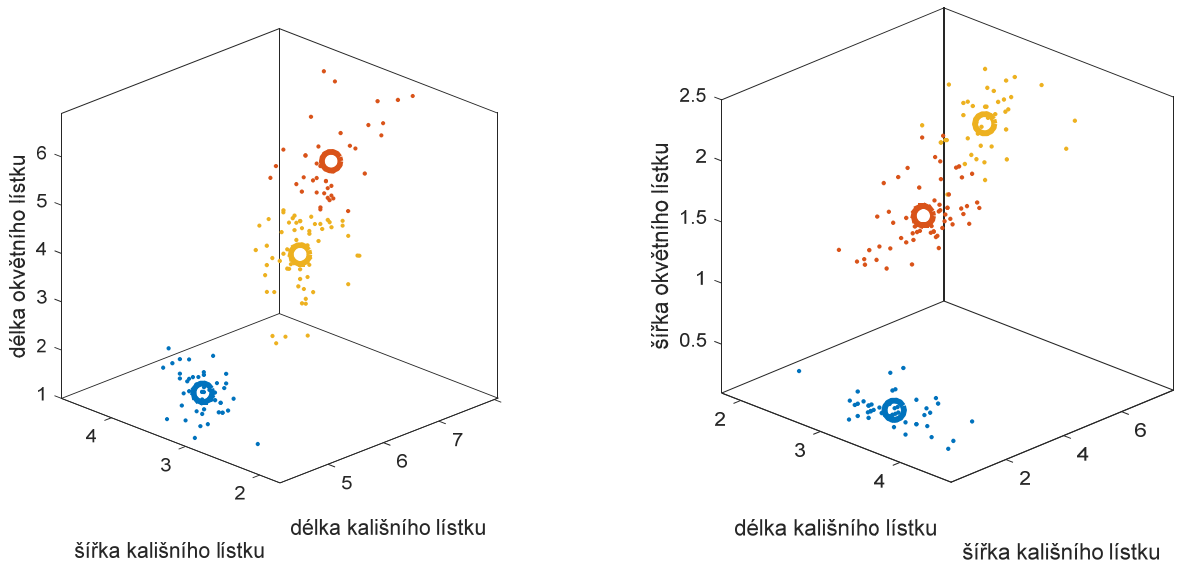
Od každé třídy je v datasetu 50 vzorků, u nichž pozorujeme čtyři atributy:

- délka kališního lístku v cm (sepal length),
- šířka kališního lístku v cm (sepal width),
- délka okvětního lístku v cm (petal length),
- šířka okvětního lístku v cm. (petal width).

Cílem neuronové sítě je klasifikovat správně druhy kosatců (*species*: *Setosa, Versicolor a Virginica*) na základě vložených vstupů (čtyři atributy, viz výše).



Obr. 15) Rozložení dat v rámci Iris datasetu [16]



Obr. 16) Rozložení datasetu a centroidů (výpočet pomocí Matlab funkce `kmeans`)

Databáze se hojně využívá i ve statistických metodách, sám Fišer ji použil v rámci diskriminační analýzy, která je jednou z metod mnohorozměrné statistické analýzy [15]. V rámci statistiky můžeme na databázi sledovat různé vlastnosti, jako vztah mezi délkou a šířkou okvětního lístku, které dvojice rozměrů nejvíce korelují nebo zda mají příslušné třídy další podtřídy, které lze pomocí analýzy dat najít.

5.2 Dopředná neuronová síť a úloha identifikace kosatců

Prvním úkolem je natrénování dopředné neuronové sítě pomocí metody backpropagation. Při práci se sítí vycházíme z výše zmíněné Fišerovy databáze. Prvním krokem je vhodná příprava dat. Dále jsou popsány části kódu na konkrétním příkladu, a výstupy, na základě kterých můžeme hodnotit výkonnost sítě. Na závěr jsou uvedeny příklady konkrétních nastavení sítě a jejich hodnoty.

5.2.1 Příprava dat

Původní Fišerova databáze je koncipována jako matice 5x150, kde první čtyři sloupce jsou hodnoty atributů v centimetrech a pátý sloupec je slovně vyjádřený typ kosatce, tedy požadovaný výstup.

Jak bylo řečeno v kapitole o algoritmu backpropagation, výstup sítě nabývá hodnot $\{0,1\}^n$. Z tohoto důvodu je třeba, aby jednotlivé druhy kosatců (jakožto požadované výstupy) byly určeny taktéž za pomoci těchto hodnot. Za předpokladu, že máme tři výstupy, můžeme slovní hodnocení stanovit následovně:

- Iris setosa → 100
- Iris versicolor → 010
- Iris virginica →001.

Jelikož funkce pro učení neuronových sítí vyžaduje vstupy a výstupy zvlášť, tak i dataset byl rozdělen na dva textové soubory. Posledním krokem byla úprava textových souborů tak, aby v matlabovské syntaxi představovaly matici. To znamená, že jednotlivé hodnoty jsou odděleny mezerou a každý řádek matice čárkou. Takto upravené textové soubory lze nalézt v elektronické příloze jako backpropagation/`vstupy.txt a vystupy.txt`.

5.2.2 Obecný popis kódu a jeho výstupů

V této části se budeme zabývat kódem pro trénování neuronové sítě pomocí metody backpropagation. Výstupem tréninkového algoritmu je nejen síť schopná klasifikovat, ale i řada grafů vypovídajících o průběhu trénování, chybovosti, regresi hodnot atd. Tyto grafy jsou vykreslovány již v průběhu učení sítě, kdy vizualizují její průběh. Prezentovaný program je k dispozici v backpropagation/vzorovy_skript.m v rámci elektronické přílohy.

Trénování dopředných neuronových sítí v prostředí programu Matlab je možné dvěma způsoby. Jednak je možné spustit interaktivní prostředí, které nás nastavením sítě krok po kroku provede, nebo je možné napsat kód v rámci příkazového řádku či skriptu. Vlastnoruční psaní kódu je výhodnější, jelikož poskytuje mnohem více možností, než interaktivní program. My si nyní popíšeme postup s psaným kódem a popíšeme si možnosti, které nabízí.

Načtení tréninkových dat:

```
x = vstupy';
t = vystupy';
```

Proměnné vstupy a výstupy odpovídají výše zmíněným předpřipraveným datům v podobě matic o rozměrech 150x4 a 150x3. Transpozice je nutná jen v případě, že vektory vstupů jsou zadány řádkově, jelikož ji následně budeme dle vzorce (2) násobit váhovým vektorem 150x1.

Volba optimalizační metody pro trénink sítě, tj. volba trénovací funkce

Zde se nám nabízí tři hlavní možnosti. První dvě se řadí mezi exaktní metody, kdy se využívá analýza validačních dat. To znamená, že během procesu učení sítě je sledována validační množina. Ve chvíli, kdy její chybovost začne narůstat, se učení zastaví.

První je optimalizační *funkce konjugovaných gradientů* (scaled conjugate gradient, `trainscg`), jejíž výhodou je, že příliš nezatěžuje paměť.

Další funkcí je *Levenberg-Marquardt* (`trainlm`), která je oproti funkci konjugovaných gradientů časově úspornější za cenu větších paměťových nároků.

Třetím algoritmem je *Bayesovské pravidlo* (Bayesian Regulation, `trainbr`), které se od předchozích dvou liší tím, že nepracuje s validační množinou. Obecně o něm lze říci, že dobře generalizuje malé nebo komplikované datasety, případně datasety obsahující šum. Trénink končí při minimalizaci vah. [15]

V Matlabu existují i další algoritmy pro algoritmus zpětného šíření chyby, nicméně tři výše uvedené jsou nejrychlejší.

U uvedených optimalizačních funkcí můžeme mj. nastavovat [15]:

- `showWindow` otevření trénovací GUI, odkud je možné sledovat vizualizaci procesu učení,
- `max_fail` maximální chyba sítě,
- `goal` cíl účelové funkce,
- `min_grad` minimální gradient účelové funkce,
- `epochs` maximální počet cyklů,
- `time` maximální doba tréninku.

Poslední čtyři parametry stanoví ukončení učení sítě, a to i přes to, že se chybovost validační funkce nezačala horšit.

```
trainFcn = 'trainlm'                    % volba trénovací funkce
```

Volba počtu skrytých neuronů

Posledním faktorem, který nám určuje dopřednou síť, je počet skrytých neuronů. Problematika volby počtu skrytých neuronů byla nastíněna v kapitole 2.1. V tomto bodě tedy můžeme vytvořit samotnou síť. To se dělá za pomoci funkce `fitnet`, jejímiž argumenty jsou počet neuronů ve skryté vrstvě a zvolená optimalizační funkce. V tuto chvíli Matlab vytváří objekt sítě.

```
hiddenLayerSize = 2;                    % počet neuronů ve skryté vrstvě
net =                                   % vytvoření sítě
fitnet(hiddenLayerSize,trainFcn);
```

Zpracování dat

Při vytvoření objektu sítě Matlab automaticky přiřadí síti funkce, které upravují data do vhodné podoby pro klasifikaci. V případě příliš velkých dat totiž budou váhové koeficienty velmi malé, což v důsledku znamená mnohem delší dobu potřebnou k natrénování neuronové sítě.

```
net.input.processFcns = {'removeconstantrows','mapminmax'};
net.output.processFcns = {'removeconstantrows','mapminmax'};
```

Rozdělení dat

Následující část kódu určuje rozdělení množiny tréninkových vzorů na trénovací, testovací a validační část. Vzorky do jednotlivých podmnožin mohou být na základě zadaného poměru rozděleny náhodně či podle konkrétního pravidla.

```
net.divideFcn = 'dividerand';           % náhodné rozdělení dat
net.divideMode = 'sample';
net.divideParam.trainRatio = 70/100;    % poměr trénovací množiny
net.divideParam.testRatio = 15/100;     % poměr testovací množiny
net.divideParam.valRatio = 15/100;      % poměr validační množiny
```

Volba účelové funkce

Další část kódu, volba účelové funkce, je zpravidla taktéž generovaná automaticky, a to na základě zvolené optimalizační funkce. Dále jsou definovány funkce, vykreslující průběh učení na základě parametrů.

```
net.performFcn = 'mse';                 % účelová funkce
net.plotFcns = {'plotperform',...       % druhy vykreslovaných grafů
'plottrainstate','ploterrhist',...
'plotregression', 'plotfit'};
```

Trénink a testování sítě

V poslední části se dostáváme k samotnému tréninku sítě a jejímu testování na základě předchozích parametrů. Trénink probíhá pomocí funkce `train`, jejímiž argumenty je síť, kterou jsme vytvořili, trénovací vzory a požadované výstupy. Na závěr síť zobrazíme.

Testování probíhá tak, že síti předložíme vzory z testovací množiny a porovnáváme je s požadovanými výstupními hodnotami.

