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

APLIKACE NEURONOVÝCH SÍTÍ
NEURAL NETWORKS APPLICATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Miloslav Macůrek**
AUTHOR

VEDOUCÍ PRÁCE — **prof. RNDr. Ing. Jiří Šťastný, CSc.**
SUPERVISOR

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

Ústav:	Ústav automatizace a informatiky
Student:	**Miloslav Macůrek**
Studijní program:	Strojírenství
Studijní obor:	Základy strojního inženýrství
Vedoucí práce:	**prof. RNDr. Ing. Jiří Šťastný, CSc.**
Akademický rok:	2016/17

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:

Aplikace neuronových sítí

Stručná charakteristika problematiky úkolu:

Umělé neuronové sítě patří k vhodným metodám pro aplikaci na problémy, které jsou jen obtížně řešitelné klasickými deterministickými matematickými metodami, nebo tam, kde aplikace exaktních výpočetních metod vyžaduje nepřiměřené zjednodušení úlohy.

Cíle bakalářské práce:

1. Charakteristika umělé neuronové sítě.
2. Uveďte základní typy neuronových sítí a příslušných učících algoritmů.
3. Proveďte analýzu neuronových sítí pro různé možnosti jejich aplikačního nasazení (klasifikace, predikce, shlukování).
4. Uveďte příklad řešení pomocí zvolené neuronové sítě.

Seznam literatury:

NORGAARD, M. Neural Networks for Modelling and Control of Dynamic Systems. Springer, London, 2000.

MANDIC, D. P. Recurrent Neural Networks for Prediction, Learning Algorithms, Architectures and Stability. Wiley, Chichester, 2001.

ŠNOREK M., JIŘINA M. Neuronové sítě a neuropočítače. ČVUT, Praha, 1996.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2016/17

V Brně, dne

L. S.

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

ABSTRAKT

Bakalářská práce se zabývá problematikou neuronových sítí, a to konkrétně jejich historií, charakteristikou, základními typy neuronových sítí a učících algoritmů a analýzou jejich aplikačního nasazení, demonstrované na jednoduchých příkladech v softwaru MATLAB.

ABSTRACT

This Bechelor's thesis will cover the theme of neural networks, their history, characteristics, basic typology of neural networks and learning algorithms and analysis of their applications demonstrated in simple examples created in MATLAB software.

KLÍČOVÁ SLOVA

Neuronové sítě, učící algoritmy, aplikace neuronových sítí, dopředná síť, Hopfieldova síť, Kohonenovy mapy.

KEYWORDS

Neural networks, learning algorithms, neural networks application, feedforwar network, Hopfield network, Kohonen maps.

BIBLIOGRAFICKÁ CITACE

MACŮREK, Miloslav. *Aplikace neuronových sítí.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2017. 66 s. Vedoucí bakalářské práce prof. RNDr. Ing. Jiří Šťastný, CSc.

PODĚKOVÁNÍ

Děkuji prof. RNDr. Ing. Jiřímu Šťastnému, CSc. za poskytnutou odbornou pomoc a vedení.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato práce je mým původním dílem, zpracoval jsem ji samostatně pod vedením prof. RNDr. Ing. Jiřího Šťastného, CSc. a s použitím literatury uvedené v seznamu literatury.

V Brně dne 15. 5. 2017 ……

Miloslav Macůrek

OBSAH

1 ÚVOD

Neuronové sítě jsou relativně nový výpočetní nástroj, který našel využití v řešení mnoha komplexních praktických problémů. Jejich výhodou je výborné zpracování informací, a to především i u nelineárních problémů, vysoká tolerance chyb a šumu zpracovávaných dat a také schopnosti učení a zobecnění. Princip fungování je založen na biologických sítích neuronů, které tvoří nervové sítě živých organismů, nicméně cílem není přesná replikace jejich chování, ale využití poznatků o jejich fungování pro řešení praktických problémů.

Využitelnost neuronových sítí spočívá ve schopnosti vytvořit funkci z obecného měření, nebo pozorování. To je především užitečné v případech, kdy komplexnost dat, nebo úkolu činí tvorbu této funkce jiným způsobem zdlouhavou a nepraktickou. Nejvíce využívané jsou neuronové sítě pro řešení problémů aproximace funkce, klasifikace, predikce, shlukování, nebo obecného zpracování dat, které nachází využití například u *image processing*, či *face identification*. Další oblasti aplikací zahrnují *system identification*, *decision making*, nebo robotiku a řízení.

Cílem této práce je shrnout teoretické poznatky o neuronových sítích, popsat 3 nejvyužívanější z hlediska jejich stavby a základní matematiky a vytvořit jednoduché příklady jejich aplikací v oblastech modelování, predikce, klasifikace a shlukování za využití předprogramovaných neuronových sítí v softwaru Matlab.

2 HISTORIE

2.1 Počátky

Neuronové sítě jako koncept si berou vzor v neuronových soustavách živých organismů, jako je například lidský mozek. Tyto soustavy jsou studovány již tisíce let a tyto poznatky jsou užitečné pro tvorbu umělých systémů pracujících na stejném principu, čehož se využívá v umělé inteligenci.

První práce věnující se problematice neuronových sítí, *A Logical Calculus of the Ideas Immanent in Nervous Activity* [2], vznikla v roce 1943 a autory jsou Warren McCulloch a Walter Pitts. Jejich sítě byly založeny na jednoduchých prvcích, které byly považovány za binární zařízení s pevnými prahovými hodnotami. Výsledkem jejich modelu byly jednoduché logické funkce.

Donald Hebb v roce 1949 pokračoval ve výzkumu tématu a své poznatky sepsal v knize *The Organizations of Behavior* [3], kde mimo jiné poukázal na zesilování neuronových spojení pokaždé, když jsou použity, což je základem pro učení, a také se zde poprvé objevuje učící pravidlo pro synapse neuronů.

Na tomto základě byl vybudován první neuropočítač, Snark, který nicméně nenašel praktického využití. Jeho autorem byl Marvin Minsky a postaven byl v roce 1951. [4]

2.2 Slibné pokroky

V roce 1957 byl model neuronu zobecněn Frankem Rosenblattem, tento zobecněný neuron byl nazván perceptron a počítal s reálnými čísly. Jednalo se o pevnou jednovrstvou architekturu s n vstupními a m výstupními neurony. Zároveň navrhl nový učící algoritmus, který byl schopen v reálném čase nalézt váhový vektor parametrů, a to nezávisle na úvodní konfiguraci. Mimo to sepsal knihu na téma neurovýpočtů, *Principles of Neurodynamics* [5].

Dalším systémem by ADALINE (ADAptive LInear Element) vyvinutý v roce 1960 inženýry Widrow a Hoffem. Metody učení byly jiné než u perceptronu, využívaly takzvané Least-Mean-Squares učící pravidlo. V roce 1962 potom Widrow a Hoff vytvořili učící postup, který umožnil přezkoumat hodnoty předtím, než je váhy upraví. [4]

2.3 Kritika

I přes rychlý vývoj neuronových sítí se našli odpůrci, kteří tento systém kritizovali. Ani to však nakonec vývoj nezastavilo.

Roku 1969 pánové Minsky a Papert sepsali knihu *Perceptrons: A Introduction to Computational Geometry*. Tato byla součástí kampaně pro zdiskreditování vývoje neuronových sítí a poukazovala na několik zásadních problémů s nimi. Mimo jiné v ní generalizovali limity jednovrstvého perceptronu. Přestože autoři si byli vědomi, že pokročilejší perceptrony mají více vrstev a Rosenblattovy jednoduché dopředné perceptrony mají vrstvy 3, definovali perceptron jako dvouvrstvý systém, schopný řešení pouze lineárně separovatelných problémů a nedokáže si poradit s například s problémy, ve kterých se vyskytuje výhradně funkce OR.

Zájem o neuronové sítě tedy upadal a spolu s minimálními dotacemi na výzkum to zapříčinilo nízký počet vědců pokračujících v práci na problémech typu *pattern recognition*. A přesto v této době vzniklo několik paradigmat, které jsou v dnešní době nadále rozvíjeny.

Klopf v roce 1972 ve své práci *Brain function and adaptive systems – a heterostatic theory* [6] položil základy pro učení umělých neuronů na biologickém principu. Roku 1974 Paul Werbos vytvořil metodu zpětného šíření chyby pro dopřednou síť - učící algoritmus, který však nebyl plně doceněn až do roku 1986.

Fukushima na základě své práce z roku 1975 *Cognitron: A selforganizing multilayered neural network* sestavil krokově učenou vícevrstvou neuronovou síť pro přepis ručně psaných textů do strojové podoby.

V roce 1976 Grossberg v práci *Adaptive pattern classification and universal recoding* adaptivní rezonanci jako teorii vědomého zpracování informací lidmi. [4]

2.4 Inovace

V 80. letech 20. století několik událostí způsobilo obnovený zájem o problematiku. Kohonen přispěl mnoha poznatky z oblasti a vytvořil umělou neuronovou síť známou jako Kohonenova mapa, nebo Kohonenova síť.

Hopfield z Caltechu v roce 1982 představil svou práci *Neural Networks and Physical Systems with Emergent Collective Computational Abilities*, kde popsal rekurentní neuronovou síť sloužící jako adresovatelný paměťový systém pro obsah. Jeho práce přesvědčila mnoho kvalifikovaných vědců, matematiků a inženýrů k návratu k problematice neuronových sítí.

Metoda zpětného šíření chyby, původně objevená roku 1974 Werbosem, byla znovuobjevena roku 1986 v knize *Learning Internal Representation by Error Propagation* od Rumelharta a kolektivu. Tato metoda se používá k minimalizování chyby při výpočtu neuronovou sítí.

Roku 1985 Americký Institut Fyziky (*American Institute of Physics)* začal, co se později stalo každoročním setkáním - *Neural Networks for Computing*. V roce 1987 proběhla v San Diegu první volně přístupná konference o neuronových sítích v moderní historii - *IEEE International Conference on Neural Networks* a *International Neural Net-*

work Society (INNS) byla založena. Roku 1988 vznikl INNS deník *Neural Networks*, následován dalšími jako *Neural Computation* roku 1989 a *IEEE Transactions on Neural Networks* v roce 1990.

Roku 1987 Carpenter a Grossberg v práci *Adaptive pattern classification and universal recoding* popsali ART1 – neřízený učící model speciálně navržený pro rozpoznávání binárních vzorů. [4]

2.5 Současnost

V oblasti neuronových sítí bylo dosaženo významných pokroků, dostatek na přilákání velké pozornosti a získání větších finančních prostředků pro výzkum a vývoj. Neuronové sítě jsou aplikovány v mnoha oblastech, kde by konvenční výpočtové a programovací metodu neuspěly. Jsou vyvíjeny neuronové čipy a vznikají aplikace na řešení komplexních problémů. Neuronové sítě jsou jednoznačně hitem současnosti.

Mezi lety 2009 a 2012 byly výzkumnou skupinou okolo pana Schmidhubera vyvíjeny rekurentní a hluboké dopředné neuronové sítě popsané v práci *Deep learning in neural networks: an overview.*

V roce 2014 vědci z IBM představili procesor TrueNorth s architekturou podobnou lidskému mozku. IBM prezentovala čip velikosti poštovní známky schopný simulovat funkci milionů neuronů s 256 miliony spojení v reálném čase. Systém je schopen provádět 46 až 400 miliard synaptických operací za sekundu.

V roce 2015 ve své disertační práci Matthew Lai představil šachový program Giraffem založený na neuronových sítích a učících algoritmech. Jeho cílem bylo prezentovat možnosti odlišného přístupu k řešení, než je standartní brute-force calculation. Klasické šachové enginy (nejsilnější jsou v dnešní době Stockfish, Houdini a Komodo) pouze procházejí více či méně ořezaný strom možných budoucích tahů, hodnotí pozici dle daných nastavených parametrů (kůň v centru šachovnice má hodnotu 3,02, protože ovládá mnoho důležitých polí, kdežto v rohu má pouze hodnotu 2,96 atp.) a rozhodují o budoucím tahu na základě funkce min-max. Díky velké výpočetní síle dnešních procesorů jsou schopny dosáhnout velmi vysoké hloubky propočtu, se kterou se zlepšuje jejich hodnocení pozice. Giraffe měl umožnit enginu se hru v podstatě učit stylem, jakým se učí i lidé. Klasické brute-force calculation enginy mají však za sebou dlouhý vývoj a dosáhly velmi vysoké úrovně (žádný člověk jim dnes nemůže konkurovat, ani World Chess Champion GM Magnus Carlsen) a Giraffe se tak nikdy nedostal na úplnou špičku. [7]

Matthew Lai, který se stal součástí teamu společnosti Google DeepMind tedy projekt ukončil. Další aplikace neuronových sítí byla testována na japonské hře Go, kde s programem AlphaGo v roce 2016 dokázali porazit nejsilnějšího hráče, Lee Sedola. Rozdíl oproti šachu u hry Go je v počtu možných kombinací, kterých je v Go mnohem více, a proto klasické brute-force calculation programy neměly velký úspěch. [8]

V roce 2017 oznámila společnost Intel nový procesor Lake Crest, který je cílen přímo pro deep learning a přinést by měl až stonásobný výkon oproti současným řešením, které jsou povětšinou realizována pomocí grafických čipů. [9]

Pro mistrovství světa v ragby 2019 a olympijské hry 2020 v Japonsku plánuje společnost Toshiba nasadit systém analýzy obrazu založený na neuronových sítích, který by měl v reálném čase vyhodnocovat dění na hřišti pouze z obrazu klasických kamer používaných pro televizní vysílání a pomoci rozhodčím posuzovat například hraniční zákroky hráčů, nebo určit, zda se míč dostal celým objemem za brankovou čáru ve sporných situacích. [4][10]

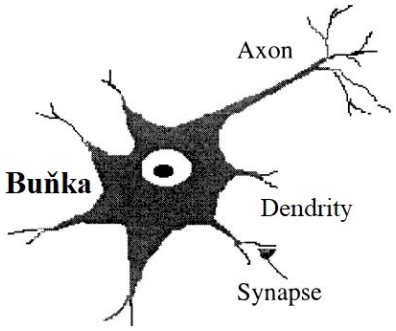
3 CHARAKTERISTIKA

Neuronové sítě jsou výpočetní nástroj pro modelování a řešení komplexních praktických problémů. Mohou být charakterizovány jako struktury, složené z husté sítě propojených jednoduchých výpočetních jednotek, nazývaných umělé neurony, které jsou schopny mnoha paralelních výpočtů pro zpracování dat. Neuronové sítě jsou pouze velmi abstraktním napodobením jejich biologických protějšků, nicméně cílem není přesné napodobení jejich funkce, nýbrž použití znalostí o nich k řešení praktických problémů. Atraktivita biologických neuronových sítí spočívá především v jejich výborných charakteristikách zpracování informací jako je nelinearita, paralelita, robustnost, tolerance chyb a selhání, schopnost učení, zpracování nepřesných a zašuměných informací a schopnost zobecnění.

Umělé modely s těmito charakteristikami jsou žádoucí, protože nelinearita umožňuje lepší přizpůsobení se datům, tolerance šumu poskytuje přesnou předpověď vývoje i za přítomnosti nepřesných dat a chyb měření, paralelismus napomáhá rychlému zpracování dat, schopnost učení dává možnost modifikace vnitřní struktury systému pro přizpůsobení měnícímu se prostředí a zobecnění je využito při aplikaci modelu na nová data. Hlavním cílem neuronových sítí a neuropočítačů je vytvořit matematický algoritmus, který bude napodobovat zpracování informací a získání znalostí lidským mozkem. [1]

3.1 Biologický neuron

Lidský nervový systém se skládá z milionů neuronů (nervových buněk) různých typů a délek, podle toho, kde v těle se nacházejí. Neurony, jak je vidět v obr. 1, se skládají z těla nervové buňky, které obsahuje jádro řídící aktivitu buňky, mnoha dendritů, které přijímají signály od sousedních neuronů a předávají je buňce ke zpracování, a axonu, který se dělí do mnoha větví a předává informace od buňky dendritům ostatních neuronů přes synapse, mikroskopické ‚mezery'.

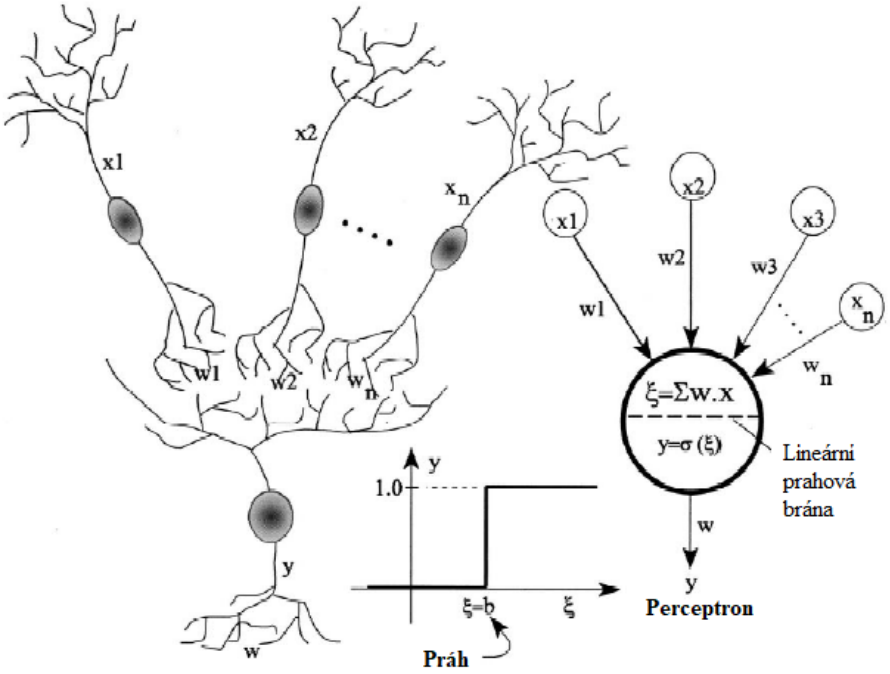


Obr. 1: Nervová buňka (převzato z [11])

Přenos signálu probíhá pomocí drobných elektrických impulsů velikosti desítek milivoltů. Impuls přichází z dendritů přes tělo buňky do před-synaptické membrány synapse. Po dosažení membrány, tato vypustí chemické neurotransmitery v množství odpovídajícím síle vstupního signálu. Tyto neurotransmitery přejdou do po-synaptické membrány, odkud pokračují do dendritů sousedních neuronů, kde generují nový elektrický impuls, který se dále šíří stejným postupem přes tento další neuron. Množství signálu, který projde přes neuron závisí na mnoha faktorech, především na síle přijímaného signálu, síle spojení s neurony, od kterých signál přišel, a prahu přijímajícího neuronu. Protože má neuron mnoho dendritů, je schopen přijímat a předávat mnoho signálů simultánně. Neuron vysílá signál na základě těchto příchozích signálů a to, když přesáhnou kritickou hodnotu elektrického napětí na membráně. Přijímané excitační signály podporují vyslání signálu neuronem a inhibiční naopak působí proti vyslání signálu. [1]

3.2 Analogie

Přestože umělé neuronové sítě nejsou přesnou replikou funkčnosti svých biologických protějšků, základní stavba je zde dodržena. Spojení mezi umělými neurony představují axony a dendrity, váha reprezentuje synapse a práh aproximuje aktivitu jádra neuronové buňky. Obr. 2 ukazuje *n* biologických neuronů (vlevo) s různými signály intenzity *x* a silou synapse (váhou) *w* vstupujícími do neuronu s prahem *b* a ekvivalentní umělý neuronový systém. Oba systémy jsou schopny učení, a to úpravou síly spojení u biologické a velikosti vah u umělé sítě. [1]



Obr. 2: Porovnání biologického a umělého neuronu (převzato z [1])

3.3 Umělý neuron

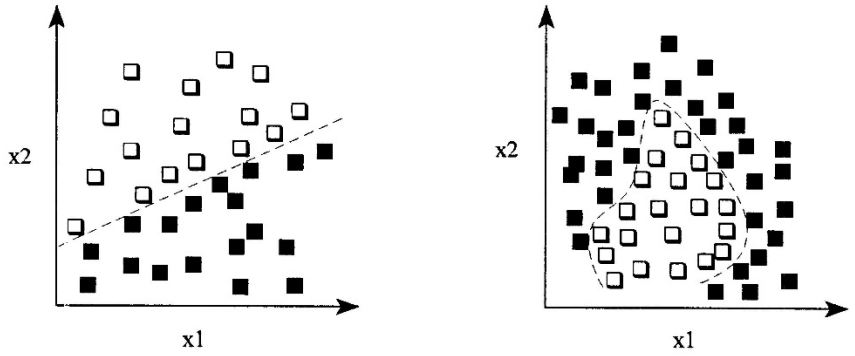
Umělý neuron přijímá vstupy jako podněty z prostředí, kombinuje je do jednotného ‚síťového' vstupu ξ (vnitřní potenciál), ten prochází přes lineární prahovou bránu a odchází jako výstup *y* do dalších neuronů, nebo do prostředí. Jak bylo ukázáno v obr. 2, pouze pokud ξ přesáhne prahovou hodnotu *b* neuronu, dochází k aktivaci. Běžně se pro výpočet ξ uvažuje lineární dynamika neuronu. Síťový vstup se počítá jako skalární součin vstupních signálů *x*, závisející na neuronech a jejich vahách *w*. Pro *n* signálů lze operace perceptronových neuronů vyjádřit dle vztahu (3.1).

$$y = \begin{cases} 1, \ if \ \sum_{i=1}^{n} w_i x_i \geq b \\ 0, \ if \ \sum_{i=1}^{n} w_i x_i < b \end{cases} \tag{3.1}$$

Ve vztahu (3.1) je aktivace neuronu znázorněna jako 1 a klidový stav jako 0. Kladné váhy ($w_i > 1$) zesilují síťový signál ξ a spojení je nazýváno excitační, zatímco záporné váhy tlumí aktivitu neuronu a nazývají se inhibiční. Systém skládající se z umělého neuronu a vstupů jako na obr. 2 se nazývá perceptron a vytváří mapování mezi aktivitou vstupů (podnětů) a výstupním signálem. V rovnici (3.1) může být práh neuronu považován jako vstup, jehož hodnota intenzity je vždy $x = 1$ a váha $w = b$. V tomto případě nabývá suma z rovnice (3.1) hodnot od 0 do *n* a ξ se porovnává s 0. [1]

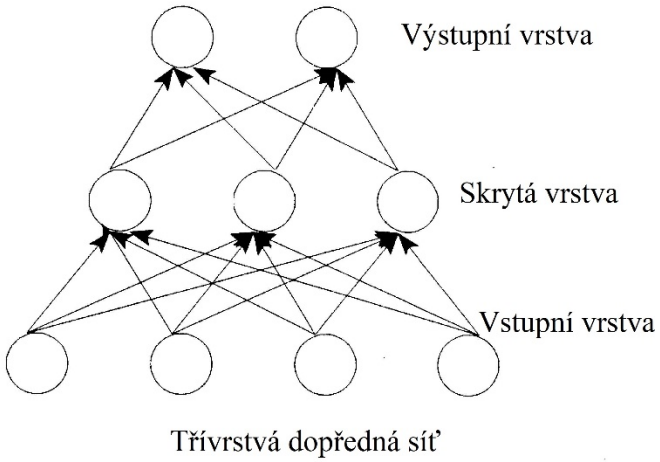
3.4 Perceptrony

Perceptron (obr. 2) může být učen na sadě příkladů pomocí speciálního učícího pravidla. Váhy perceptronu (včetně váhy prahu) jsou upravovány vzhledem k rozdílu požadovaného správného výstupu *Y* a výstupu řešení perceptronu *y* v každém příkladu. Chyba je funkce všech vah a formuje nepravidelnou multidimenzionální hyperrovinu s mnoha vrcholy, sedlovými body a minimy. Užitím specializovaných technik hledání, učící proces cílí na získání sady vah odpovídající globálnímu minimu. Rosenblatt v roce 1962 vytvořil perceptronové pravidlo, které najde optimální vektor vah v konečném počtu iterací, nezávisle na počátečním stavu, nicméně toto pravidlo pracuje přesně pouze s lineárně separovatelnými třídami objektů. Obr. 3 ukazuje lineárně a nelineárně separovatelné dvouobjektové klasifikační problémy.



Obr. 3: Lineárně a nelineárně dělitelné třídy objektů (převzato z [1])

Aby bylo možno pracovat i s nelineárně dělitelnými třídami, bylo nutno přidat mezivrstvu neuronů mezi vrstvu vstupní a výstupní, jak lze vidět na obr. 4, a vzniká tak vícevrstvá architektura (MLP z anglického *Multilayer Perceptron*). Protože tyto vrstvy neinteragují s prostředím, říká se jim skryté vrstvy. Jedním z příkladů vícevrstvé sítě je zpětná síť (Rumelhart, 1986) učená delta učícím pravidlem.



Obr. 4: Vícevrstvá neuronová síť (převzato z [1])

Učení MLP není tak přímé jako učení jednoduchého perceptronu, ale bylo nutno rozšířit učící algoritmus, aby byl schopen upravovat i váhy neuronů skryté vrstvy. [1]

3.5 Dělení

Neuronové sítě mohou být děleny podle mnoha kritérií v závislosti na jejich charakteristických vlastnostech. Dělit sítě lze podle jejich aplikací (modelování, shlukování), stupně propojení (plné, částečné), směru toku informací sítí (dopředná, zpětná), kdy zpětná je dynamickým systémem, kde stav v jakémkoliv čase je závislý na předchozích stavech,

typu učícího algoritmu, který představuje řadu rovnic, které využívají výstupy získané ze sítě spolu s libovolným výkonovým měřítkem pro aktualizaci vnitřní struktury neuronové sítě, dělit lze dále podle učícího pravidla, které pohání učící algoritmus, a také podle úrovně řízení učení (řízené, neřízené, hybridní). Řízené učení sítě je založeno na učící sadě, která obsahuje jak vstupy, tak požadované výstupy daných prvků, ze kterých se síť učí, zatímco pro učení neřízené učená síť nedostává požadované výstupy pro dané vstupy a dělí zadané hodnoty do shluků podle jejich podobnosti a rozdílnosti (př. samo-organizační mapy jako Kohonenova mapa). Hybridní řízení potom kombinuje řízené a neřízené učení dle potřeby využití sítě. [1]

3.6 Učící pravidla

Učící pravidla definují, jakým způsobem jsou váhy upravovány mezi jednotlivými učícími cykly (epochami). Jsou 4 základní pravidla.

The error-correction learning (ECL), tedy pravidlo opravy chyb, je využíváno v řízeném učení, kdy aritmetický rozdíl (chyba, *error*) mezi řešením neuronové sítě v průběhu učení a požadovanou hodnotou je použita na úpravu vah spojení s cílem postupného snížení celkové chyby sítě.

Boltzmanovo učící pravidlo je odvozeno z termodynamických principů a teorie informací a je velmi podobné ECL, ale každý neuron generuje výstup dle Boltzmanova statistického rozdělení, což velmi zpomaluje učení sítě.

Hebbovo pravidlo, založené na neurobiologických experimentech, je nejstarším učícím pravidlem. Pravidlo říká, že pokud jsou neurony na obou stranách spojení aktivovány zároveň a opakovaně, spojení je zesíleno. Tedy na rozdíl od ECL a Boltzmanova pravidla, úprava vah spojení je prováděna lokálně na základě aktivity spojených neuronů.

Soutěživé učení (*competitive learning*) nechává všechny neurony soutěžit mezi sebou tak, že pouze jeden neuron v každé iteraci učení je aktivován a váhy všech jeho spojení s ostatními neurony jsou v dané iteraci upraveny. [1]

3.7 Přeučení sítě

Neuronové sítě s malým počtem neuronů mají horší schopnost popsat závislosti v trénovacích vzorcích dat, nicméně její schopnost generalizace, tedy nalezení obecného vzoru a jeho aplikace na nová data, může být vyšší než u sítě s příliš velkým počtem neuronů. Sítě s velkým počtem neuronů naopak nemají problém s přizpůsobením se učeným datům, ale jejich schopnost generalizace klesá – tento jev se nazývá přeučení sítě (*overfitting*). K přeučení také může dojít v případě velkého množství vstupních parametrů a malého počtu trénovacích vzorků. Cílem využití neuronové sítě není maximalizace výkonu na zadaných datech, ale rozumný kompromis mezi ziskem přesnosti v daném vzorku a schopnosti generalizace pro nová neznámá data.