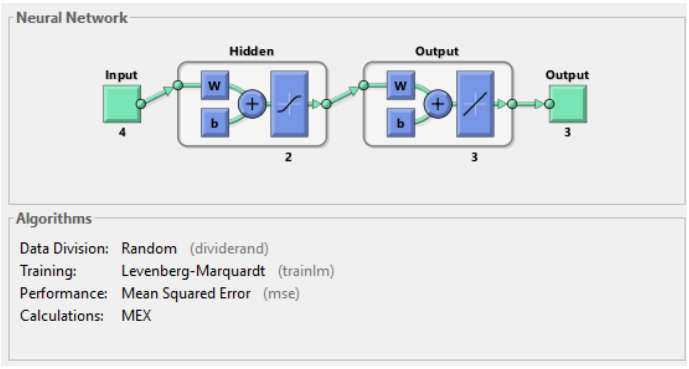
```
[net,tr] = train(net,x,t);              % metoda pro trénování sítě

y = net(x);                             % testování sítě
e = gsubtract(t,y);
performance = perform(net,t,y);
```

5.2.3 Charakteristiky natrénované sítě

Následující obrázky a informace jsou popisovány na základě výstupních charakteristik sítě natrénované dle ukázkového algoritmu z předchozí podkapitoly. Síť je uložena v elektronické

příloze v `backpropagation/Ukazkova_sit.mat`. Během trénování sítě dle předchozího algoritmu vyskočí interaktivní okno (Obr.17, Obr.18). První pole (Obr.17) ukazuje schematicky znázorněnou topologii sítě, v tomto konkrétním případě tedy síť se čtyřmi vstupy, skrytou vrstvou o dvou neuronech a sigmoidální transformační funkcí a tři výstupy. V dalším bloku vidíme popis použitých algoritmů, jejichž význam jsme si osvětlili v předchozí podkapitole. Na Obr.18 můžeme sledovat průběh učení: počet proběhlých cyklů (epoch), čas (time), snížení chybovosti sítě v průběhu učení (performance), snižování hodnoty gradientu v průběhu učení a počet po sobě následujících správných klasifikací validační množiny (validation checks). Poslední pole nám dává možnost rozkliknout grafy související s vlastnostmi sítě, které si nyní postupně popíšeme.



Obr. 17) Informační okno: topologie sítě, použité algoritmy

Obr. 18) Výstup trénování: informace k tréninku sítě, aktivace grafů

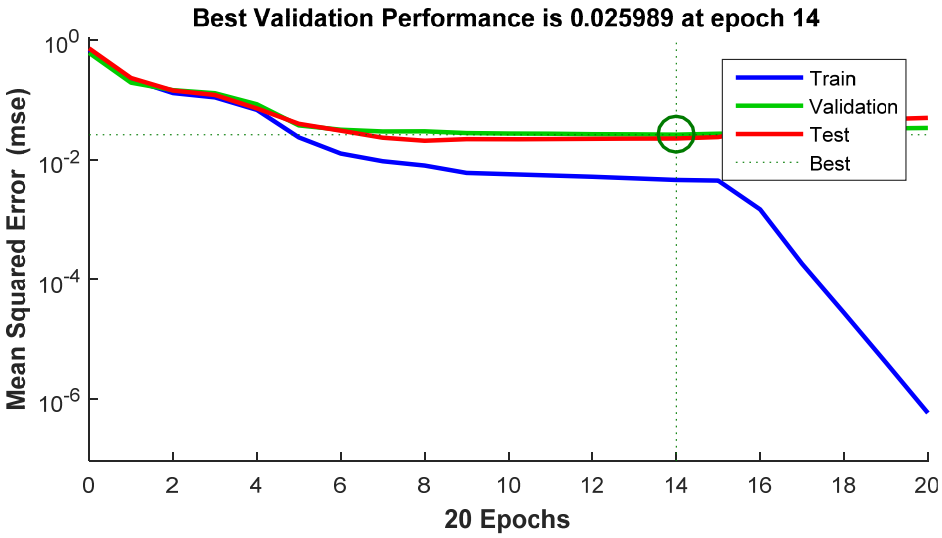
Prvním grafem, který si můžeme prohlédnout, je vývoj střední kvadratické chyby v průběhu trénování (Obr. 19). Jelikož jsme pro učení použili metodu Levenberg-Marquardt, tak trénink končí ve chvíli, kdy se střední kvadratická chyba u trénovací množiny začne zvyšovat. V tomto případě ve 14. iteraci, kdy síť považujeme za nejpřesnější.

V pořadí druhým odkazem, označeným jako Training State, jsou grafy vývoje gradientu, koeficientu učení a správnosti klasifikace validační množiny, které můžeme vidět na Obr.20. Zde pozorujeme, stejně jako u předchozího grafu, že kolem 14. cyklu se validační požadované výstupy přestávají shodovat s odezvou sítě.

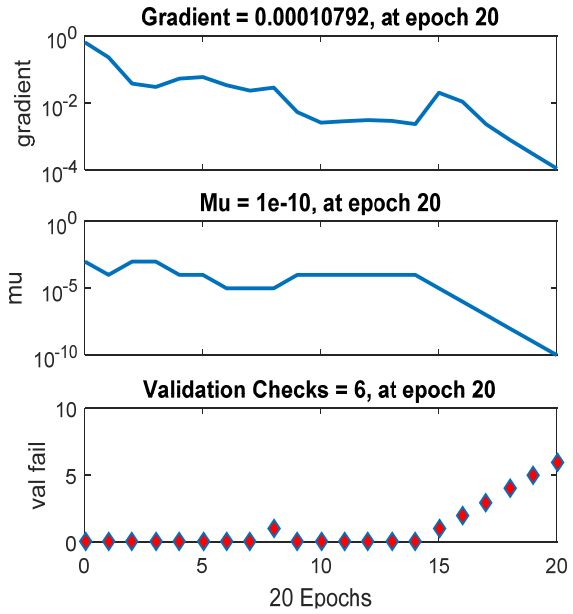
Dalším generovaným grafem je chybový histogram (Error Histogram) na Obr.21, který znázorňuje odchylku všech cílových výstupů od jednotlivých skutečných výstupů (máme tedy 3*150 hodnocených instancí).

Dále jsou k dispozici regresní grafy (regression) na Obr. 22, které určují míru shody výstupů generovaných sítí s výstupy požadovanými. Důležitou roli zde hraje graf testovací množiny, který znázorňuje, jak dobře síť klasifikuje neznámé vzory. Žádoucí jsou hodnoty regres co nejbližší 1.

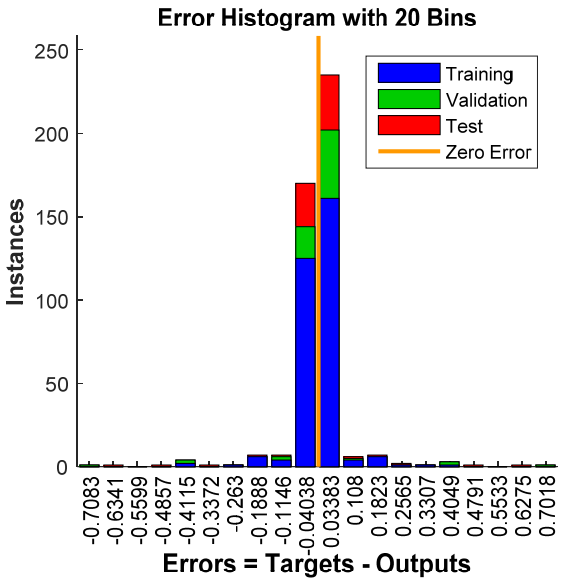
Poslední z možností (plotfit) generuje graf jen pro sítě s jednorozměrným vstupem, tudíž nemá pro naši síť o čtyřech vstupech význam.



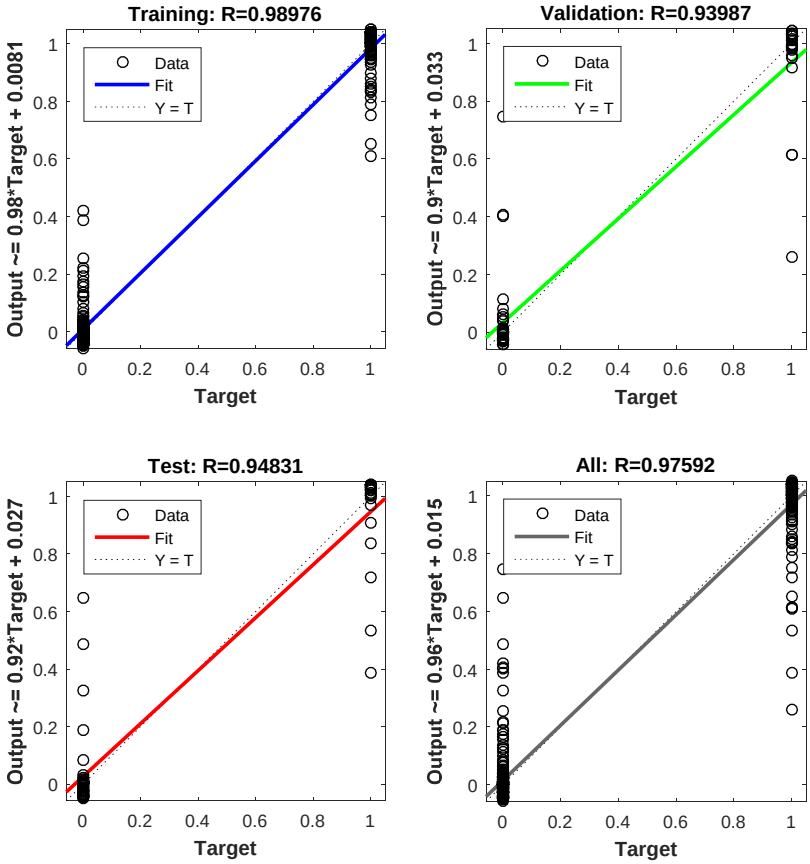
Obr. 19) Časový vývoj střední kvadratické chyby



Obr. 20) Časový vývoj gradientu, koeficientu učení a chybné klasifikace ověřovací množiny



Obr. 21) Chybový histogram



Obr. 22) Regresní grafy

5.2.4 Praktické příklady

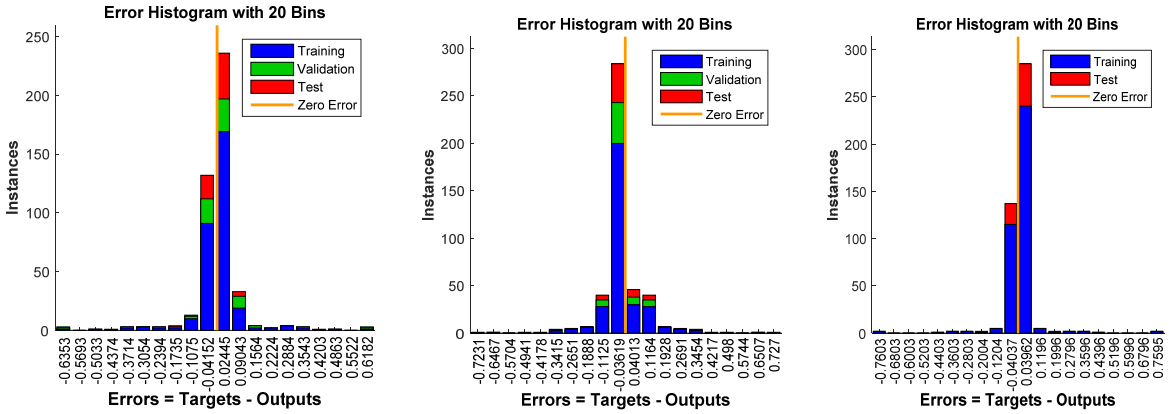
V této části jsou na Fišerově datasetu ukázány různé výstupy pro různá nastavení sítě. Než přistoupíme k samotným testovaným sítím, připomeňme si, že rozdělení vzorů testovací množiny, stejně tak jako počáteční nastavení váhových koeficientů, jsou náhodné. Proto při testování téhož algoritmu víckrát dostáváme různé hodnoty. Každý z použitých algoritmů je uložen v elektronické příloze včetně konkrétní sítě, kterou vygeneroval, pod zde určeným pořadovým číslem.

V rámci jednotlivých experimentů budeme zkoumat nejdříve vliv změny přechodové funkce na výkon sítě, poté budeme měnit počet neuronů ve skryté vrstvě a na závěr budeme měnit poměry jednotlivých podmnožin tréninkové množiny. U každého použitého algoritmu je navíc pomocí funkce '`tic toc`' určena doba učení.

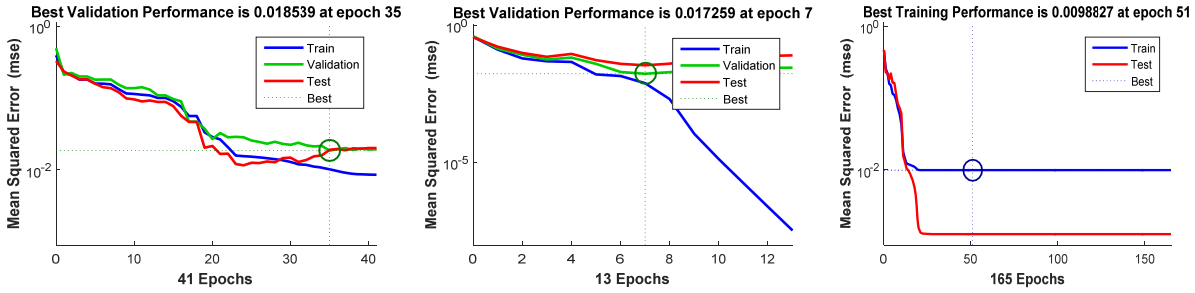
Vliv přechodové funkce:

Nejdříve zkoumáme vliv přechodové funkce za konstantního počtu neuronů ve skryté vrstvě (2 neurony) a při výchozím nastavení poměru tréninkové, testovací a validační množiny (70/15/15%). Zkoumané údaje jsou shrnuty v Tab.1).

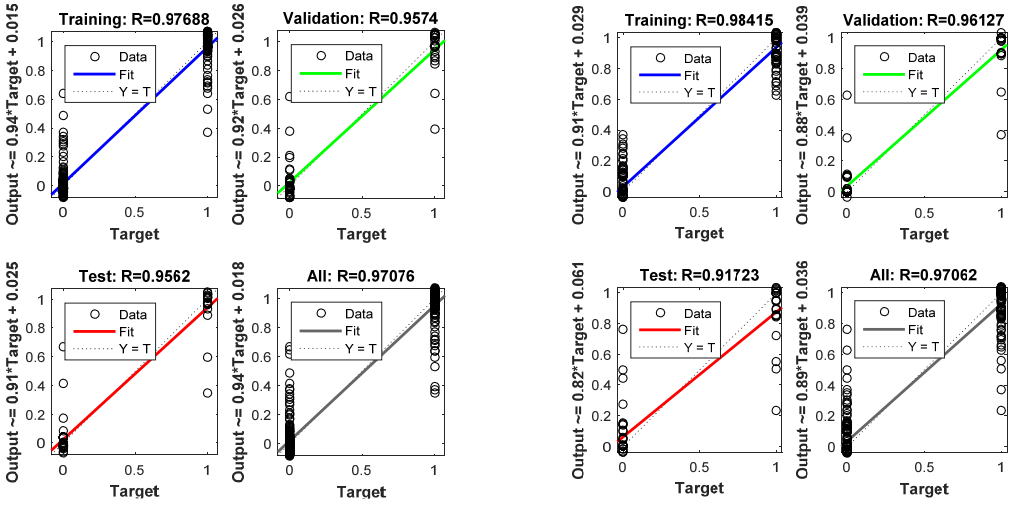
- algoritmus1 = metoda konjugovaných gradientů (`trainscg`),
- algoritmus2 = metoda Levenberg-Marquardt (`trainlm`),
- algoritmus3 = Bayesovská metoda (`trainbr`).



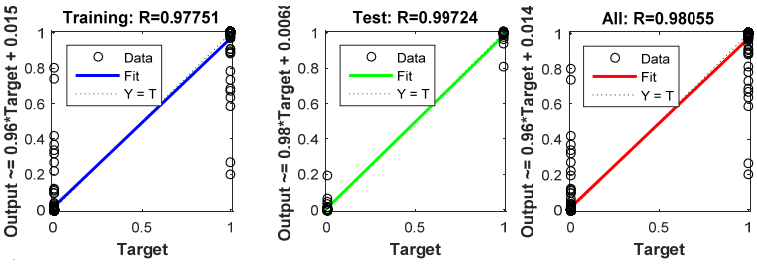
Obr. 23) Chybové histogramy algoritmů 1, 2 a 3



Obr. 24) Vývojové grafy algoritmu 1, 2 a 3



Obr. 25) Regresní grafy algoritmů 1 a 2



Obr. 26) Regresní grafy algoritmu 3

Tab 1) Tabulka pro porovnání hodnot algoritmů 1, 2 a 3

	Doba učení	Regrese testovací podmnožiny	Celková regrese tréninkové množiny	Nejlepší validační hodnoty
trainscg	26.61s	0.95	0.97	0.0185
trainlm	11.76s	0.92	0.97	0.0172
trainbr	97.65s	0.99	0.98	0.0099

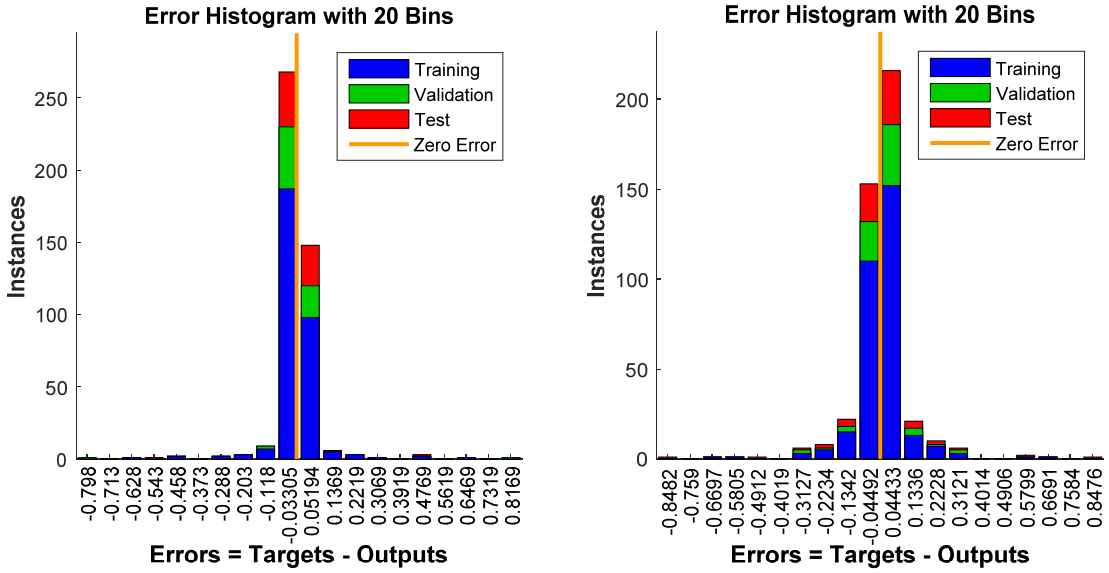
Z tabulky i z grafů vidíme, že zdaleka nejlepších výsledků pro námi zvolená data dosahuje Bayesovská metoda. Tento výkon je však za cenu několikanásobně delší doby učení. V našem případě malého datasetu o 150 vzorech není trénovací doba tak závažná, nicméně pro rozsáhlé databáze by tento fakt již mohl představovat problém. Další dvě zkoumané metody jsou, co se týče výkonu srovnatelných hodnot, avšak Levenberg-Marquardtova metoda je významně rychlejší. Pro další srovnávání tedy použijeme právě ji jako kompromis mezi rychlostí učení a výkonem sítě.

Vliv počtu neuronů ve skryté vrstvě

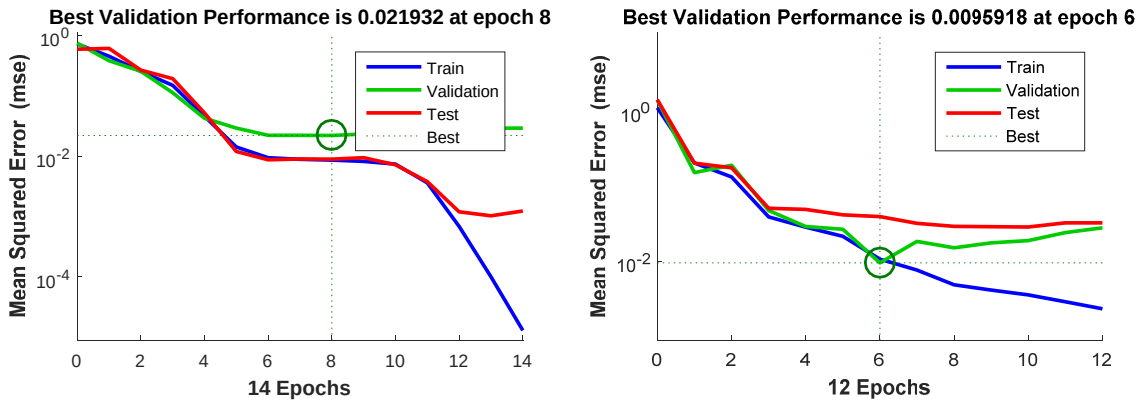
Počtem neuronů pro skrytou vrstvu jsme se okrajově zabývali v rámci kapitoly o metodě zpětného šíření chyby. Závěrem bylo, že neexistuje univerzálně aplikovatelný vzorec. V této úloze se tedy pokusíme demonstrovat vliv počtu neuronů ve skryté vrstvě na výkon sítě. Ve všech sítích bude aplikována Leven-Marquardtova metoda a bude držen poměr tréninkové, testovací a validační podmnožiny na 70:15:15.

Jelikož algoritmus pro dva neurony ve skryté vrstvě s příslušným nastavením byl již aplikován u dřívějšího zkoumání, zavedeme jeho výsledky pouze do srovnávací tabulky.