Proto vzorek dat pro učení sítě dělíme na tři části – trénovací, testovací a validační množinu. Trénovací množina je náhodně vybraná část dat, na které se síť učí, testovací množina je část dat sloužící k zastavení učení, aby nedošlo k přeučení sítě a validační množina je zbytek dat, která sít dosud neměla k dispozici, a na kterých se ověřuje konečná kvalita modelu.

Vhodné dělení dat je většinou v poměru 70-15-15. Po skončení učení jsou vyhodnoceny výsledky a nejlepší model je ten, který nemá příliš velké rozdíly ve výkonech na jednotlivých množinách. [12]

4 NEJVYUŽÍVANĚJŠÍ NEURONOVÉ SÍTĚ

Neuronových sítí je mnoho typů, které plní rozdílné funkce. V této práci jsou zmíněny jen ty základní, které jsou nejvíce rozšířené a využívané, a v této části jsou podrobněji popsány 3 z nich – dopředná, Hopfieldova a Kohonenova.

4.1 Dopředná síť

Dopředná síť se zpětným šířením chyby (*feedforward backpropagation network*) je nejčastěji využívanou neuronovou sítí, a to především díky její univerzálnosti a flexibilitě. Nejčastěji využívanými sítěmi jsou sítě dvouvrstvé (jedna skrytá a jedna výstupní vrstva), protože bylo dokázáno, že přidání více skrytých vrstev již nemá vliv na schopnost řešit složitější problémy. Namísto počtu skrytých vrstev má tedy hlavní vliv na výkon sítě počet neuronů skryté vrstvy. Méně využívané jsou potom jednovrstvé perceptrony a třívrstvé sítě. [1]

Dopředná síť je učena řízeně (s učitelem). To znamená, že pro její učení je třeba sada vstupních dat s jejich požadovaným výstupem a v určitém počtu iterací učící algoritmus upravuje váhy spojení neuronů tak, aby bylo dosaženo co nejpřesnějšího výstupu sítě.

Pro učení vícevrstvých sítí je třeba použít zobecněné *delta rule* pro perceptron, které počítá chyby i pro neurony ve skryté vrstvě. Nejprve je třeba určit aktuální chybu sítě $E = f(x, d)$, kdy x je hodnota vstupu, d je požadovaný výstup (*target*) a y je reálný výstup. Chyba E je vyjádřena rovnicemi (4.1) a (4.2).

$$E(w) = \sum_{k=1}^{p} E_k(w) \tag{4.1}$$

$$E_k(w) = \frac{1}{2} \sum_{j \in Y} (y_j - d_j)^2 \tag{4.2}$$

Aktuální konfiguraci vah značí w a p je počet vzorů. Celková chyba sítě je sumou chyb jednotlivých vzorů. Y je množina neuronů ve výstupní vrstvě.

Minimalizace chyby a adaptace jednotlivých vah se realizuje zobecněným pravidlem pro perceptrony (4.3).

$$w^t = w^{(t-1)} - \Delta w^t \tag{4.3}$$

w^t označuje novou váhu spojení a $w^{(t-1)}$ váhu původní. Δw^t je změna váhy počítaná dle vztahu (4.4).

$$\Delta w^t = \frac{\delta E(w)}{\delta w} \tag{4.4}$$

Díky vysoké univerzálnosti jsou dopředné sítě využívány pro mnoho aplikací, jako jsou klasifikace, predikce, řízení a aproximace matematických funkcí (modelování). [12]

4.2 Hopfieldova síť

Hopfieldova neuronová síť patří mezi zpětné (*feedback network*). Skládá se z *n* neuronů a každý z nich je zároveň vstupem a výstupem. Výstupní signál neuronů je zároveň vstupem pro ostatní neurony sítě – síť je plně propojená, ale bez vazeb neuronů na sebe sama.

Učení je jednorázové a síť se učí sadu vzorů jako stabilní stavy sítě. Při předložení nového vzoru si síť vybavuje nejpodobnější naučený vzor zkonvergováním do nejbližšího stabilního stavu. Váha mezi neurony představuje pravděpodobnost, že spojené neurony budou současně zaktivovány při hledání správného řešení. Pokud se tedy aktivita obou neuronů vylučuje, vazba mezi nimi bude silně inhibiční.

Učení se řídí Hebbovým zákonem. Tréninková sada pro učení obsahuje vstupní data a jejich požadované výstupy a každý vzor je učen pouze jednou. Výsledné váhy jsou potom určeny pomocí (4.5).

$$w_{ji} = \sum_{k=1}^{p} x_{kj} \cdot x_{ki} \qquad \wedge \qquad 1 \leq j \neq i \leq n \tag{4.5}$$

Počet vzorů je *p* a w_{ji} je váha mezi neurony *i* a *j*. x_{kj} a x_{ki} jsou *i*-tý a *j*-tý neuron v *k*-tém vzoru.

Při řešení neznámých vzorů se stavy neuronů nastaví na hodnotu vstupů. Následně probíhá výpočet vnitřních potenciálů jednotlivých neuronů dle (4.6), kde y_i je výstupem neuronu *i*. Potom je určen nový výstup y_j podle (4.7).

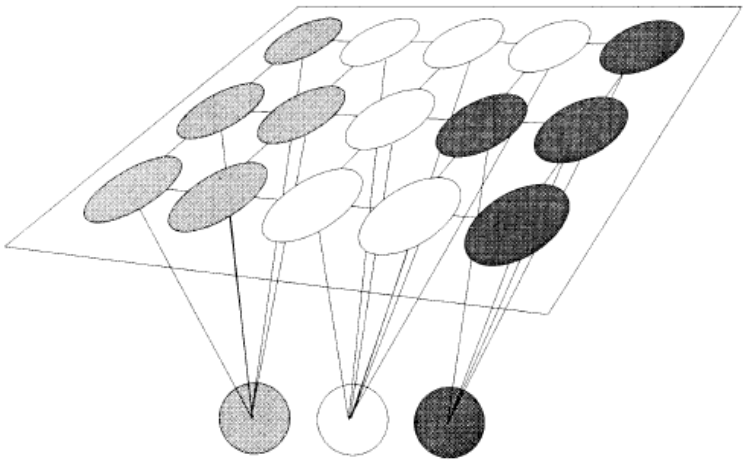
$$\xi_j^{(t-1)} = \sum_{i=1}^{n} w_{ji} \cdot y_i^{(t-1)} \tag{4.6}$$

$$y_j^t = \begin{cases} 1, & \xi_j^{(t-1)} > 0 \\ y_j^{(t-1)} & \xi_j^{(t-1)} = 0 \\ -1, & \xi_j^{(t-1)} < 0 \end{cases} \tag{4.7}$$

Výsledné stavy jsou tedy bipolární – nabývají pouze hodnot 1 a -1, což je velmi podobné biologickým neuronům, které nabývají pouze stavů aktivace a klidu. Tento výpočet se opakuje, doku se stavy neuronů mění. [12]

4.3 Kohonenova síť

Samo-organizační sítě nebo mapy (*self-organizing map,* SOM) jsou založeny na soutěživém neřízeném učení, tedy jejich učení probíhá pouze na množině vstupních dat bez předpokládaných výstupů. Neurony mezi sebou soutěží o to, který z nich bude v dané iteraci aktivován. Síť se tedy sama přizpůsobuje vstupním datům a předpokládané výstupy nejsou třeba. Kohonenovy mapy jsou určeny pro rozhodování, třídění a rozlišování dat.



Obr. 5: Struktura Kohonenovy mapy (převzato z [11])

Kohonenovy samo-organizační mapy vycházejí z Kohonenova učení. Jedná se o síť jednovrstvou s úplným propojením vstupní a kompetiční vrstvy, jak lze vidět na obr. 5. Každý neuron má tedy informaci o všech vstupech. Neurony v kompetiční vrstvě jsou uspořádány do 2D obdélníkového, nebo hexagonálního tvaru a váhy jednotlivých neuronů jsou definovány pomocí jejich polohy v něm. Vzdálenost d_j vstupního vektoru od vzoru uloženého ve vahách neuronu je potom funkcí každého neuronu (4.8), kdy N je počet vstupů.

$$d_j = \sum_{i=1}^{N-1} (x_i(t) - w_{ij}(t))^2 \tag{4.8}$$

Učení sítě spočívá v uspořádání neuronů do oblastí klasifikovaných vstupními daty. Váhy jsou iterativně adaptovány a dochází k porovnávání vstupních vzorů a vektorů, uložených v neuronech. Když je nalezen neuron, který nejlépe odpovídá vzoru, jsou jeho váhy upraveny. Výběr nejpodobnějšího neuronu j^* se provádí podle podmínky (4.9).

$$d_{j^*} = \min_j (d_j) \tag{4.9}$$

Úprava vah nejpodobnějšího neuronu a jeho okolí se provádí vztahem (4.10).

$$W_{ij}(t+1) = W_{ij}(t) + \alpha(t)\left[x_i(t) - w_{ij}(t)\right] \tag{4.10}$$

α je parametrem učení, nastaveným před začátkem učení na hodnotu blížící se 1. Učení sítě lze rozdělit na dvě fáze. Tou první je hrubé učení, kdy se váhy výrazně mění a dochází k rozprostření sítě po ploše potřebné pro obsažení zadaných vstupních dat. Ve druhé fázi dochází k menším změnám vah a síť je jemně kalibrována.

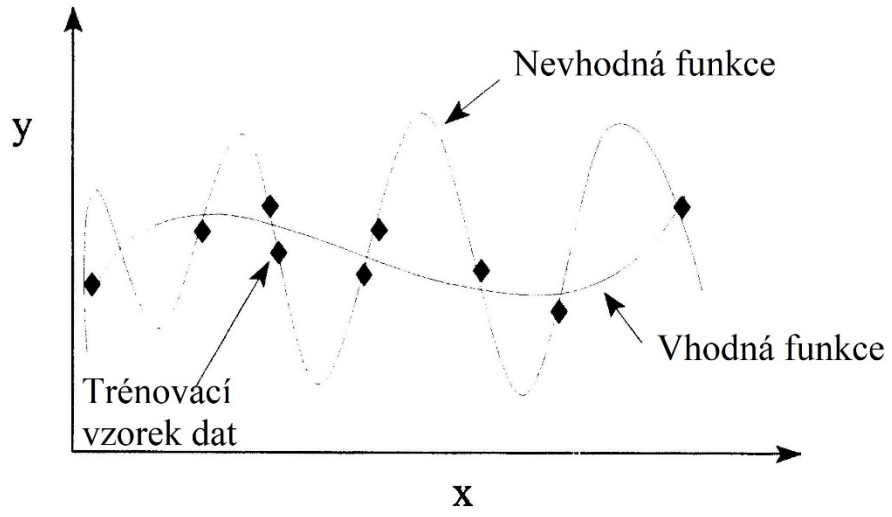
Po naučení sítě následuje její využití, kdy je síti předložen neznámý vzor, který je sítí zařazen do odpovídající kategorie. Tímto však nemusí proces končit – síť se může nadále učit s každým novým vzorem (tzv. adaptační režim) a potom by byly váhy vítězného neuronu opět upraveny, nebo se váhy neupravují (neadaptační režim). [12]

5 APLIKACE

Obecně se dá říci, že neuronové sítě jsou velmi silné a často lepší než ostatní dostupné nástroje v řešení problémů, které spadají do následujících kategorií.

5.1 Modelování

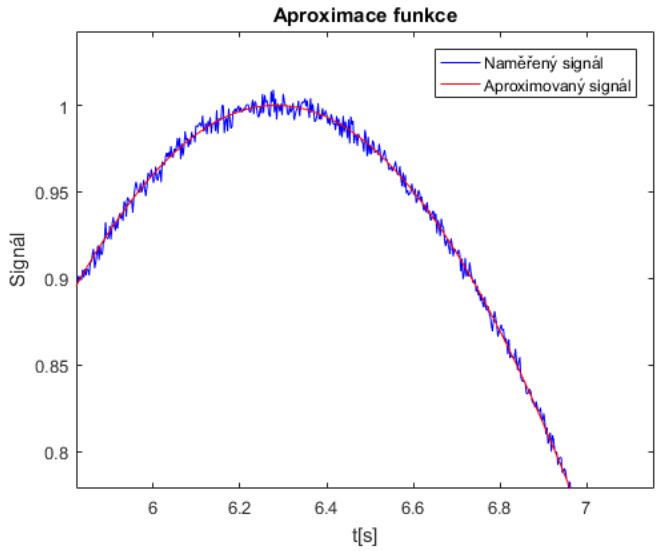
Modelování, nebo aproximace funkce, zahrnuje trénování neuronové sítě na sadě dat, která obsahuje hodnoty jak vstupů, tak výstupů, s cílem aproximovat základní pravidla, která vstupy a výstupy spojují. Vícevrstvé neuronové sítě jsou považovány za univerzální v této oblasti a dovedou aproximovanou jakoukoliv funkci s jakoukoliv požadovanou přesností, a proto jsou ve většině případů pro tento účel používány právě ony. Aproximace funkce je využíváno pro řešení problémů, kde není dostupný žádný teoretický model, jako například pro zpracování naměřených dat z experimentů, nebo jako náhrada teoretických modelů, které jsou náročné analyticky počítat. Modelování znázorňuje obr. 6. [1]



Obr. 6: Modelování (převzato z [1])

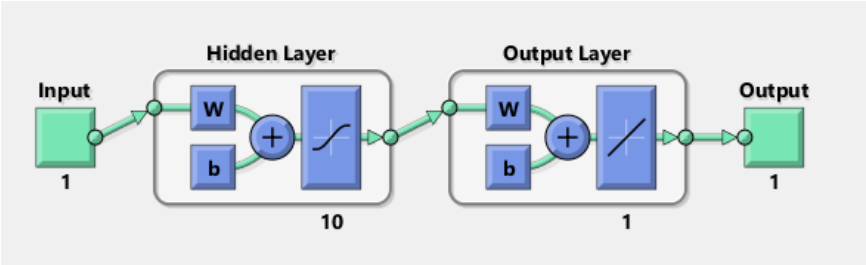
5.1.1 Praktický příklad modelování

Pro demonstraci modelování byla použita funkce $y = \cos(x)$ v intervalu $< 0, 4\pi >$, který byl rovnoměrně rozdělen na 5000 vstupních hodnot. Tato funkce byla následně modifikována náhodnou chybou $\pm 1\,\%$ pro simulaci naměřených hodnot experimentu, jak lze vidět z obrázku 6. Stejnou chybu mají například běžné multimetry pro měření v elektrotechnice. Simulace neuronové sítě byla provedena v Matlabu za použití funkce *Neural Fitting* pod integrovaným *Neural Network Toolbox*.