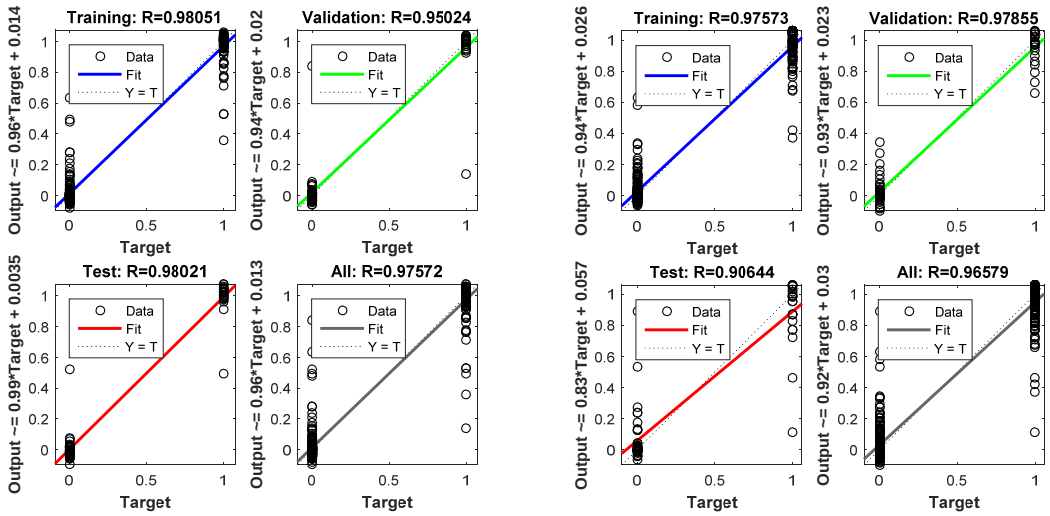
- algoritmus5 - síť s pěti skrytými neurony,
- algoritmus6 - síť s deseti skrytými neurony.



Obr. 27) Chybové histogramy algoritmů 4 a 5



Obr. 28) Vývojové grafy algoritmů 4 a 5



Obr. 29) Regresní grafy algoritmů 4 a 5

Tab 2) Tabulka pro porovnání hodnot algoritmů 2, 4 a 5

	Doba učení	Regrese testovací podmnožiny	Celková regrese tréninkové množiny	Nejlepší validační hodnoty
Dva skryté neurony	11.76s	0.92	0.97	0.0172
Pět skrytých neuronů	9.13s	0.98	0.97	0.0219
Deset skrytých neuronů	8.75s	0.91	0.97	0.0095

Z tabulky vidíme, že testované sítě jsou srovnatelné, co se týče celkové regrese dat. Nejrychleji byla naučena síť s deseti skrytými neurony, avšak časové rozdíly v učení jednotlivých sítí jsou malé, proto je můžeme zanedbat. Zajímavé je, že síť s deseti neurony vykazuje nejlepší validační hodnoty, avšak regrese testovací podmnožiny vykazuje nejméně přívětivé výsledky z testovaných sítí. Jelikož pro nás je nejdůležitější schopnost klasifikovat nenaučené vzory, je pro nás právě testovací podmnožina rozhodujícím faktorem. Z nabízených možností se tedy jeví jako nejlepší volba síť s pěti neurony ve skryté vrstvě.

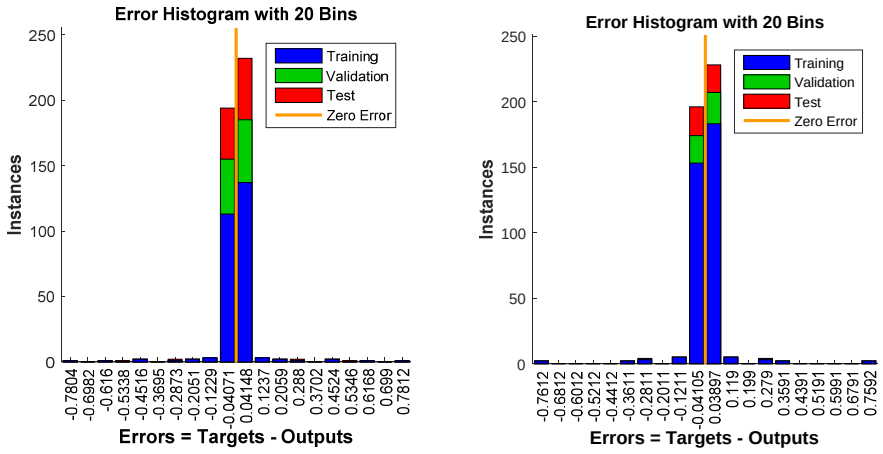
Získané výsledky však nemusí být zcela vypovídající z důvodu, že počáteční nastavení vah je u metody zpětného šíření chyby zcela náhodné. Pro srovnání zde proto uvádíme tabulku Tab.3) hodnot získaných průměrem deseti různých učení sítě.

Tab 3) Průměrné hodnoty pro deset opakování algoritmů 2,4 a 5

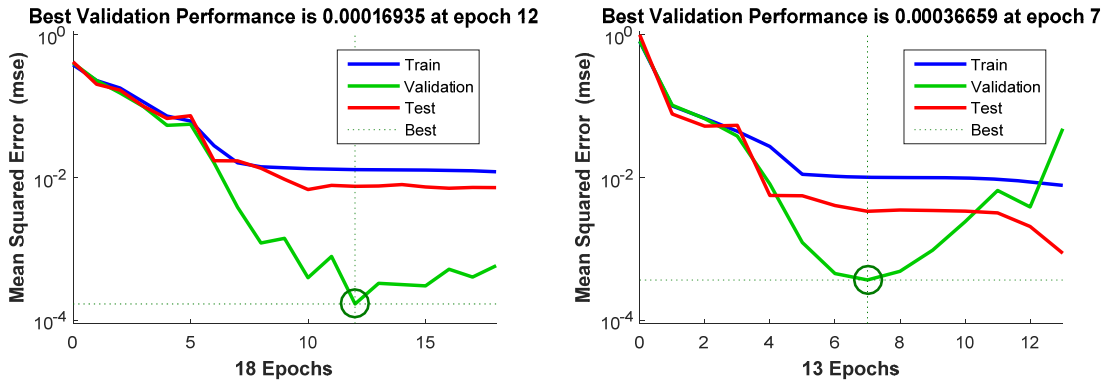
	Doba učení	Regrese testovací podmnožiny	Celková regrese tréninkové množiny	Nejlepší validační hodnoty
Dva skryté neurony	7.7s	0.94	0.98	0.0138
Pět skrytých neuronů	7.79s	0.94	0.97	0.0156
Deset skrytých neuronů	5.66s	0.97	0.97	0.0159

Vliv rozdělení tréninkové množiny

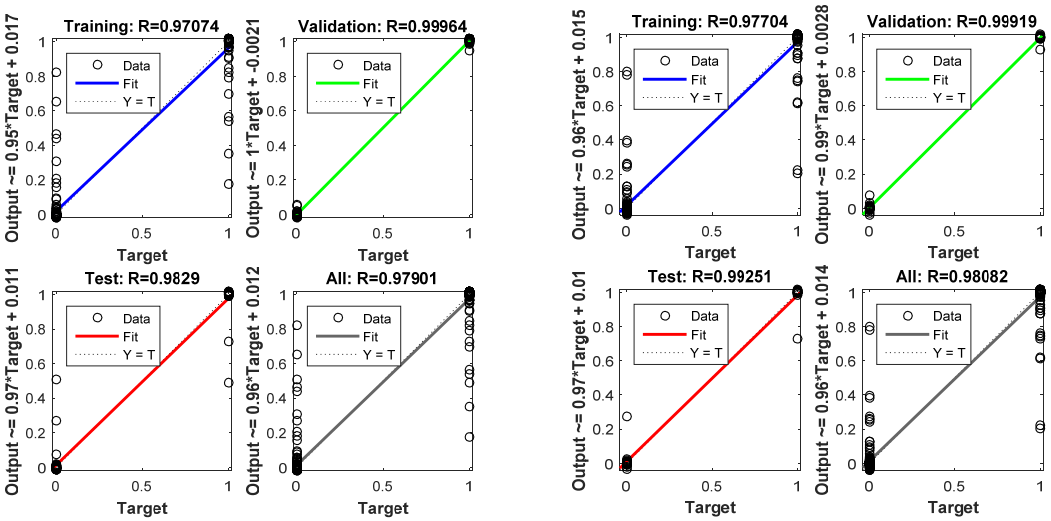
V poslední části zabývající se metodou backpropagation budeme zkoumat vliv poměru mezi trénovací, testovací a validační podmnožinou. V rámci předchozích příkladů již máme znázorněny výstupy sítě pro poměr 70:15:15, který tedy opět zaneseme do tabulky pro srovnání. Další testovací poměry budou 60:20:20 (algoritmus6) a 80:10:10 (algoritmus7). Pro tyto hodnoty bude v síti pět neuronů ve skryté vrstvě a síť bude učena Levenberg-Marquardtovou metodou.



Obr. 30) Chybové histogramy algoritmů 6 a 7



Obr. 31) Vývojové grafy algoritmů 6 a 7



Obr. 32) Regresní grafy algoritmu 6 a 7

Tab 4) Tabulka pro porovnání hodnot algoritmů 4, 6 a 7

	Doba učení	Regrese testovací podmnožiny	Celková regrese tréninkové množiny	Nejlepší validační hodnoty
70:15:15	9.13s	0.98	0.97	0.0219
60:20:20	11.94s	0.98	0.97	0,00017
80:10:10	8.81s	0.99	0.98	0.00037

Z uvedených dat vidíme, že doba natrénování se nijak významně neliší. Velmi zajímavým jevem jsou velmi nízké hodnoty chyby u validační množiny. Z uvedených hodnot lze konstatovat, že jak v rámci regrese testovací podmnožiny, tak i celkové regrese je pro Iris dataset konfigurace 80:10:10.

5.2.5 Diskuze k dosaženým výsledkům

Na základě sedmi různých konfigurací jsme zjistili, že nejvýhodnější konfigurací speciálně pro Fišerovu databázi kosatců je síť učena na základě Levenberg-Marquardtovy metody s pěti neurony ve skryté vrstvě a s rozdělením tréninkové množiny v poměru 80:10:10.

Je otázkou, do jaké míry tento výsledek ovlivňuje náhodné nastavení vah či rozdělení do podmnožin. Nicméně aby se předešlo prezentaci extrémně neodpovídajících výsledků způsobených náhodnou konfigurací, byl každý z prezentovaných algoritmů spuštěn víckrát. Následně bylo porovnáno, zda se hodnoty při opakované inicializaci příliš neliší od hodnot uvedených.

Dále musíme uvážit, že jsme netestovali všechny možné hodnoty a kombinace, třeba sítě se třemi, nebo šesti neurony. Také například metoda konjugovaných gradientů v kombinaci s jiným počtem neuronů ve skryté vrstvě a jiným poměrem podmnožin by generovala lepší výsledky, než kterých jsme dosáhli. Nicméně testování všech možností sahá za hranice této práce.

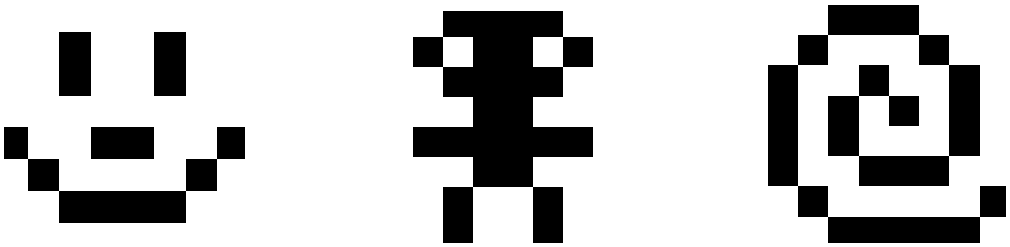
5.3 Hopfieldova síť a rozpoznávání vzorů

Hopfieldova síť nachází uplatnění v klasifikaci vzorů. V našem případě se jedná o obrázky o rozměrech 8x8 pixelů. Nejdříve budou obrázky zpracovány tak, aby odpovídaly vstupům Hopfieldovy sítě. Dále pak bude popsán kód v Matlabu pro naučení sítě našimi vzory. Poslední částí této kapitoly bude ověření správné funkčnosti sítě, kdy budou předkládány poškozené obrázky, přičemž síť bude reprodukovat obrázky původní. Data k příkladu jsou součástí elektronické přílohy ve složce Hopfieldova síť.

5.3.1 Vzory, jejich předzpracování a natrénování sítě

Jak bylo vysvětleno v kapitole 3.5, Hopfieldova síť má omezenou kapacitu učení danou vzorcem (28). Aplikujeme-li tento vzorec na naši síť o rozměrech 8x8, tedy o 64 vstupech, zjistíme, že do sítě mohou být uloženy až tři různé vzory.

Vzory, které budeme síť učit můžeme vidět na Obr. 33. Pro použití v Matlabu jsou uloženy jako bitové mapy pod názvy `smajlik.bmp, clovek.bmp, spirala.bmp.`



Obr. 33) Vzory pro naučení neuronové sítě

Vzorové obrázky musíme zpracovat tak, aby mohli být použity jako vstupy Hopfieldovy sítě. To znamená konvertovat je na vektory o hodnotách z {-1,1}. K tomu nám slouží krátký kód, který je k nalezení jako `nacteni_vzoru.m` v elektronické příloze.

```
obr=imread('smajlik.bmp');          % načtení obrázku ze souboru *.bmp
obr=rgb2gray(obr);                  % převod obr. dat na odstíny šedé
obr=im2double(obr);                 % převod na hodnoty double{0,1}
                                    % z=z==255
matice=ones(size(obr)).*-1;         % prahování na černobílou
                                    % -1 černá / 1 bílá
matice(obr>0.9)=1;                  % hodnota prahu zvolena 0.9

vektor1=reshape(matice,[1,64]);     %tranformace matice na řádkový vektor
```

Nyní tedy máme data zpracovaná a můžeme začít trénovat. Učení Hopfieldovy sítě je v Matlabu velmi snadné, v příloze lze najít pod názvem `trenink_site.m`.

```
T=[vektor1;vektor2;vektor3]';       %sestaveni treninkove mnoziny
sit=newhop(T);                      %trenink site
```

Síť, kterou jsme tímto způsobem natrénovali, je uložena pod názvem `Hopfieldova_sit.mat`.

5.3.2 Algoritmus pro klasifikaci

Pro klasifikaci opět potřebujeme předzpracované vstupy, tentokráte získané z poškozených obrázků, které chceme klasifikovat. Předzpracování probíhá stejně jako u vzorů. Pro načtení poškozených obrázků je v elektronické příloze soubor `nacteni_vstupu.m`.

Simulace sítí obecně v Matlabu probíhá za pomoci funkce sim. následovně[15]: `[Y,Xf,Af] =sim(net{Q TS},Xi,Ai)`, kde net je konkrétní síť, `Q` je počet vektorů, které chceme klasifikovat, `TS` je počet časových kroků, `Xi` jsou inicializační podmínky a `Ai` jsou vektory, které vyhodnocujeme.

V našem konkrétním případě budeme testovat obrázky každý zvlášť, počet vektorů, které chceme klasifikovat, odpovídá 1. Počet časových kroků není zadán, jelikož chceme, aby simulace skončila, až je dosaženo ustálení. Následuje vektor obrázku, který chceme klasifikovat.

```
[Y,Pf,Af] = sim(net,1,[],[test]); %simulace sítě
```

Po ukončení simulace má síť v proměnné Y uložen výstup simulace sítě. Ten chceme převést na původní binární matice, abychom mohli z výstupu opět sestavit obrázek.

```
Y=Y';

A=reshape(Y,[8,8]);                %převedení výstupu zpět na matici 8x8

A=A';

A(A>0)= 1;                         %zaokrouhlení hodnot větších než 0 na 1
A(A<0)=-1;


A=reshape(A,[8,8]);

vystup=A';
```

Nyní výstup převedeme zpět do binárního obrázku, který je uložen pod názvem `výstup.bmp` do výchozího adresáře a zobrazíme.

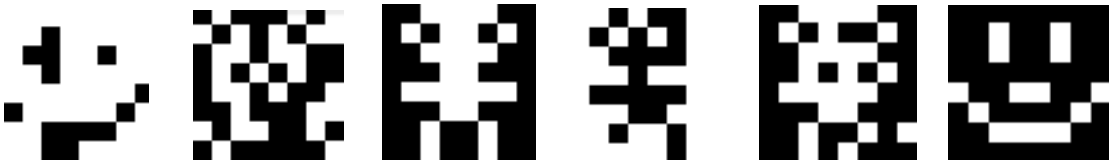
```
imwrite(vystup,'vystup.bmp');          % převedení na obrázek a uložení

z=round(vystup);                       % zobrazení obrázku
z=[z  z(:,end)]; z=[z; z(end,:)];
[x,y] = meshgrid(1:1:8, 1:1:8);
surf(x,y,z)
view(90,90)
colormap gray
```

5.3.3 Konkrétní aplikace

V kapitole o Hopfieldově síti jsme psali, že síť je schopná rozeznávat i velmi poškozené či barevně invertované obrázky. Ke každému z tréninkových obrázků byla vytvořena jeho poškozená verze, dále u jednoho byla vytvořena velmi poškozená verze a jeden vzor byl

invertován. Příslušné obrázky můžeme vidět na Obr. 34. Nyní se tedy pokusíme tyto verze obrázků klasifikovat.

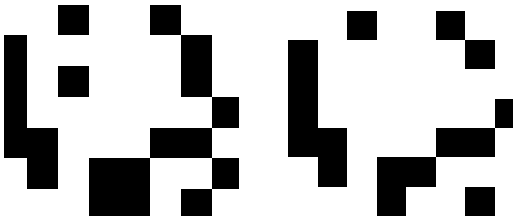


Obr. 34) Zleva: poškozený smajlík, hodně poškozená spirála, invertovaný člověk, poškozený člověk, invertovaný poškozený člověk, invertovaný smajlík

Jelikož výstupem klasifikace je původní obrázek, který již zde máme výše, zaznamenáme úspěšnost klasifikace pouze do tabulky. Pro případné ověření stačí spustit příslušný skript z elektronické přílohy.

Tab 5) Úspěšnost klasifikace Hopfieldovy sítě

	Jméno skriptu	Správnost klasifikace
Poškozený smajlík	`Hop1.m`	ANO
Hodně poškozená spirála	`Hop2.m`	ANO
Invertovaný člověk	`Hop3.m`	NE
Poškozený člověk	`Hop4.m`	ANO
Invertovaný poškozený člověk	`Hop5.m`	NE
Invertovaný smajlík	`Hop6.m`	NE (klasifikován jako člověk)



Obr. 35) Špatná klasifikace invertovaného člověka a invertovaného poškozeného člověka

5.3.4 Diskuse k dosaženým výsledkům

Při testování naší neuronové sítě jsme ověřili, že síť je schopná klasifikovat poškozené i velmi poškozené obrázky. Síť si však nevede dobře na inverzních vzorech.

Prvním případem je inverzní smajlík, který by měl ukazovat na vzorového smajlíka, jelikož inverzní hodnoty vzorů by měly být taktéž atraktorem v rámci energetické plochy. Avšak jak vidíme vzor člověka, je pravděpodobně pro vektor inverzního smajlíka silnějším atraktorem.

U inverzního obrázku člověka a poškozeného obrázku inverzního obrázku člověka síť dokonce indikuje nenaučený vzor. To si můžeme vysvětlit například vznikem fantomů, kterého bychom se v případě nutnosti mohli pokusit odučit některým ze specifických algoritmů k tomu určených. Jelikož však odučovací algoritmy Hopfieldovy sítě přesahují tento text a nemají v rámci Matlabu jednoduchou implementaci, smíříme se pouze s jejich konstatováním.

5.4 Aplikace Kohonenovy mapy na Iris dataset

Poslední praktickou aplikací je klasifikace dat z Fišerovy databáze pomocí Kohonenovy samoorganizační mapy. Všechny soubory použité v tomto úseku jsou uloženy v elektronické příloze ve složce Kohonen.

Z kapitoly 4 víme, že data v rámci samoorganizačních map jsou tříděna pouze na základě vstupů, požadované hodnoty tedy nejsou potřeba. Jelikož SOM mají stejnou podobu vstupů jako algoritmus backpropagation, tedy matice, kde řádky odpovídají jednotlivým vzorům, můžeme vzít již předzpracovaná data za metody backpropagation. Přejdeme tedy rovnou k popisu kódu.

5.4.1 Obecný popis kódu a význam výstupů

Nyní si popíšeme jednoduchý algoritmus pro vytvoření a naučení Kohonenovy samoorganizační mapy. Klasicky začneme načtením vstupů. Následuje určení rozměrů výstupní vrstvy, vytvoření sítě, učení sítě a testování sítě.

Jak víme z předchozích kapitol, Kohonenova samoorganizační mapa má dvě vrstvy. První vrstvou jsou vstupní neurony, nás však zajímá především druhá vrstva, která je představována dvourozměrnou mřížkou, na jejíchž uzlech jsou situovány výstupní neurony.

```
vstupy=load('vstupy.txt');              % načtení vstupů
x = vstupy';

dimension1 = 10;                        % rozměry výstupní mřížky
dimension2 = 10;
```

Dalšími nastavitelnými parametry sítě je počet kroků, ve kterých bude učena (diskrétní čas), velikost okolí neuronu, topologie výstupní vrstvy (z jakých tvarů se mřížka skládá-čtverce, hexagony,...) a způsob měření vzdálenosti mezi neurony [15]. Za pomoci těchto parametrů můžeme vytvořit Kohonenovu samoorganizační mapu.

```
coverSteps= 100;                                % počet diskrétních kroků
initNeighbor = 3;                               % nastavení velikosti
                                                % okolí
topologyFcn = 'hextop';                         % nastavení tvaru polí
                                                % mřížky
                                                % nastavení způsobu
distanceFcn='linkdist'                          % měření vzdálenosti

net = selforgmap([dimension1 dimension2],...    % vytvoření sítě pomocí
coverSteps, initNeighbor, topologyFcn,...       % parametrů výše
distanceFcn);
```

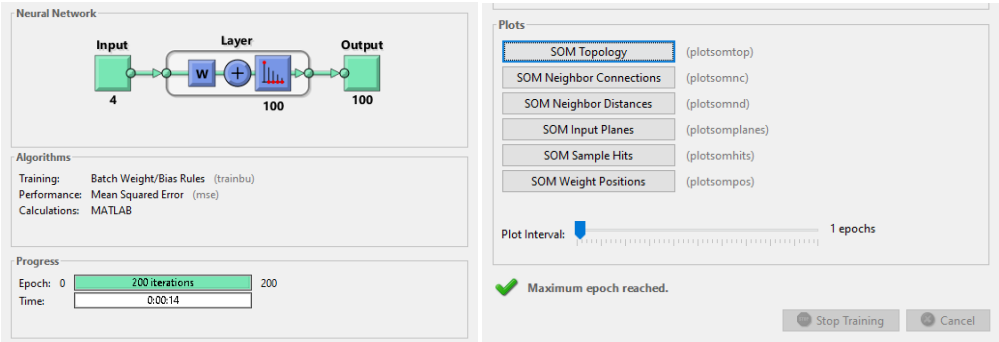
Dále je síť trénována za pomoci vstupů a testována.

```
[net,tr] = train(net,x);                % trénink sítě
y = net(x);                             % testování sítě
```

Tento algoritmus je možno nalézt v elektronické příloze pod názvem `nauceni.m`. Síť trénovaná pomocí tohoto kódu je k nalezení tamtéž pod názvem `Kohonenova_sit.mat`. Na této síti budou dále demonstrovány charakteristiky natrénované sítě.