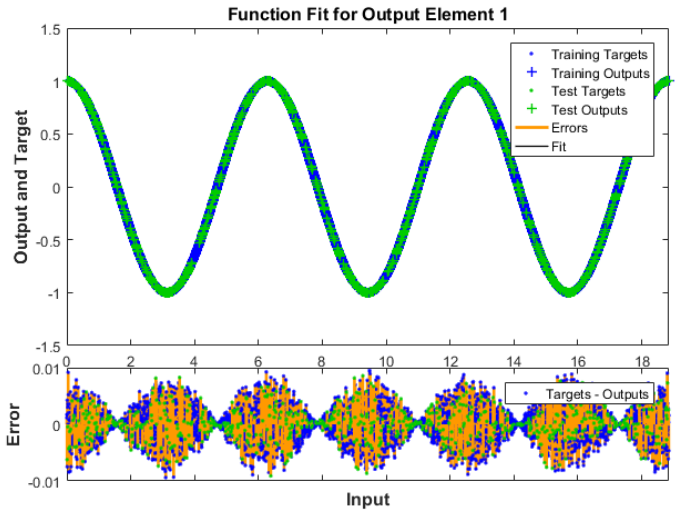
Obr. 7: Simulovaný naměřený signál a jeho aproximace

Použita byla dvouvrstvá dopředná neuronová síť s jedním vstupem, jedním výstupem a deseti skrytými vrstvami neuronů, učení bylo realizováno pravidlem *Bayesian Regularization*. Schéma sítě zobrazuje obr. 8.



Obr. 8: Schéma neuronové sítě pro aproximaci

V obr. 7 a 9 lze vidět výsledek aproximace. Ta byla velmi dobrá a pro daný interval, pro který byla aproximace provedena bylo dosaženo střední kvadratické chyby při učení $8{,}074 \cdot 10^{-6}$.



Obr. 9: Výstup aproximace při učení neuronové sítě

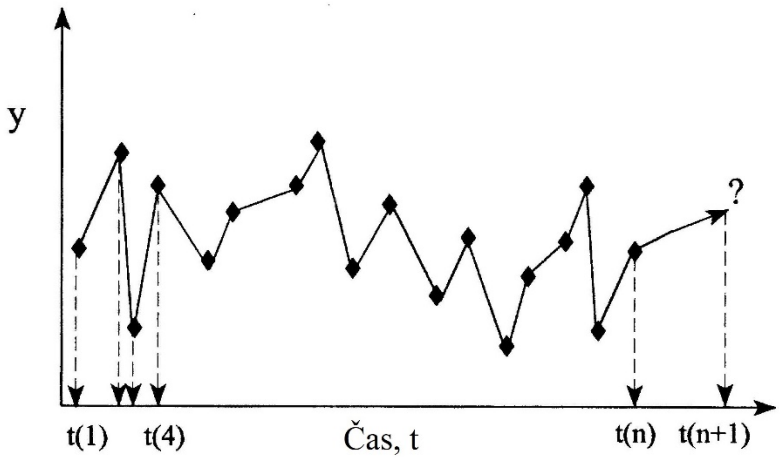
Z obr. 9 je zřejmé, že model je nejméně přesný v amplitudách aproximované funkce cosinus, kde chyba dosahuje tisícin až jedné setiny.

Pro přesnější modelování by bylo možné rozšířit neuronovou síť o další skryté neurony a neuronové vrstvy. Tvorba modelu by byla časově náročnější, ale bylo by možno dosáhnout lepších výsledků.

V příloze 1 je naučenou sítí vygenerovaná funkce *fitdata* a skript, který generoval náhodná data pro učení této sítě a zároveň vykresloval výstupy ve formě grafů.

5.2 Predikce (Time Series)

Predikce využívá trénování sítě na sadě dat z určitého časového rozmezí, která reprezentují určitý fenomén, tak jak je znázorněno v obr. 10, a následné využití naučené sítě pro předpověď dalšího vývoje. To znamená, že síť bude predikovat výsledek funkce *Y* v čase (*t+1*) na základě jednoho, nebo více předchozích pozorování hodnot funkce v časech (*t-2*), (*t-1*) a *t*. [1]



Obr. 10: Predikce (převzato z [1])

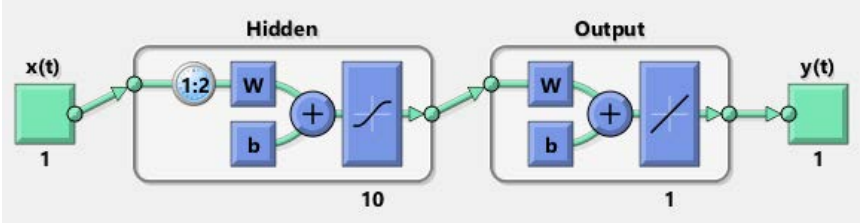
5.2.1 Praktický příklad predikce

Pro praktickou prezentaci predikce byla zvolena opět funkce $y = \cos(x)$ a cílem bylo naučit neuronovou síť pravidlem *Bayesian Regularization* na intervalu $< 0, 2\pi >$, následně exportovanou funkcí predikovat vývoj na intervalu $< 2\pi, 4\pi >$ a porovnat s původní funkcí. Simulace neuronové sítě byla provedena v Matlabu za použití funkce *Neural Time Series* pod integrovaným *Neural Network Toolbox.*

Pro predikci byla použita nelineární input-output síť s *problem definition* (2), kde *x* je jeden vstup a *y* je jeden výstup, deseti skrytými neurony a hodnotou zpoždění $d = 2$

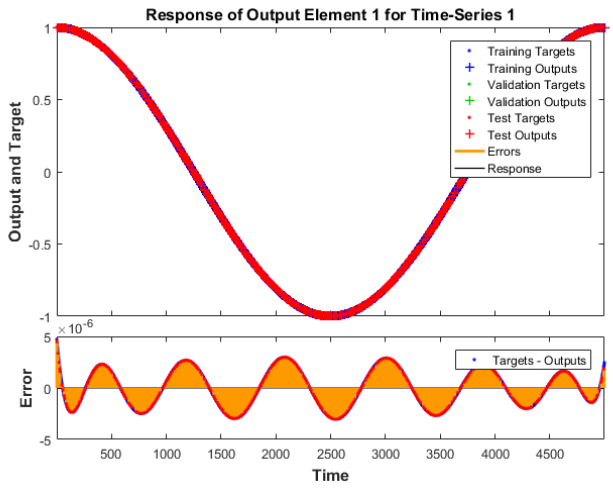
$$y(t) = f(x(t-1),...,x(t-d)) \tag{2}$$

Schéma sítě je znázorněno na obr. 11.



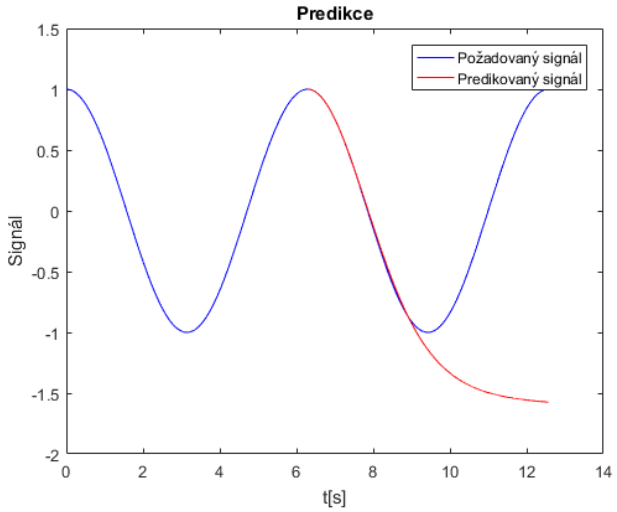
Obr. 11: Schéma neuronové sítě pro predikci

Výstup z učení je na obr. 12, kdy střední kvadratická chyba učení byla $3{,}36 \cdot 10^{-12}$, což je velmi dobrý výsledek na takto malý počet skrytých neuronů.



Obr. 12: Výstup učení neuronové sítě pro predikci

Výsledek predikce funkcí vytvořenou neuronovou sítí je znázorněn na obr. 13. Následující vývoj funkce byl velmi dobře predikován až přibližně do času $t = 2{,}8\pi$, kdy dochází k odpojení průběhu predikované funkce od funkce požadované a následná předpověď vývoje tedy není nadále přesná.



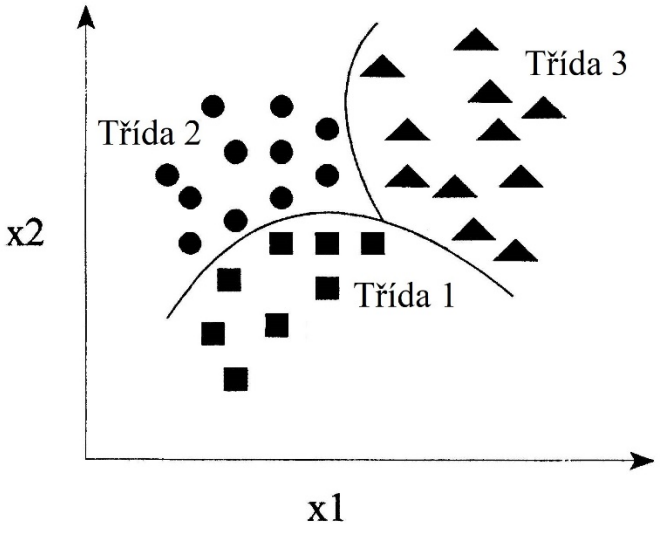
Obr. 13: Predikce

V příloze 2 je naučenou sítí vygenerovaná funkce *prediction* a skript, který generoval náhodná data pro učení této sítě a zároveň vykresloval výstupy ve formě grafů.

5.3 Klasifikace

Klasifikace spočívá v přiřazení neznámého vzoru, za využití řízeného učení (*supervised learning*), do jedné z několika předdefinovaných kategorií v závislosti na jedné, nebo více vlastnostech, které tuto kategorii charakterizují. Toto znázorňuje obr. 14. Možné použití

klasifikace zahrnuje oblast počítačového vidění, a to konkrétně k rozpoznávání objektů, jako mohou být například písmena a číslice z fotek a obrázků. [1]

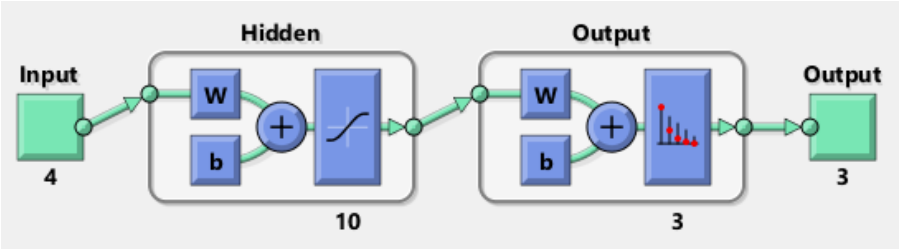


Obr. 14: Klasifikace dat (převzato z [1])

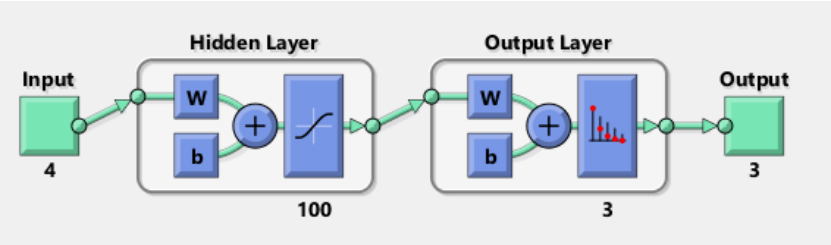
5.3.1 Praktický příklad klasifikace

Klasifikace bude demonstrována na uměle vytvořeném třídění hřídelí dle 4 parametrů do 3 skupin – kusy vyhovující, nevyhovující opravitelné a nevyhovující neopravitelné (zmetky). Byla zvolena myšlená hřídel kontrolované délky $300^{+0,1}_{-0,1}\,mm$ a průměru $60^{+0}_{-0,03}\,mm$ (tolerance rozměru hřídele h7), kontrolované tolerance válcovitosti $4\,\mu m$ a minimální drsnosti povrchu Ra 3,2. V tomto příkladu bude demonstrován i vliv počtu neuronů skryté vrstvy a četnost prvků v souboru pro učení sítě v Matlabu za použití funkce *Neural Pattern Recognition* pod integrovaným *Neural Network Toolbox*.

Obě testované sítě jsou dvouvrstvé dopředné sítě se 4 vstupy (délka, průměr, válcovitost a drsnost povrchu), skrytou vrstvou neuronů, ve které se v prvním případě nachází 10 neuronů, nadále síť 1, a v druhém síť 2 se 100 neurony (schéma na obr. 15 a 16), a třemi výstupy, které určují, do které ze tří kategorií bude daný kus patřit. Pro první případ obsahoval učící soubor 5 000 prvků a pro druhý 1 000 000 prvků. Učícím pravidlem bylo *scaled conjugate gradient backpropagation*.



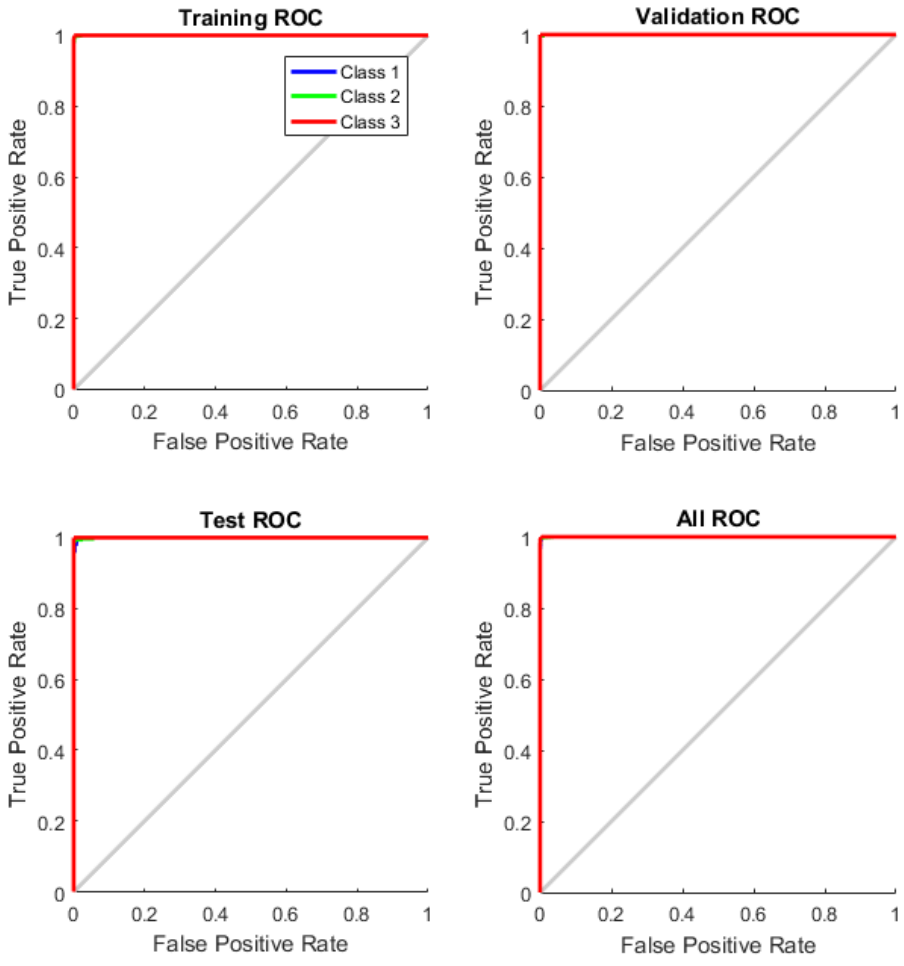
Obr. 15: Schéma neuronové sítě 1



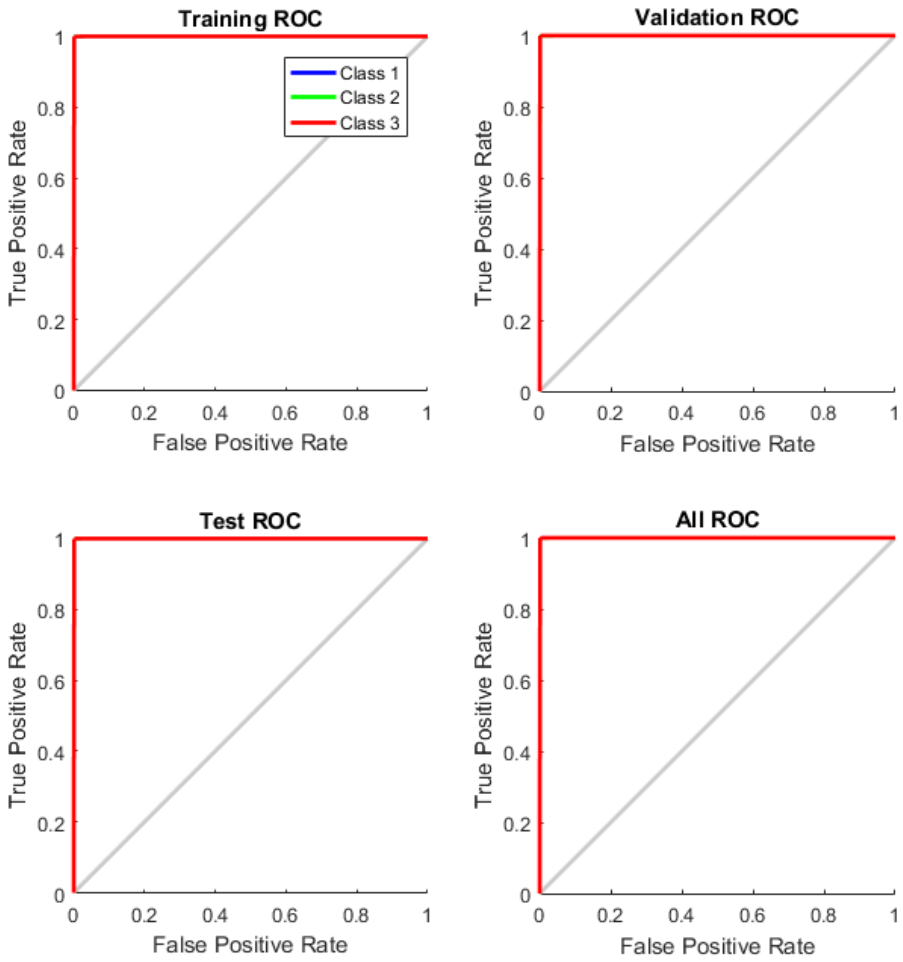
Obr. 16: Schéma neuronové sítě 2

Cílem příkladu bylo naučit sítě rozřazovat kusy, vygenerovat s jejich pomocí funkce rozřazující kusy o parametrech, na které nebyly testovány, změřit přesnost rozřazování na souboru 1 000 000 nových prvků a přesnosti obou porovnat.

Učení sítě 1 bylo velmi rychlé, v řádu jednotek sekund, a na obr. 17 jsou znázorněny provozní charakteristiky pro síť 1, na obr. 18 potom pro síť 2, jejíž učení bylo výrazně pomalejší a trvalo 37 minut. Čím ostřejší je přechod grafu z vertikální do horizontální linky, tím lepších výsledků bude klasifikační funkce generovaná sítí dosahovat. Již zde je dobře viditelné, že rozdíl přesnosti obou sítí nebude velký.



Obr. 17: Provozní charakteristiky sítě 1



Obr. 18: Provozní charakteristiky sítě 2

Následné testování na 1 000 000 vzorků ukázalo, že obě sítě, i přes výrazný rozdíl ve vstupních hodnotách pro učení a v počtu neuronů, jsou pro řešení tohoto případu prakticky identické. Výsledky testování zobrazuje tab. 1.

Síť	Počet chyb	Přesnost [%]
1	39 827	99,2035
2	7 063	99,8587

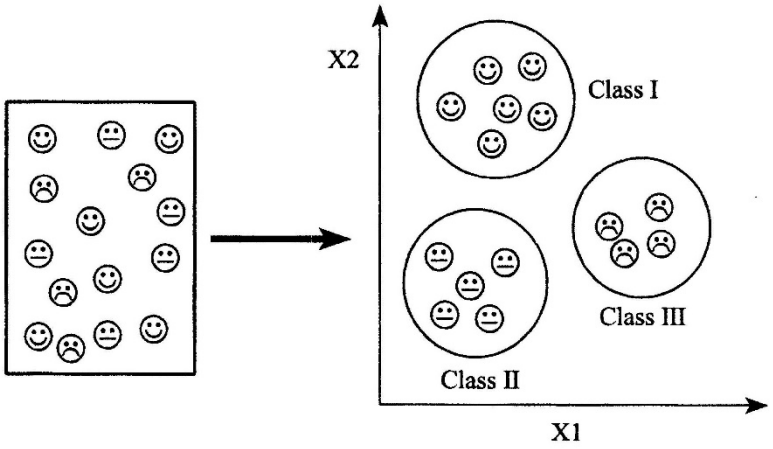
Tab. 1: Výsledky klasifikace

Rozdíl mezi funkcemi pro klasifikaci generovanými oběma sítěmi byl v celkové přesnosti prakticky zanedbatelný – pouhých 0,6 %. V celkovém objemu výroby 1 000 000 kusů by však použití sítě 1 znamenalo o 32 764 špatně posouzených hřídelí více., což by se negativně promítlo na zisku firmy.

V příloze 3 jsou naučenou sítí vygenerované funkce *classification* (síť 1), *class_acc* (síť 2) a skript, který generoval náhodná data pro učení obou sítí a zároveň vykresloval výstupy ve formě grafů.

5.4 Shlukování (clustering)

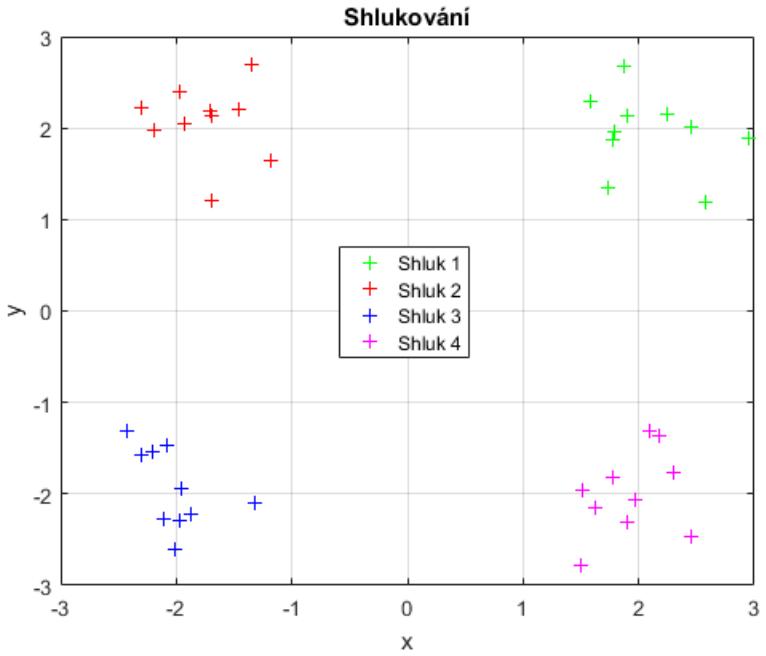
Shlukování se provádí při neřízeném učení (*unsupervised learning*), kdy shluky (třídy), jsou formovány zkoumáním podobností a rozdílností vstupních dat, jak ukazuje obr. 19. Síť zařazuje podobné vzory do stejného shluku. [1]



Obr. 19: Shlukování (převzato z [1])

5.4.1 Praktický příklad shlukování

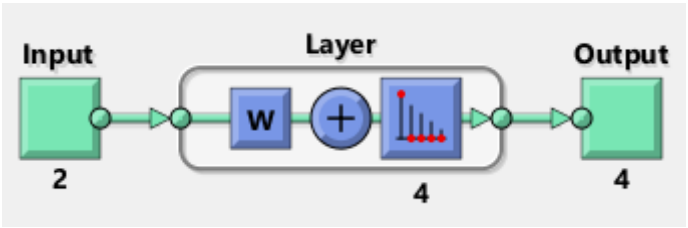
Pro demonstraci shlukování byl vytvořen testovací soubor 40 bodů ve 2D prostoru, viditelně dělitelný do 4 shluků po 10 bodech podle umístění v jednom ze 4 kvadrantů, jak ukazuje obr. 20. Cílem bylo demonstrovat schopnosti neuronové sítě tento vzor detekovat a body dle něj dělit do 4 kategorií.



Obr. 20: Vstupní body pro shlukování

Pro tento účel byla použita samo-organizační mapa (*self-organizing map*, SOM) v Matlabu za použití funkce *Neural Clustering* pod integrovaným *Neural Network Toolbox*. Učícím pravidlem bylo pravidlo pro samo-organizační mapy.

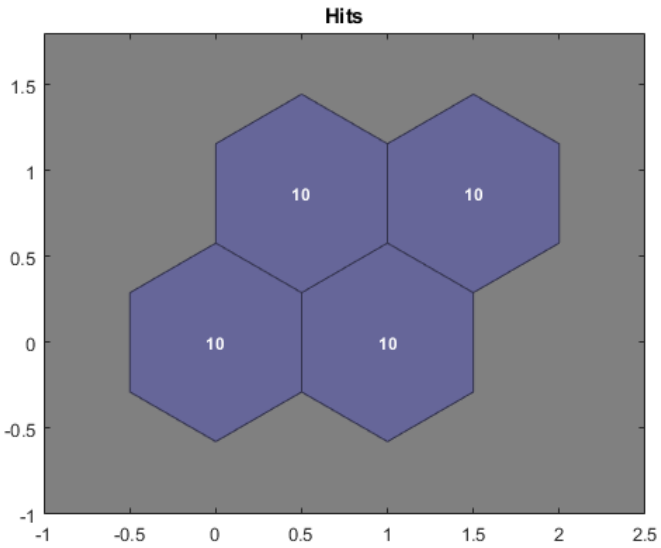
Síť dělí zadaná data do shluků (*clusters*) jejichž počet udává počet neuronů SOM vrstvy sítě. Čím více neuronů má SOM vrstva k dispozici, tím do více kategorií je síť schopna data dělit. SOM v Matlabu je dvojdimenzionální s $a \times a$ rozložením neuronů. Pro příklad je dostačující mapa 2×2, schéma na obr. 21, nicméně velmi pěkně je vidět to samé dělení v mapě 3×3, jejíž schéma se liší oproti předcházející pouze právě tímto počtem neuronů SOM vrstvy.