5.4.2 Charakteristiky natrénované sítě

Stejně jako v případě dopředných sítí v průběhu trénování vyskočí okno obsahující grafy a údaje o právě natrénované síti. Pro Kohonenovu samoorganizační mapu vypadá okno dle Obr.36. V této části si osvětlíme význam příslušných grafů a údajů.



Obr. 36) Okno spuštěné během trénování SOM sítě

První pole znázorňuje topologické schéma sítě. V našem případě vidíme, že máme síť se čtyřmi vstupy, trénovanou rozdělenou na 100 shlukových oblastí a se 100 výstupními neurony. Druhá oblast indikuje, že jsme k trénování použili metodu trainbu určenou pro učení bez učitele za pomoci úpravy vah a biasů[15]. Další řádky nám říkají, že k určování chyby byla použita střední kvadratická odchylka a že výpočty provádí Matlab. Třetí oblast nás informuje o počtu cyklů, které již proběhly a o době, jakou výpočet trval. Poslední pole nám tradičně dává nabídku grafů vypovídajících o charakteristice sítě. Tyto grafy si nyní popíšeme.

Jak název napovídá, první z nabízených grafů (Topology) nám vykresluje topologii sítě, která je znázorněna na Obr. 37, kde vidíme rozměr mřížky 10x10 hexagonů. V druhém grafu (Neighbor connections) na Obr.38 jsou již na uzlech mřížky zakresleny neurony a červeně jsou naznačeny vzdálenosti mezi nimi dle námi zadané metriky, tedy jakožto přímá vzdálenost mezi neurony.

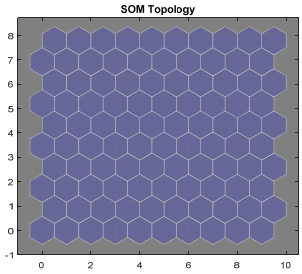
Třetí graf (Neighbor weight distances) na Obr.39 je již komplikovanější, jelikož znázorňuje vzdálenost modře zaznačených neuronů na základě váhových koeficientů, a to pomocí odstínů žluté barvy. Světlé odstíny indikují, že neurony jsou si ve váhovém prostou blízko, zatímco tmavé barvy signalizují velkou vzdálenost. Světlé oblasti tedy představují shluky neuronů s podobnými vstupy, v našem případě třídu kosatců. Naopak tmavé části grafu značí, že do těchto shlukových oblastí spadá málo nebo vůbec žádné vzory.

Čtvrtý odkaz generuje čtyři grafy (Input planes), viz. Obr.40. Každý graf je vykreslen pro jeden vstup neuronu a odpovídá váhovému příspěvku příslušného vstupu ke každému z neuronů výstupní sítě. Světle žluté oblasti indikují, že příslušný vstup má velký vliv na jeho výstup a naopak políčka v tmavých odstínech indikují, že příslušné výstupní neurony jsou na daném vstupu téměř, nebo zcela nezávislé.

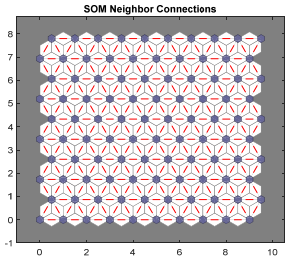
Pátá možnost v nabídce (Hits) na obrázku Obr. 41 je snazší na pochopení. Jednotlivá čísla v polích indikují, kolik vzorů do této oblasti spadá. Prázdná pole pak indukují hranici mezi jednotlivými třídami, kam nespadá žádná z tréninkových instancí kosatce.

Posledním z grafů indikujících zachycuje vzájemné polohy neuronů ve váhovém prostoru (Weight positions) na Obr.42. Zelené tečky znázorňují jednotlivé tréninkové kosatce,

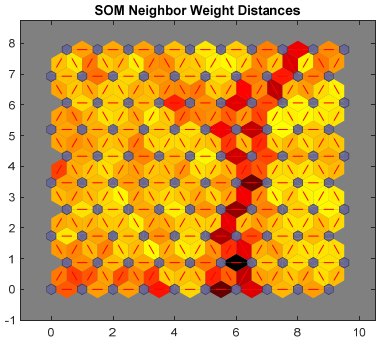
jejichž poloha je určena jejich parametry. Modré tečky symbolizují neurony, přičemž ty neurony, které spolu sousedí, jsou spojeny červenou čarou. Na první pohled se nám jejich sousedství nemusí zdát, nicméně uvědomme si, že takto vygenerovaná dvourozměrná mapa je zobrazením čtyřrozměrných dat.



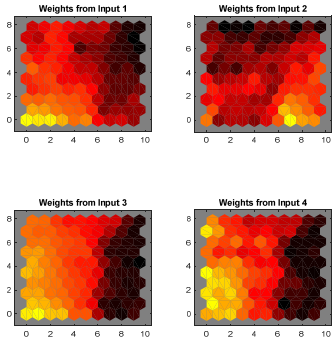
Obr. 37) Topologie výstupní vrstvy



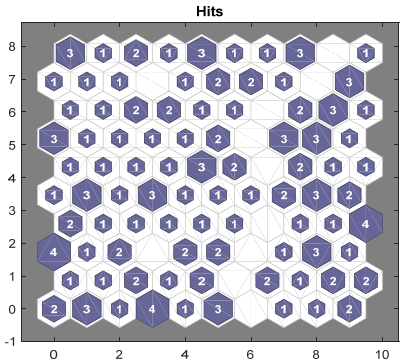
Obr. 38) Znázornění neuronů a vzdáleností mezi nimi v topologii



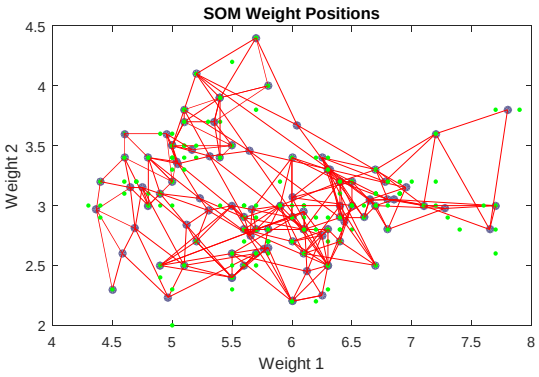
Obr. 39) Graf vzdáleností ve váhovém prostoru



Obr. 40) Grafy závislosti hodnoty výstupních neuronů na jednotlivých vstupních hodnotách



Obr. 41) Počty neuronů spadajících do oblasti



Obr. 42) Zobrazení sítě v dvojrozměrném výstupním prostoru

5.4.3 Charakteristiky sítě pro různá nastavení

V této části si ukážeme, zda má počáteční nastavení parametrů, které bylo vysvětleno v rámci popisu kódu, vliv na charakteristiky sítě. Budeme zkoumat vliv vybraných parametrů na síť.

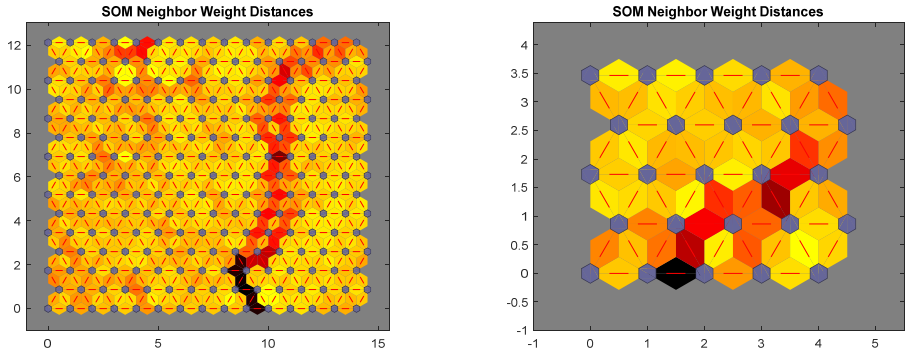
Konkrétně se bude jednat o počty neuronů v kompetiční vrstvě a velikost okolí neuronů. Ke každé síti budou vygenerovány grafy, které tuto síť charakterizují a které jsme si popsali v předchozí podkapitole. Záměrně budou voleny jiné parametry než u ukázkového kódu, abychom mohli naše výstupy porovnat i s touto sítí. Uváděné kódy pro natrénování sítí jsou opět součástí elektronické přílohy.

Vliv rozměrů na síť:

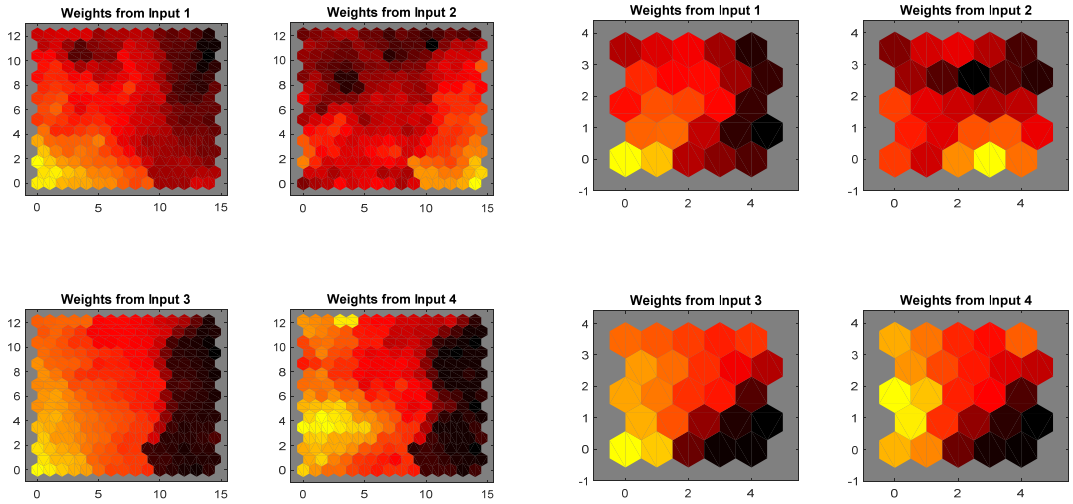
Prvním zkoumaným parametrem bude podoba grafů pro sítě o různých rozměrech. Zbylé nastavení bude stejné jako nastavení u ukázkového kódu, tedy 100 diskrétních kroků, okolí o velikosti 3, síť bude rozdělena na hexagony a vzdálenost mezi neurony bude měřena napřímo. Rozměry testovaných sítí:

- algoritmus1 o rozměrech 15x15 hexagonů ,
- algoritmus2 o rozměrech 20x20 hexagonů.