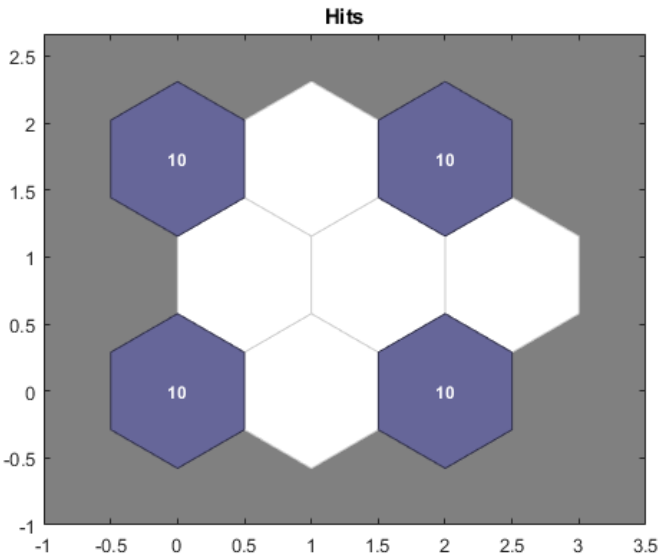
Obr. 21: Schéma sítě pro shlukování se SOM vrstvou

Výstupem naučení sítě je graf zobrazující tzv. *sample hits*, tedy počty prvků zařazených v každém shluku sítě. Pro obě testované mapy neuronová síť správně nalezla dělní zadaných dat do 4 shluků, jak je znázorněno na obr. 22 a 23. Na obr. 23 je vidět, že sít neuronů, která udávala počet shluků na 9, využila také pouze 4 shluky a zbylých 5 zůstalo prázdných. To je dáno poměrně jasným oddělením vlastností zadaných prvků a zároveň jejich vysoké podobnosti mezi sebou, proto byly body každého kvadrantu zařazeny do společného shluku. V případě použití SOM větších velikostí by byly i prvky každého shluku

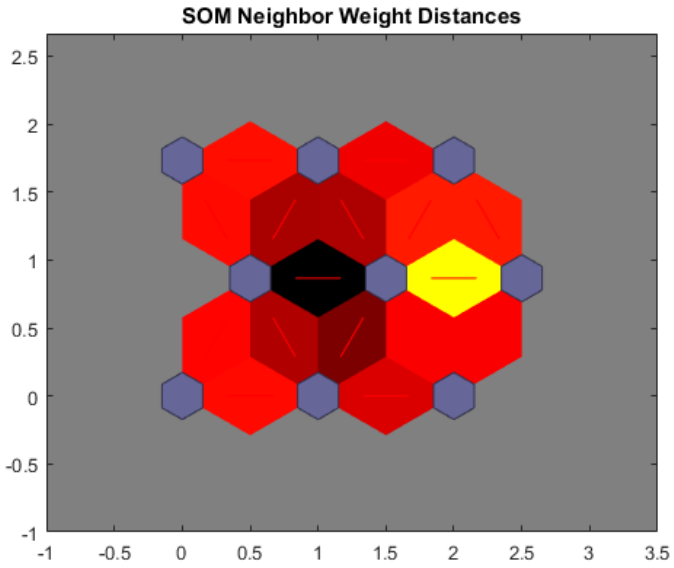
rozděleny do několika sousedních shluků s větší podobností oproti zbylým shlukům. Podobnost sousedních shluků zobrazuje obr. 24, kde je znázorněn graf tzv. *Neighbor weight distances*, který zobrazuje podobnost shluků pomocí barev, kdy světlejší barva znamená větší podobnost vlastností a tmavá znamená menší podobnost. V tomto případě je velká podobnost vlastností pouze mezi dvěma shluky, ve kterých není zařazen ani jeden z prvků a podobnosti mezi shluky, do kterých jsou prvky zařazeny a jejich sousedy je malá, což dokazuje také to, že do těchto sousedních shluků nebyl zařazen žádný prvek testovaného souboru.



Obr. 22: Shlukování se SOM 2×2



Obr. 23: Shlukování se SOM 3×3



Obr. 24: Neighbor weight distances

Po naučení neuronová síť generuje funkce *clustering* a *clustering9*, které při zadání souřadnic bodů jako vstupního parametru vrátí sloupcový vektor, který zařadí 1 na řádek představující shluk, do kterého má bod být zařazen. Tyto funkce jsou zařazeny v příloze 4 společně se skriptem, který generoval náhodná data pro naučení sítě a vykresloval výstupy formou grafů.

6 ZÁVĚR

V práci byla zpracována teorie neuronových sítí, jejich historie a podobnost s biologickými strukturami, na kterých jsou tyto umělé neuronové sítě založeny. Z hlediska teoretického byly popsány principy fungování sítě a jejich matematický základ. Dále byly podrobněji rozebrány 3 důležité neuronové sítě, které jsou v dnešní době nejvíce využívány – dopředná vícevrstvá síť se zpětným šířením chyby, Hopfieldova síť a Kohonenovy samo-organizační mapy.

V kapitole 5 byly popsány a předvedeny základní možnosti aplikací neuronových sítí, a to na příkladech modelování, klasifikace, predikce a shlukování. Výsledky byly přes použité jednoduché sítě s malým počtem neuronů skryté neuronové vrstvy velmi dobré. Vyšší přesnosti by šlo dosáhnout navýšením počtu neuronů sítě, či použítím jiného typu neuronové sítě v případě modelování, predikce a klasifikace. V případě shlukování by se pro dosažení vyšší přesnosti rozšiřoval počet neuronů kompetiční vrstvy a tím zvětšovalo 2D pole a s ním i počet shluků, do kterých se vstupní data dělila. V každém shluku by se pak nacházela data s větším koeficientem podobnosti.

Neuronových sítí a jejich typů je mnoho a v této práci byly popsány pouze základní z nich. Pro řešení modelování, predikce a klasifikace by šlo použít mimo předvedených dopředných i sítě zpětné (*Recurrent Neural Networks*), jako jsou popsané Hopfieldovy, nebo Boltzmanovy, a pro shlukování jiné typy samo-organizačních map. Všechny tyto patří mezi dynamické neuronové sítě, které se vyvinuly jako pokračování prvních sítí, které se nazývaly statickými a patřil mezi ně například slavný perceptron, který dal základy celému oboru. Další podrobnější zpracování by si zasloužily i tzv. *Memory networks*.

7 SEZNAM POUŽITÉ LITERATURY

[1] BASHEER, I.A a M HAJMEER. Artificial neural networks: fundamentals, computing, design, and application. *Journal of Microbiological Methods* [online]. Elsevier B.V, 2000, 43(1), 3-31 [cit. 2017-04-22]. DOI: 10.1016/S0167-7012(00)00201-3. ISSN 0167-7012.

[2] MCCULLOCH, W a W PITTS. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*. 1990, 52(1-2), 99-115. DOI: 10.1016/S0092-8240(05)80006-0. ISSN 00928240. Dostupné také z: http://www.springerlink.com/index/10.1016/S0092-8240(05)80006-0

[3] HEBB, D. O. *The organization of behavior: a neuropsychological theory*. Mahwah, N.J.: L. Erlbaum Associates, c2002. ISBN 978-0805843002.

[4] MACUKOW, Bohdan. *Neural Networks – State of Art, Brief History, Basic Models and Architecture*., 3. DOI: 10.1007/978-3-319-45378-1_1. Dostupné také z: http://link.springer.com/10.1007/978-3-319-45378-1_1

[5] ROSENBLATT, Frank. *Principles of neurodynamics: perceptrons and the theory of brain mechanisms*. Spartan Books, 1962.

[6] KLOPF, A. H. AIR FORCE CAMBRIDGE RESEARCH LABS HANSCOM AFB MA. *Brain Function and Adaptive Systems: A Heterostatic Theory*. 1972.

[7] Giraffe. *Chess Programming Wiki* [online]. San Francisco: Tangient, 2005 [cit. 2017-04-22]. Dostupné z: https://chessprogramming.wikispaces.com/Giraffe

[8] Artificial intelligence: Google's AlphaGo beats Go master Lee Se-dol. *BBC* [online]. London: BBC, 1997 [cit. 2017-04-22]. Dostupné z: http://www.bbc.com/news/technology-35785875

[9] Stonásobný výkon. Nový čip Intelu cílí na umělou inteligenci. *IDnes.cz* [online]. Praha: MAFRA, 1999 [cit. 2017-04-22]. Dostupné z: http://technet.idnes.cz/intel-umela-inteligence-deep-learning-d8k-/tec_technika.aspx?c=A161122_194021_tec_technika_dvz

[10] Fotbalistům již simulování neprojde. Umělá inteligence pohlídá fauly. *IDnes.cz* [online]. Praha: MAFRA, 1999 [cit. 2017-04-22]. Dostupné z: http://technet.idnes.cz/cebit-analyza-fotbal-simulovani-dak-/tec_reportaze.aspx?c=A170320_162722_tec_reportaze_nyv

[11] 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]. Elsevier B.V, 2000, 22(5), 717-727 [cit. 2017-04-16]. DOI: 10.1016/S0731-7085(99)00272-1. ISSN 0731-7085.

[12] ŠNOREK, Miroslav. *Neuronové sítě a neuropočítače*. Praha: ČVUT Praha, 2004, 156 s. ISBN 80-01-02549-7.

8 SEZNAM PŘÍLOH

Příloha 1 Funkce a skript pro modelování
Příloha 2 Funkce a skript pro predikci
Příloha 3 Funkce a skript pro klasifikací
Příloha 4 Funkce a skript pro shlukování
Příloha 5 CD s elektronickou verzí práce a všemi funkcemi a skripty ve formátu *.m pro software Matlab

PŘÍLOHY

Příloha 1

Funkce *fitdata* generovaná naučenou neuronovou sítí v softwaru Matlab pro příklad modelování v kapitole 5.1.1

```
function [y1] = fitdata(x1)
%fitdata neural network simulation function.
%
% Generated by Neural Network Toolbox function genFunction, 13-May-
2017 13:44:38.
%
% [y1] = fitdata(x1) takes these arguments:
%   x = 1xQ matrix, input #1
% and returns:
%   y = 1xQ matrix, output #1
% where Q is the number of samples.

%#ok<*RPMT0>

% ===== NEURAL NETWORK CONSTANTS =====

% Input 1
x1_step1.xoffset = 0;
x1_step1.gain = 0.106103295394597;
x1_step1.ymin = -1;

% Layer 1
b1 = [-6.2893932918513862;3.8389485929001159;2.9031667826111969;-
3.3150740486663617;0.52928847537354773;-
0.60168801506321046;2.6498239336318212;1.6718948144756347;-
3.1112484009141088;-6.6031842184102434];
IW1_1 = [5.3311261415422626;-4.6238300486033275;-
6.2880780460600434;5.7399059743349401;-3.9302019407245452;-
3.4040133077067987;-7.5570116028100021;3.4053621882164267;-
3.871674272944841;-6.0000428030882942];

% Layer 2
b2 = -1.9286742483120027;
LW2_1 = [-2.9213693555127338 -2.6642686413700183 1.2368739891252816 -
1.2737582141750521 -4.3641665926010056 6.5296712393123055
0.29126225120145555 6.5871055291578102 4.5326842636597622 -
1.8964668376281519];

% Output 1
y1_step1.ymin = -1;
y1_step1.gain = 0.991640726683129;
y1_step1.xoffset = -1.00833555992572;

% ===== SIMULATION ========

% Dimensions
Q = size(x1,2); % samples

% Input 1
xp1 = mapminmax_apply(x1,x1_step1);
```

```
% Layer 1
a1 = tansig_apply(repmat(b1,1,Q) + IW1_1*xp1);

% Layer 2
a2 = repmat(b2,1,Q) + LW2_1*a1;

% Output 1
y1 = mapminmax_reverse(a2,y1_step1);
end

% ===== MODULE FUNCTIONS ========

% Map Minimum and Maximum Input Processing Function
function y = mapminmax_apply(x,settings)
y = bsxfun(@minus,x,settings.xoffset);
y = bsxfun(@times,y,settings.gain);
y = bsxfun(@plus,y,settings.ymin);
end

% Sigmoid Symmetric Transfer Function
function a = tansig_apply(n,~)
a = 2 ./ (1 + exp(-2*n)) - 1;
end

% Map Minimum and Maximum Output Reverse-Processing Function
function x = mapminmax_reverse(y,settings)
x = bsxfun(@minus,y,settings.ymin);
x = bsxfun(@rdivide,x,settings.gain);
x = bsxfun(@plus,x,settings.xoffset);
end
```

Skript v softwaru Matlab volající tuto funkci a vykreslující potřebné grafy závislostí. Tento skript také generoval náhodná data pro naučení, kterých je příliš mnoho, aby zde byly vypisovány.

```
clc, clear all;
x = linspace(0,6*pi,5000);
y1 = cos(x);
r = ones(1, length(x)) + rand(1,length(x))*0.01 - rand(1,len-
gth(x))*0.01;

y = y1.*r;

f = fitdata(x);

plot(x,y,'b',x,f,'r')
title('Aproximace funkce')
xlabel('t[s]')
ylabel('Signál')
legend('Naměřený signál','Aproximovaný signál')
```

Příloha 2

Funkce *prediction* generovaná naučenou neuronovou sítí v softwaru Matlab pro příklad predikce v kapitole 5.2.1

```
function [y1,xf1] = prediction(x1,xi1)
%prediction neural network simulation function.
%
% Generated by Neural Network Toolbox function genFunction, 13-May-
2017 17:04:50.
%
% [y1,xf1] = prediction(x1,xi1) takes these arguments:
%   x1 = 1xTS matrix, input #1
%   xi1 = 1x2 matrix, initial 2 delay states for input #1.
% and returns:
%   y1 = 1xTS matrix, output #1
%   xf1 = 1x2 matrix, final 2 delay states for input #1.
% where TS is the number of timesteps.

% ===== NEURAL NETWORK CONSTANTS =====

% Input 1
x1_step1.xoffset = 0;
x1_step1.gain = 0.318309886183791;
x1_step1.ymin = -1;

% Layer 1
b1 = [-0.595961021536347;1.2800277346217119;-0.8182242905237509;-
0.62044667115114194;0.82459637060811586;0.52264969430234942;2.51370320
89128835;2.5823440605372592;0.5896953775647833;-1.1008478210598971];
IW1_1 = [0.78294514818539152 0.8181131370427569;-0.93232153650464 -
0.93323509546437411;-0.4911112751476166 -
0.45203867774272094;0.16628672834190697 -
0.20517318155041267;0.65763604442885337
0.28785874663520128;0.71983993638756205 0.9161181129567616;-
1.0662827267771411 -0.7168850331790908;0.83566135252475759
0.99563498680206508;0.17197669373582838 -0.13424430757050262;-
0.88691663513485219 -0.79274927935696238];

% Layer 2
b2 = -0.88751857557338276;
LW2_1 = [2.0056139934126795 -0.76020913632289444 -0.47012605240661692
0.6354475454699845 0.46329944492553649 -1.6708709446624088
2.0634398145881114 1.7949800159303653 -0.70288689971640972
1.6330371727490471];

% Output 1
y1_step1.ymin = -1;
y1_step1.gain = 1.00000009873554;
y1_step1.xoffset = -0.999999802528938;

% ===== SIMULATION ========

% Dimensions
TS = size(x1,2); % timesteps

% Input 1 Delay States
xd1 = mapminmax_apply(xi1,x1_step1);
xd1 = [xd1 zeros(1,1)];
```

```
% Allocate Outputs
y1 = zeros(1,TS);

% Time loop
for ts=1:TS

    % Rotating delay state position
    xdts = mod(ts+1,3)+1;

    % Input 1
    xd1(:,xdts) = mapminmax_apply(x1(:,ts),x1_step1);

    % Layer 1
    tapdelay1 = reshape(xd1(:,mod(xdts-[1 2]-1,3)+1),2,1);
    a1 = tansig_apply(b1 + IW1_1*tapdelay1);

    % Layer 2
    a2 = b2 + LW2_1*a1;

    % Output 1
    y1(:,ts) = mapminmax_reverse(a2,y1_step1);
end

% Final delay states
finalxts = TS+(1: 2);
xits = finalxts(finalxts<=2);
xts = finalxts(finalxts>2)-2;
xf1 = [xi1(:,xits) x1(:,xts)];
end

% ===== MODULE FUNCTIONS ========

% Map Minimum and Maximum Input Processing Function
function y = mapminmax_apply(x,settings)
y = bsxfun(@minus,x,settings.xoffset);
y = bsxfun(@times,y,settings.gain);
y = bsxfun(@plus,y,settings.ymin);
end

% Sigmoid Symmetric Transfer Function
function a = tansig_apply(n,~)
a = 2 ./ (1 + exp(-2*n)) - 1;
end

% Map Minimum and Maximum Output Reverse-Processing Function
function x = mapminmax_reverse(y,settings)
x = bsxfun(@minus,y,settings.ymin);
x = bsxfun(@rdivide,x,settings.gain);
x = bsxfun(@plus,x,settings.xoffset);
end
```

Skript v softwaru Matlab volající tuto funkci a vykreslující potřebné grafy závislostí. Tento skript také generoval náhodná data pro naučení, kterých je příliš mnoho, aby zde byly vypisovány.

```
clc, clear all;
x = linspace(0,4*pi,5000);
```

```
y = cos(x);

x2 = linspace(2*pi,4*pi,5000);
y2 = prediction(x2,x2);

plot(x, y,'b',x2,y2,'r')
title('Predikce')
xlabel('t[s]')
ylabel('Signál')
legend('Požadovaný signál','Predikovaný signál')
```

Příloha 3

Funkce *classification* generovaná naučenou neuronovou sítí v softwaru Matlab pro příklad klasifikace v kapitole 5.3.1

```
function [y1] = classification(x1)
%classification neural network simulation function.
%
% Generated by Neural Network Toolbox function genFunction, 14-May-
2017 14:51:51.
%
% [y1] = classification(x1) takes these arguments:
%   x = 4xQ matrix, input #1
% and returns:
%   y = 3xQ matrix, output #1
% where Q is the number of samples.

%#ok<*RPMT0>

% ===== NEURAL NETWORK CONSTANTS =====

% Input 1
x1_step1.xoffset =
[299.801762464157;59.9352331188008;0.0010483789166051;1.6];
x1_step1.gain =
[5.06852103569809;20.0577741500847;512.604834967615;1.25];
x1_step1.ymin = -1;

% Layer 1
b1 =
[2.7954579603040051;8.2975681209877017;8.5807766245524313;1.3365352747
950632;-6.8014339656019338;-6.5278038068410105;9.2241111388474586;-
1.2911504804307155;3.1405337636050019;1.6323802831941989];
IW1_1 = [-0.57268859263296945 6.3518133206032186 0.7326775548612311 -
0.35591486140815837;-16.297757934539884 0.50628439425838789 -
0.041523589648013906 0.11397479950792565;-0.39008684595679305 -
0.4726188293323671 -16.15993168965073 -
0.019886809715572402;0.18060638011353683 0.96973720696208243
6.6809513292279954 -6.0654390480452989;-0.69928127495144998
7.1270141945200747 0.26175974071154207 -
0.061756842771189688;0.17773554623661927 -20.602725850125818 -
0.16328367312521466 0.037738859311079176;17.400926498368751 -
0.13470905345931561 0.056514337718879148
0.078645907331390916;2.2267659149287731 2.0777029666575211 -
0.013649470666384837 -0.49280710148125773;-0.2183931224136762
1.5272422003210415 -1.3962184437095542 -
1.0041682771507363;0.46102349459119191 0.55907191734226014
4.6570530943756356 4.4153143272411475];

% Layer 2
b2 = [10.541475996601156;-8.217277950303842;-2.2687198043500585];
LW2_1 = [0.84622571614695752 1.246545451186583 -10.314611051832854
3.42162142179725 -3.6369899037728075 10.581895780550807 -
10.286286357955236 -5.5677414810724857 7.7997179614518259
2.6819805954001406;-2.3711885080444359 6.4092726334393957
3.2911325362311383 -1.0896764066639266 -5.3184603280941243 -
4.240204600948001 6.3042360801300683 0.074598000915771692 -
6.7906507734263908 -2.4871561984764878;0.51480755162605329 -
```

```
6.1136903159511 6.9337467005179061 -2.9662295504706839
9.7793507173456398 -6.1067911591071615 3.1640038651281404
3.9693198892815236 -2.6200560605960632 -1.8594049355454842];

% ===== SIMULATION ========

% Dimensions
Q = size(x1,2); % samples

% Input 1
xp1 = mapminmax_apply(x1,x1_step1);

% Layer 1
a1 = tansig_apply(repmat(b1,1,Q) + IW1_1*xp1);

% Layer 2
a2 = softmax_apply(repmat(b2,1,Q) + LW2_1*a1);

% Output 1
y1 = a2;
end

% ===== MODULE FUNCTIONS ========

% Map Minimum and Maximum Input Processing Function
function y = mapminmax_apply(x,settings)
y = bsxfun(@minus,x,settings.xoffset);
y = bsxfun(@times,y,settings.gain);
y = bsxfun(@plus,y,settings.ymin);
end

% Competitive Soft Transfer Function
function a = softmax_apply(n,~)
if isa(n,'gpuArray')
    a = iSoftmaxApplyGPU(n);
else
    a = iSoftmaxApplyCPU(n);
end
end
function a = iSoftmaxApplyCPU(n)
nmax = max(n,[],1);
n = bsxfun(@minus,n,nmax);
numerator = exp(n);
denominator = sum(numerator,1);
denominator(denominator == 0) = 1;
a = bsxfun(@rdivide,numerator,denominator);
end
function a = iSoftmaxApplyGPU(n)
nmax = max(n,[],1);
numerator = arrayfun(@iSoftmaxApplyGPUHelper1,n,nmax);
denominator = sum(numerator,1);
a = arrayfun(@iSoftmaxApplyGPUHelper2,numerator,denominator);
end
function numerator = iSoftmaxApplyGPUHelper1(n,nmax)
numerator = exp(n - nmax);
end
function a = iSoftmaxApplyGPUHelper2(numerator,denominator)
if (denominator == 0)
    a = numerator;
else
```

```
    a = numerator ./ denominator;
end
end

% Sigmoid Symmetric Transfer Function
function a = tansig_apply(n,~)
a = 2 ./ (1 + exp(-2*n)) - 1;
end
```

Funkce *class_acc* generovaná naučenou neuronovou sítí v softwaru Matlab pro příklad klasifikace v kapitole 5.3.1

```
function [y1] = class_acc(x1)
%class_acc neural network simulation function.
%
% Generated by Neural Network Toolbox function genFunction, 14-May-
2017 14:48:42.
%
% [y1] = class_acc(x1) takes these arguments:
%   x = 4xQ matrix, input #1
% and returns:
%   y = 3xQ matrix, output #1
% where Q is the number of samples.

%#ok<*RPMT0>

% ===== NEURAL NETWORK CONSTANTS =====

% Input 1
x1_step1.xoffset =
[299.800087530592;59.9350674156421;0.00100195798873526;1.6];
x1_step1.gain =
[5.00152886520795;20.0321256997835;500.288429305272;1.25];
x1_step1.ymin = -1;

% Layer 1
b1 = [4.3449691608419743;-5.5841192974729132;-4.2994810618851451;-
6.1226652002379369;-
5.1804107247890814;5.1541478049524967;5.7653769744781513;5.75750056699
38282;3.1769351275973627;-
6.2165283472867063;3.0873860162968372;3.8897471988290802;3.82892429465
04443;-2.8276404717334107;3.7055638632863346;-9.6718887149826962;-
18.288448383536256;-4.1081489271333202;2.9841780358774312;-
7.1780990329244156;-3.6290862861209265;-
5.1130464828959399;3.3240070812941904;2.2325050492272154;-
4.0918286593303463;-1.463394570475451;-
2.2991510538710211;11.828047696527214;4.2059556405649836;-
2.7329649400452101;2.462908483515958;4.5465146925759496;3.777510341656
7561;-3.9085418626766488;1.8126924376730362;-2.4031240404030108;-
2.1122103431504216;0.17840485646227888;1.1418029782851011;-
0.17982080684545421;3.7379422390473893;-0.43867486434240122;-
1.5489134005870207;0.033806279786511281;-0.48898711303749176;-
0.97143253580734146;-2.4366897166931238;3.7868526370335251;-
1.7129031727122215;-0.084651684375432495;-
2.2211506507107117;4.8630828114287388;-
0.80766132290432102;2.2900382521588538;0.81684580197359324;-
1.1922605016306898;0.1461091550983592;-1.1569252272168873;-
1.6769632053162353;-4.564238827324413;-0.4036471599900317;-
1.5288182071743677;2.7025467032253507;4.9867729357254911;2.00248571991
```

```
89903;1.3375051583030335;-
0.43262987825010424;1.0922407720627689;3.7037968669871888;-
0.41564408319768492;-2.3514442064922165;-1.252543947824011;-
4.4985049261941965;-
7.7311488194185332;2.4239140778289752;4.6282908350202492;-
1.0795923481350385;1.1696926166140575;-1.4791720390358609;-
3.5596685432480504;2.0892720995953566;4.034330527583788;3.485161604487
0353;1.3972533866286116;-2.8002472250235426;-2.3090707888324209;-
4.8374419258197472;-4.5444962677696097;-
2.230284619398307;3.0970900550447529;-4.3903930353117753;-
5.6496990609532745;4.0551456859273189;-
3.7529205099122982;2.8018046161060779;3.5851707409280493;3.31463994806
25313;-5.5635552014619511;5.8647164778397167;-8.7042059548204271];
IW1_1 = [-1.1716621265036189 3.0231417670470404 1.7400359799898271
3.7684063262933298;0.37680862496721235 2.5269234700781018
1.7588994359377723 -0.46033628453043518;0.24809335891501913
2.6975028615832874 -2.1236350804348536 -
1.1533767874426173;0.23684028814075564 -1.7912659729742131 -
1.9088844497000284 -0.31275244324252105;2.2789609685033692 -
1.8293667458978569 1.8471338903091965 0.32360788603536067;-
0.74186831934299602 4.8753362094927084 -6.1780958702352127
0.19129772536731851;-6.0716946521139272 2.6509298828694408
0.55277665123009301 2.675369030573918;-5.5015007262713409 -
2.5394067691331434 -2.9122830071516042
0.045406651118847369;0.98394870994653127 0.2987677082830178
4.0738049969535659 2.9240655416850982;10.026495260572789 -
1.0617463176305579 0.035400850848222823 -1.4513715927802902;-
0.53896387822185798 4.2113166802688315 1.9157443763711581
3.7714001219816584;-5.8465190225616066 -1.7524132562348993 -
0.69183843913211773 -6.1217435800848623;-6.9440624156653898
0.24147997068704541 -0.046998901684374789 -
0.10383705144455271;3.0819172212752375 -0.75766559409370116 -
3.1019031795510452 4.1236550973609631;-5.0794213249956233
4.981207791164926 -0.88830024836683985 -
4.2910231361612832;12.767241799577389 -1.3160912414942072
0.38651979204977366 -3.7122092015794146;0.66848356705264511
0.50473765307380547 35.481852054630664 -
0.010671415520059509;2.7446945914002883 -2.766118517144927 -
1.2651464807008708 2.7386773982021642;-4.5122183477002347
3.1223820945801233 1.6627890020657554 4.3591431579127367;-
1.0793119411347025 7.6602741571885957 -0.424045909917252 -
0.0021839439581303305;5.9862018204061789 -0.31759838061042661
4.9270623589046112 -2.3020793219385181;4.2856596685657857 -
4.7118608640456001 -1.6715806585960964
2.7733351602462677;1.1206514163759029 5.2878408029410151 -
1.5671610597915866 1.5486251916767462;-4.2886640216907992
5.0864898148686599 -1.1011124941212092
1.91525485985068;2.5712301648925786 -2.0968493006391626 -
1.6084565428581865 -1.8401391028879059;6.875262128567627 -
0.84003155812873176 0.24139904117038788
2.7484884505939231;3.7156818160635612 0.53860557235125461 -
2.6880980972129018 -0.035443457020783727;-0.79068985506974998
37.608880805402237 0.45180470615630869 0.043717498172264481;-
2.8568718420099981 -3.2573540598104311 -2.3594741641008277
4.9588966118353399;5.601432831732188 -5.6555168848170467
0.95320541561833938 2.6324536659663416;-3.4847475494631488
5.4944832531976919 -0.5155016193481422 1.4524453242362454;-
4.7385101872617712 -3.2684860878653041 0.38692494911381936
3.8129655033190306;2.3966389722019819 -8.5435687171216017 -
0.84165066327785654 8.5684327897228094;4.7025157674400955
2.3352493117131958 3.0840311694444869 5.5517994012931204;-
```