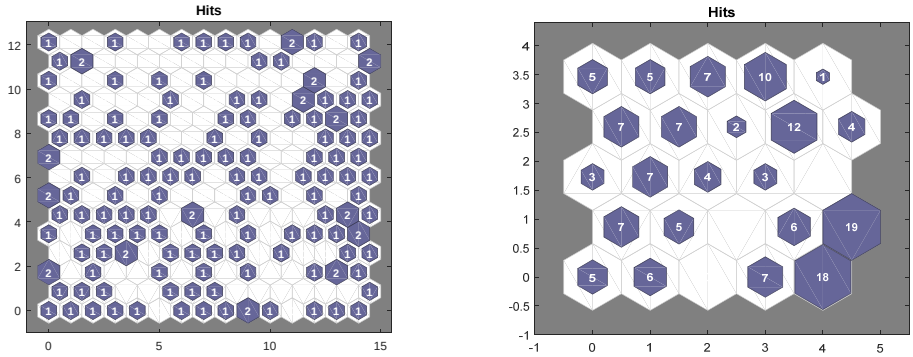
Grafy generované algoritmem 1 a 2:



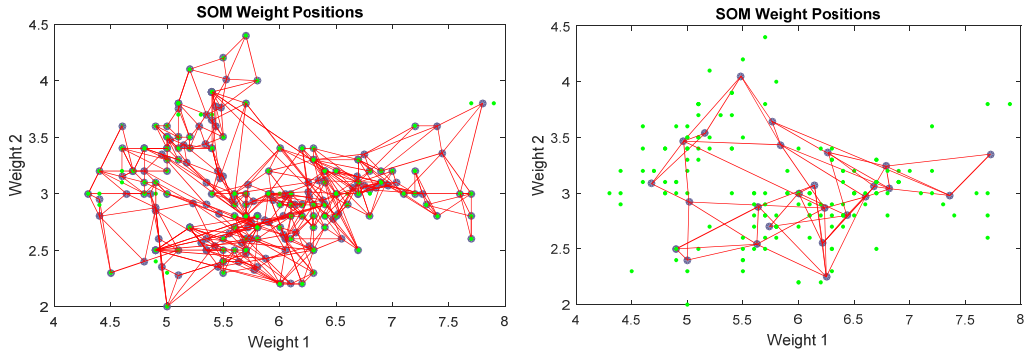
Obr. 43) Znázornění vzdáleností mezi neurony ve váhovém prostoru pro nastavení 1 a 2



Obr. 44) Vliv vstupů na výstupy pro zadaná nastavení



Obr. 45) Počty neuronů spadajících do jednotlivých oblastí



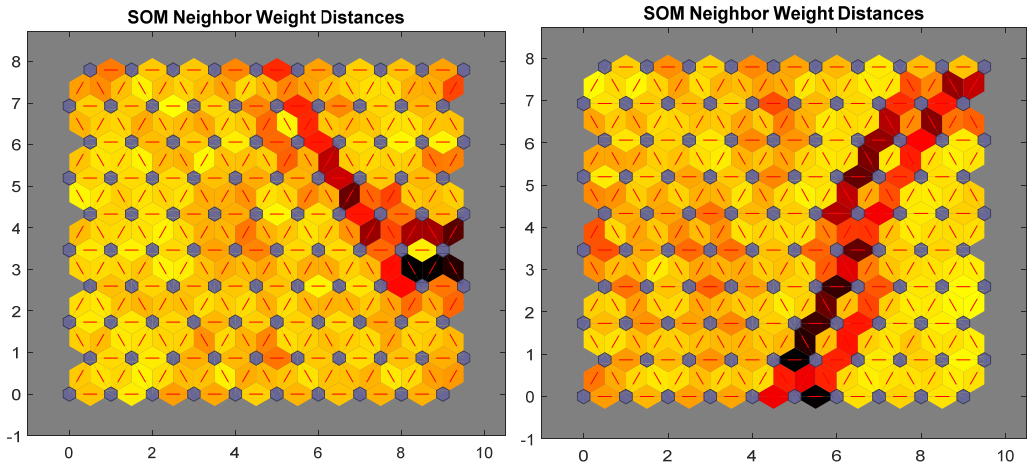
Obr. 46) Rozložení neuronů v rámci výstupní vrstvy

Na grafech můžeme pozorovat, že ačkoliv mají různá nastavení, která upravují pomyslná rozlišení, barevné rozložení je ve všech třech případech (jak u algoritmů 1 a 2, tak u ukázkového příkladu) nápadně podobná. Doba natrénování se pochopitelně s vyšším počtem neuronů prodlužuje.

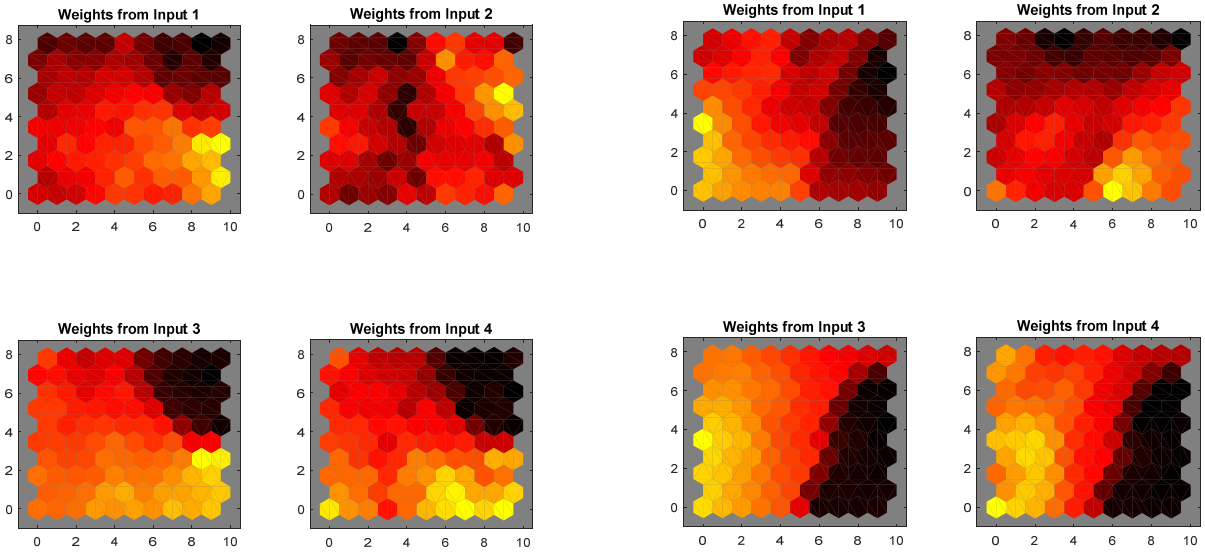
Vliv inicializované velikosti okolí na síť

Nyní se podíváme, jaký vliv má nastavení okolí na síť. Ostatní parametry budou konstantní, nastaveny shodně s ukázkovým kódem.

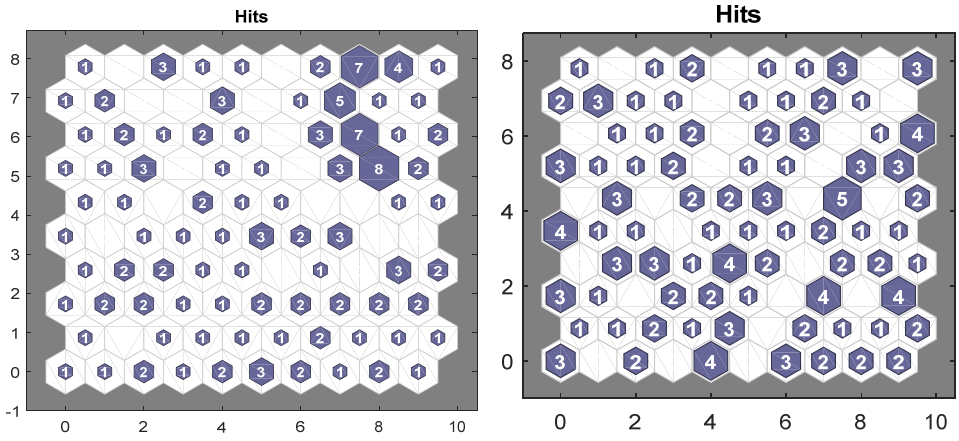
- Algoritmus3 - okolí neuronu = 10,
- Algoritmus4 - okolí neuronu = 1.



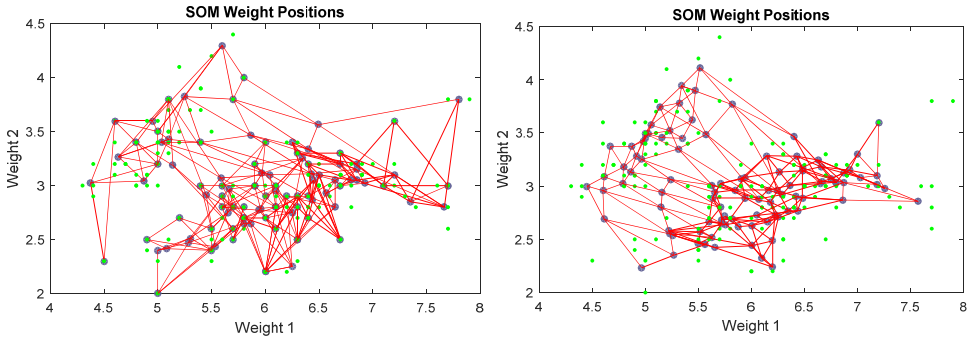
Obr. 47) Vzdálenosti neuronů ve váhovém prostoru pro algoritmy 3 a 4



Obr. 48) Příspěvky vstupů k výstupu pro algoritmy 3 a 4



Obr. 49) Počty vstupů spadajících do konkrétní oblasti pro algoritmy 3 a 4



Obr. 50) Rozložení neuronů ve výstupní vrstvě pro algoritmy 3 a 4

Porovnáme-li jednotlivé vygenerované grafy, může se nám zdát, že se velmi liší. Podíváme-li se však pozorněji, můžeme na všech vykreslených grafech pozorovat, že se jedná o v podstatě do jisté míry podobné rozdělení, které se od sebe liší zhruba o 90° otočení. Samozřejmě, že sítě ani po natočení nejsou vyloženě identické, nicméně si musíme uvědomit, stejně jako u předchozích algoritmů, že váhy sítě jsou inicializovány náhodně, a tedy ani u natrénování sítě 2x po sobě se stejnými parametry nedostaneme stejný výsledek.

5.4.4 Diskuze k dosaženým výsledkům

Z výše uvedených dat je možné usoudit, že nastavitelné parametry Kohonenovy samoorganizační mapy v porovnání s dopřednými sítěmi nehrají zas tak rozhodující roli. To je výhodné především proto, že nemusíme při učení sítě vhodnost nastavení mnohonásobně testovat a upravovat, abychom dosáhli požadovaných hodnot. Vygenerované grafy pro celkově pět různých nastavení se zdají být velmi podobné. Jak již bylo zmíněno výše, nesmíme opomínat vliv náhodného počátečního nastavení.

Jedinou výjimku tvoří graf pro velikost okolí neuronu, které je adaptováno zároveň při adaptaci vítězného neuronu. Hodnota 10 pro okolí v síti o rozměrech 10x10 hexagonů zahrnuje prakticky celou síť. Ta se proto zpočátku mění téměř celá a až po určitém počtu cyklů dojde ke snížení velikosti okolí. To může mít za následek právě odchylku příslušného grafu od ostatních.

6 ZÁVĚR

V textu práce jsme se seznámili s konceptem umělých neuronových sítí. První čtyři kapitoly představovaly rešeršní část práce. Pátá kapitola spolu s přílohou (zdrojová data, programy) představovala praktickou část práce.

První kapitola popsala stručný historický vývoj neuronových sítí od prvního představení umělého neuronu McCullochem a Pittsem, přes Hebovské učení, Minského první neuropočítač, Rosenbattovu činnost až po ADALINE a krizi neuronových sítí kvůli problému XOR. Dále byly v rámci první části rozebrány pojmy organizační dynamika, v rámci které byly vysvětleny pojmy topologie a cyklické a acyklické propojení neuronů. Následovala aktivní dynamka kde byla popsána funkce jednotlivého neuronu v rámci sítě a došlo k představení základních typů přenosových funkcí. Kapitolu uzavíral stručný popis učení s a bez učitele.

Obsah druhé kapitoly představuje stručné pojednání o optimalizační metodě backpropagatin pro učení dopředných neuronových sítí. Popis metody je rozdělen a popisován s pomocí dynamik. U organizační dynamiky je mimo jiné stručně popsána problematika volby počtu skrytých neuronů. V aktivní dynamice byl popsán výpočet potenciálu neuronu a sigmoidální přenosová funkce. Na závěr byla v rámci popisu adaptivní dynamiky popsána podstata učení s učitelem pomocí minimalizace chybové funkce realizované gradientní metodou.

Třetí kapitola prezentuje Hopfieldovu metodu učení, která je řešena za pomoci minimalizace energetické funkce a funguje jako asociativní paměť. Opět se zabýváme nejdříve organizační dynamikou, představovanou úplným propojením neuronů. Po organizační dynamice následuje adaptivní dynamika, kdy jsou vzory ukládány v podobě váhových koeficientů a reprezentují lokální minima energetické funkce. Dále je prezentován popis aktivní dynamiky v jejímž průběhu jsou vstupní data v rámci plochy energetické funkce posouvána do jejích minim, kde jsou identifikována s příslušným vzorem. Na závěr kapitoly práce nastiňuje problematiku kapacity Hopfieldovy sítě a vznik tzv. fantomů.

Kapitola o Kohonenových samoorganizačních mapách uzavírá rešeršní část textu. Naznačuje princip soutěžního učení realizovaného v rámci kompetiční vrstvy sítě. Jako první je tradičně popsána organizační dynamika, která skýtá dvě úplně propojené vrstvy. Následuje aktivní dynamika, jejímž výstupem je právě jeden excitovaný neuron ve výstupní vrstvě. Další podkapitolou je algoritmus pro naučení sítě, kde je objasněn pojem okolí neuronu a především je popsán koncept vzdálenosti vzorů a neuronů ve výstupní vrstvě, který je pro Kohonenovy samoorganizační mapy klíčový. Následně je uvedena metoda vektorové kvantizace učením, což je obdoba učení s učitelem, pro samoorganizační mapy a jsou popsány tři podoby vektorové kvantizace.

V praktické části byly realizovány všechny tři typy popisovaných sítí, přičemž byly napsány a okomentovány kódy ke každé z nich. U dopředné sítě a Kohonenovy mapy byly na základě hodnot z databáze kosatců testovány vlivy některých nastavitelných parametrů sítě. Ke každému testu byly doplněny grafy průběhu učení, které znázorňují některé charakteristické vlastnosti. U Hopfieldovy sítě byla data pro učení vytvořena v podobě bitových obrázků a testována na vytvořených poškozeních a inverzních verzích naučených

vzorů. Výsledkem testování byla tabulka hodnotící, zda síť správně klasifikovala předložený poškozený nebo inverzní vzor.

Součástí každé praktické části byla i diskuze k zjištěným výsledkům. U sítě řešené metodou backpropagation i Kohonenovy samoorganizační mapy bylo nutné u diskuze uvažovat náhodné nastavení vah při inicializaci sítí. Například u metody backpropagation díky náhodným váhovým konfiguracím testované hodnoty u vlivu počtu neuronů v síti neodpovídaly průměrným hodnotám naměřeným na deseti tréninkových cyklech. Dle testovaných hodnot (rychlost sítě, regrese testovací množiny, celková regrese sítě a nejnižší chybovosti na validační množině) bylo v rozsahu našeho měření zhodnoceno, že konkrétně pro Fišerovu databázi kosatců je nejlepší použít neuronovou síť s Levenberg-Marquardtovým učícím pravidlem, pěti neurony ve skryté vrstvě a poměrem tréninkové, testovací a validační podmnožiny 80:10:10.

V rámci praktické aplikace Hopfieldovy sítě jsme na testovaných vzorech ověřili, že pro poškozené vzory síť klasifikuje správně. Avšak tvrzení některých zdrojů, že Hopfieldova síť dokáže klasifikovat i vzory inverzní k naučeným vzorům se nám nepodařilo potvrdit a inverzní obrázky byly klasifikovány nesprávně.

Posledním testovaným typem sítě byly Kohonenovy samoorganizační mapy, kdy jsme testovali vliv rozměrů výstupní vrstvy sítě a zvolenou velikost okolí neuronu na klasifikační schopnosti sítě. Závěr ze získaných dat byl takový, že pro různé rozměry kompetiční vrstvy roztřídila síť zadané parametry kosatců s obdobným rozložením v rámci výstupní vrstvy, pouze na jiném pomyslném rozlišení sítě. Při testování vlivu rozměrů okolí na vlastnosti sítě byl znatelný rozdíl mezi výstupními, avšak zdá se, že rozdíl je způsoben natočením grafu k síti o velikosti okolí 10, než že by klasifikoval jinak.

V ohledu předloženého díla poznamenejme, že problematika umělých neuronových sítí je velmi rozsáhlá a uvedená práce zahrnuje jen malou část této velmi zajímavé oblasti umělé inteligence.

7 SEZNAM POUŽITÝCH ZDROJŮ

[1] EVA, Volná. Neuronové sítě 1. Ostrava: Ostravská univerzita v Ostravě, 2008.

[2] Historie neuronových počítačů a sítí. In: Fakulta informatiky Masarykovy univerzity [online]. Brno [cit. 2016-05-18]. Dostupné z: http://www.fi.muni.cz/usr/jkucera/pv109/2000/xneudert.html

[3] ŠÍMA, Jiří a Roman NERUDA. Teoretické otázky neuronových sítí. Praha: MATFYZPRESS, 1996. ISBN 80-85863-18-9.

[4] BLAHA, Milan. Neuronové sítě-jednotlivý neuron [online]. Brno [cit. 2016-05-18]. Dostupné z: http://portal.matematickabiologie.cz/res/f/neuronove-site-jednotlivy-neuron.pdf

[5] AGATONOVIC-KUSTRIN, S a R BERESFORD. Basic concepts of artificial neural network (ANN) modeling and its application in pharmaceutical research. Journal of Pharmaceutical and Biomedical Analysis [online]. 2000, 22(5), 717-727 [cit. 2016-05-18]. DOI: 10.1016/S0731-7085(99)00272-1. ISSN 07317085. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S0731708599002721

[6] HAMMER, Miloš. Metody umělé inteligence v diagnostice elektrických strojů. Praha: BEN - technická literatura, 2009. ISBN 978-80-7300-231-2.

[7] JAIN, A.K., JIANCHANG MAO a K.M. MOHIUDDIN. Artificial neural networks: a tutorial [online]. [cit. 2016-05-19]. DOI: 10.1109/2.485891. ISBN 10.1109/2.485891. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=485891

[8] KVASNIČKA, Vladimír. Úvod do teórie neurónových sietí. Slovenská republika: IRIS, 1997. ISBN 80-887-7830-1.

[9] KARSOLIYA, Saurabh. Approximating Number of Hidden layer neurons in Multiple Hidden Layer BPNN Architecture. International of Engineering Trends and Technology [online]. 2012, 3(6) [cit. 2016-05-20]. ISSN 2231 - 5381. Dostupné z: http://www.ijettjournal.com/volume-3/issue-6/IJETT-V3I6P206.pdf

[10] VONDRÁK, Ivo. Umělá inteligence a neuronové sítě. 3. vyd. Ostrava: VŠB - Technická univerzita Ostrava, 2009. ISBN 978-80-248-1981-5.

[11] ŠNOREK, Miroslav. Neuronové sítě a neuropočítače. Praha: Vydavatelství ČVUT, 2002. ISBN 80-010-2549-7

[12] BLAHA, Milan. Neuronové sítě-jednotlivý neuron [online]. Brno [cit. 2016-05-18]. Dostupné z: http://portal.matematickabiologie.cz/res/f/neuronove-site-soutezive-site.pdf

[13] FISHER, R. A. THE USE OF MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS. Annals of Eugenics [online]. 1936, 7(2), 179-188 [cit. 2016-05-23]. DOI: 10.1111/j.1469-1809.1936.tb02137.x. ISSN 20501420. Dostupné z: http://doi.wiley.com/10.1111/j.1469-1809.1936.tb02137.x

[14] Iris Data set. UCI Machine learning depository [online]. [cit. 2016-05-23]. Dostupné z: https://archive.ics.uci.edu/ml/datasets/Iris

[15] Matlab Documentation. Mathworks [online]. Mathworks, Inc, 2016 [cit. 2016-05-24]. Dostupné z: http://www.mathworks.com/help/matlab/

[16] Iris dataset. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2016-05-26]. Dostupné z: https://en.wikipedia.org/wiki/Iris_flower_data_set#/media/File:Iris_dataset_scatterplot.svg

8 SEZNAM ZKRATEK A SYMBOLŮ

ξ - vnitřní potenciál neuronu

b - bias neuronu

$f(\xi)$ - aktivační funkce

y - vektor výstupů sítě

x - vstupní vektor sítě

w - matice váhových koeficientů sítě

w_{ij} - hodnota váhy spojení mezi neuronem i a j

$j_{\leftarrow}$ výstupy neuronů z předchozí vrstvy - vstupy neuronu j

T_s- trénovací množina pro učení s učitelem

T_u- trénovací množina pro učení bez učitele

I_i - vektor vstupů trénovací množiny pro učení s učitelem

λ - parametr strmosti sigmoidální funkce

O_i - vektor požadovaných výstupních hodnot pro učení s učitelem

$E(w)$ - celková chyba sítě typu backpropagation

$E_k(w)$ - parciální chyba sítě typu backpropagation

ε - koeficient učení pro backpropagation

$\frac{\partial E}{\partial w_{ij}}$ - gradient chybové funkce algoritmu backpropagation

$E_e(w)$ - energetická funkce Hopfieldovy sítě

s - poloměr okolí neuronu

s(t) - množina neuronů náležejících okolí s

$Gs(t)$ - množina neuronů náležející okolí vítězného neuronu

θ - koeficient učení u Kohonenových samoorganizačních map

α - koeficient posunutí při vektorové kvantizaci učením

ω - konstantní parametr pro 3. typ vektorové kvantizace učením

9 SEZNAM PŘÍLOH

[1] elektronická příloha