2.212298463970289 -3.2371016881037566 2.0915760411181981
0.31033238783297057;3.484935813721286 2.3280920502123341
0.1799510199358349 -0.18410162106492106;0.38889145941580006 -
8.6192347403625913 -3.8062841442365305 0.057395233563838473;-
4.7979411144419881 3.9679211029887043 -0.92310001030674105
1.2117854338204535;-3.0373261809903558 0.65703108273747102
3.0033147623059544 -3.2227973072531824;-1.7092468224591619 -
0.43315814154588844 -4.1220372693521679 -4.9774049621240204;-
3.9282990101905519 3.3322261981581303 1.6937179534174285 -
5.1075027202862486;-9.1450237863115369 -1.9462021059057972
1.19275519031168 -5.4169099419014044;4.056243890093568 -
3.4411103757067827 -1.0645183375917737 -
3.9631337719336663;7.9922808894434665 1.7565235596370687 -
0.80667333455839751 -4.5737348992867926;-2.3267303585828158 -
1.5476306946512348 -2.6043361860050922
4.7128948932978059;3.6957677308761601 -3.1228769630290873
3.7181471110056643 -0.27693812281667651;-1.149970900956921 -
0.13320692998687672 -8.5910083444657133 -8.1070969643966073;-
1.0994068524495302 -1.1814586369404938 -8.4898873287374244
1.5862410737380246;3.7573824620539522 -2.5708352349208599 -
0.06082123762798048 7 -4.4139312842222456;-5.2500009874503739 -
3.227590250012911 2.085208170733774 -
4.0809635047713666;2.959764489655786 2.2472388163573513 -
1.9260608866537081 2.4816899243221284;13.465212443071174 -
0.50715609073238044 2.5548183264834305 -0.018952639430915463;-
17.510559076776545 0.17434347026912078 -0.20955249945545862
8.4312209532661981;3.3028690070038951 8.0766318816597877
3.8478965559232883 -0.10002538558542458;17.455689814271899 -
0.037529004741313932 0.29373372576996137 8.3664767598451366;-
5.3886444515685401 -1.9528716468402179 -1.4439004639611721
0.66047136783290938;-7.3853094892052074 -2.3564716917348973
0.82164249473124884 -4.6335189472953227;-1.0396681183035903
0.53936234370178315 4.7779553526719551 -0.98420087288533442;-
5.4302947518801457 5.5801549297790061 2.547800452834732 -
0.18228260313709671;-3.1081027748301464 3.7483581770191252 -
3.9258845178924302 -5.6745194296370745;-1.5147209544134841
3.635043057354745 -2.0242098976680447 1.2195362077914185;-
2.925759106472952 2.7479736495709024 1.2565768038329048 -
0.18736725801888413;7.7080416667779579 -6.3487063732904936 -
0.97474841654577948 -0.13956013395440248;10.244070738732841
3.7240959181225199 1.3617550426309946 0.02857447947452614;-
0.33382489072687688 -3.4619223028903274 -3.3690560191072398 -
2.2930449245731284;4.6306498514685854 3.4557344178074301
1.4539354504448443 3.0479978446316567;0.40550066628782472 -
10.652174903068532 -5.2753531930689057
0.10852506844417328;3.3344477787682347 3.6462613466734299
1.1301874634716218 -2.5611219032497492;3.0476784610277239 -
2.5438386350498243 2.1247049027673648 -3.5736123704016509;-
4.0033218283069019 -2.8496219904016367 0.47648093966075394
1.0066551862011826;-2.0443020071010567 -7.6556575694932212 -
0.76530301840508363 -0.015112614698623591;-3.1952275262536514 -
2.3828905160936205 0.29813764848843066 -3.7512124718078885;-
3.6802903339633328 3.3163428921892693 -2.5081080364156092
4.7847901027797617;-8.0939714053582215 6.8457442149260483 -
0.39331772284608069 6.0505788820935278;2.4694596011162764 -
5.1203728268031048 -4.1350288244956213
0.085298215606613978;4.0494696090909628 -3.8057248642163937
2.769449254310929 4.8756331678899905;-9.6521059698592158
1.7819715250632555 -0.85030275538253641
5.3594981095500662;14.333359176645322 -1.2646546653356963
0.26994447612531536 -7.9007988361120924;-1.2060370431322929 -

```
3.0955514283308734 2.0785372655922258 3.8013303383711454;-
3.2811284222731762 2.7518378189125499 -3.9663516781535595
4.7681067342746841;-0.45979322188871014 3.7197638641537347
2.0841937890548174 4.1451101763457912;4.8535646193906814
0.076022466849228393 3.3826427690361776
2.9282937607848347;0.63765224646770291 -4.4383428748043894 -
0.15825323193546015 -1.3906206798071823;1.0468359130713005
1.2769533581768546 4.6358988237750056 3.2983499024461485;-
2.1296730943813342 0.99463254473012275 2.1393105792731242 -
2.972540639038717;-1.2536002842940113 -0.53497664918226906
1.1536911970413806 -4.5605591053994265;-5.4042342555327805 -
7.3155912249899027 1.1206179004837284 -0.095931160172696872;-
1.085862717632077 1.36783508042483 -3.2401925232912814
1.2555983503026409;-3.9554333486220066 -1.1158977978493954
1.3668371710520781 5.4328147791183374;0.14424190318579047 -
2.1445529933387895 4.9314996969524971 -4.6846976957988984;-
1.8452181039288384 -4.7342958586853605 4.8349281185624919 -
0.33198751354228384;-6.1560390913881351 6.3498733662177971
0.37162739401041339 -3.4203635203499028;3.3272562443957403
3.040807277728907 -4.8912481290989644 -0.51471080941213421;-
2.2263551183820005 0.82024027396010801 -2.5227611182447145
2.6682929139308333;0.67899575810612844 0.85771700586439292
9.6583446098565879 -7.5186639192180174;0.83658135702451653 -
4.1882277359392077 -0.90722533811295492
3.2651051463169942;7.8935777415814599 0.88326426964955296
2.5366406132889177 0.094159908758576411;-1.5436301217588484
4.4349879599422621 -0.92121860173475556 -
1.4490194982971198;1.568842945781642 0.56051060466563873
2.3378160369050978 0.065268224426414334;-16.14682533276688
0.79305029365971902 -0.62110575287905223 -0.061688080208546901];

% Layer 2
b2 = [4.9534189714040302;-3.7014473834855277;-1.2627021984840532];
LW2_1 = [0.59051560674930426 -3.0824260676923032 -2.0488442384752785 -
4.0482626304022569 -3.867849260021905 7.1946936858952455
2.0044576143659287 -0.58214407130476498 3.0943297770584794
1.6877726463267031 2.0411207260223643 8.9063381314737669 -
3.3606327546432286 -4.8095456978060911 0.27353562297274842
0.022317374666395287 22.592513715665348 -3.9324640826817765
4.6777970785733149 -2.1220089850483985 5.4790862102439073 -
1.9025822121184743 -2.6770116246469597 -7.1718893973664581 -
1.4668040826465993 5.0217464821129827 -0.55340317353464175 -
24.368423993118999 8.2070219453642572 4.734816946296311 -
4.5455155140135917 8.0205283631275179 8.2281054593314611 -
7.2577057210307396 4.8720078675524014 -3.5569535114505899
7.2148773650816915 -3.8622927317294584 4.3487148264890587 -
2.6666493125646151 7.5413938441185078 -5.7858570673641401 -
2.7982528763570498 7.6806149622860058 -2.2797574123954822
4.6648014907661919 -7.834408287943301 -4.5785416855462637 -
4.3972916967943529 -3.2835417306301897 -3.6479762844434656
8.6042998285815901 11.68962003411772 -9.9214704355648458 -
11.828137339576868 6.4240197328845667 -2.6064943853942948
1.0620570876386366 8.3103219294024981 -6.5040351823814051 -
2.9569057552805118 3.3747547499720851 -3.2388455293347143 -
9.2334985564925329 3.1439441106918009 1.8604611410561316 -
5.3372003311468044 0.77797920312072399 5.6633758686376607
5.7397187355244554 9.0093977382220256 -0.46044197149278726 -
5.410057442850758 -4.2208306155358795 -4.9305631679389883
6.6309340159969912 3.0130771131527232 2.3193788294293576 -
0.1841056038533167 -6.101753663519065 2.9048182866319094 -
4.0645331861997311 0.047227345649930579 -1.3721859283599604
```

0.1741646502714089 0.32409525544700735 -9.687239409505068 -
2.8307103890017067 -3.9102967536252242 6.2873315492319888 -
9.3890765565919256 2.2614021131201421 6.6144566288831301 -
0.52827646495153457 -5.3417867445533416 2.0515360510656553 -
5.7024412569986804 -2.2868207301993868 4.4016041099207976
12.938655331931662;0.75218099010306094 2.0575540721540002
0.31672162625128852 2.724507122129618 3.9523104139450216 -
4.2597454178575349 3.7897660808563356 -6.3662678078695576 -
1.1453774018795062 -6.4246838827353487 -2.719088543906977 -
5.1049666978458239 5.1569510480101375 4.2691625125531347
4.0892968769425426 -8.810205806965083 -12.168628062394193
0.15335615444923495 -5.7953156736015945 -10.196169589037565 -
2.6447306056919824 -2.9742486072262997 0.36936276106814103
5.8311551259473413 -0.62361620634111015 -1.9240116683709474 -
1.6513589782643188 11.714526520157921 -5.3743094628503565 -
4.3019951198294315 3.1670271590406709 -6.5083915121236977 -
3.4565383795466036 4.4590041783762278 -1.4648171136397048 -
4.3085241694249152 -3.2910661368772871 6.0826467918474751 -
4.3072862682999302 1.0979857215364777 -7.1955253675949011
7.6752705718484444 5.3014229349633233 -11.404753347177413 -
0.53770402435919162 -2.5600181726231135 2.8874574376756739
1.9385987176650761 5.8657122652012088 5.0463797022400758
1.4846708243169622 -3.49428139932771 -6.2736562574074854
5.3792021110682828 7.9242742154861334 -1.5532528988325636
6.7346501515420467 -1.7437671892197342 -4.3338491486259079
6.4194632721275164 1.8097079303289112 -8.0947239762207293
11.518417402894958 9.2877892617419935 -0.93738566029601811 -
6.1356377461936722 3.2732104438815637 -8.700404718317774 -
3.4549359667865023 -3.407027404254336 -5.3147507902743429
2.8579967711548706 5.3343797691216599 9.0424744186704498
2.6856185790480587 -7.4818060468970975 2.245781004468733 -
9.5548659559251181 3.5199561219637094 3.1454607086918647 -
1.8548316057668188 2.4170992302449905 4.5397485607362267
1.3393634999601993 0.94539846856858389 1.5370779659587972
6.1874438696280825 0.64347604515818002 2.8018745943907768 -
1.1080268816773422 6.8734507539486955 3.4944124239690808 -
3.4129953181792914 -0.023576877406620855 2.4244723605292333 -
2.0941279108483744 3.1092051853419145 -2.120078706626082 -
2.2611732629313726 -9.803906191414443;-0.97027649149240036
0.55822920974468293 2.6616988572929587 0.6514317370630035
1.6640992536402701 -2.5704197794239554 -6.652380136667122
5.7229190155059921 -1.1520204230575213 4.9545847860877963
0.254716265869673 -2.872198519751425 -1.6758507442363306
0.24729230036469718 -4.3529254796560037 9.0507042891466352 -
10.63811404292256 3.999782164753368 1.4516703785838809
10.870401588651315 -2.364666733083594 4.6732066886689001
1.4527867639693584 0.55592141592399014 3.4335496293447716 -
1.7897723766354181 4.5316924162863534 12.559363848142656 -
3.4725622921164474 -0.74854114791024107 0.68360792071239429 -
2.5332360499263364 -4.7150940591745023 2.2933047745401924 -
5.0061744020449206 9.145406503261988 -3.1632504204221927 -
3.5230593508291439 -0.38702919220172732 1.1265889984256887 -
1.3307916659229682 -1.7418801858301449 -2.8482119546935754
4.9935113995217995 3.0051980227550281 -2.3026643003603433
4.0068849226938417 2.2125998385937167 -1.8148431031334862 -
0.71442902496887317 0.93393564832727949 -5.808498540525413 -
4.0189687049462401 5.7520845278024231 3.3914974497609434 -
6.4934646522636505 -4.2978110507739773 -0.070370263487332274 -
3.1441797091897032 -0.3334182306461293 -0.044683864286662087
4.9752272809975713 -7.3860579304241503 2.5758094417334005 -
1.9165130603371581 6.3911408578241504 3.1011039719878362

```
8.0096528105104312 -1.9073325141400581 -2.1185805712255932 -
5.0195097427734119 -4.5574959825565458 -1.0092563550040403 -
2.1536853943741594 2.078596132279642 0.86936386941197652 -
5.4697493270711073 8.4298812083460604 -3.5897972990728695
2.0492725068444364 -0.25460682783427929 1.7983069170927748 -
4.6001947649110164 1.4340740075845151 -0.95951296767519645 -
1.5878914561327295 2.2851688570356798 2.9772200806422022
1.1137346614119046 -2.7582763554318759 1.1877106068224144 -
5.3849030734867993 -0.75582707943793725 1.2038072737503025
1.7862706277733158 0.70132036550209709 1.7983736979511264
3.9663399706514184 -1.5292790098291977 -2.9594699619648663];
% ===== SIMULATION ========

% Dimensions
Q = size(x1,2); % samples

% Input 1
xp1 = mapminmax_apply(x1,x1_step1);

% Layer 1
a1 = tansig_apply(repmat(b1,1,Q) + IW1_1*xp1);

% Layer 2
a2 = softmax_apply(repmat(b2,1,Q) + LW2_1*a1);

% Output 1
y1 = a2;
end

% ===== MODULE FUNCTIONS ========

% Map Minimum and Maximum Input Processing Function
function y = mapminmax_apply(x,settings)
y = bsxfun(@minus,x,settings.xoffset);
y = bsxfun(@times,y,settings.gain);
y = bsxfun(@plus,y,settings.ymin);
end

% Competitive Soft Transfer Function
function a = softmax_apply(n,~)
if isa(n,'gpuArray')
    a = iSoftmaxApplyGPU(n);
else
    a = iSoftmaxApplyCPU(n);
end
end
function a = iSoftmaxApplyCPU(n)
nmax = max(n,[],1);
n = bsxfun(@minus,n,nmax);
numerator = exp(n);
denominator = sum(numerator,1);
denominator(denominator == 0) = 1;
a = bsxfun(@rdivide,numerator,denominator);
end
function a = iSoftmaxApplyGPU(n)
nmax = max(n,[],1);
numerator = arrayfun(@iSoftmaxApplyGPUHelper1,n,nmax);
denominator = sum(numerator,1);
a = arrayfun(@iSoftmaxApplyGPUHelper2,numerator,denominator);
end
```

```
function numerator = iSoftmaxApplyGPUHelper1(n,nmax)
numerator = exp(n - nmax);
end
function a = iSoftmaxApplyGPUHelper2(numerator,denominator)
if (denominator == 0)
    a = numerator;
else
    a = numerator ./ denominator;
end
end

% Sigmoid Symmetric Transfer Function
function a = tansig_apply(n,~)
a = 2 ./ (1 + exp(-2*n)) - 1;
end
```

Skript v softwaru Matlab volající tuto funkci a vykreslující potřebné grafy závislostí. Tento skript také generoval náhodná data pro naučení, kterých je příliš mnoho, aby zde byly vypisovány.

```
clc, clear all;
n = 5000;
x(1,:) = ones(1,n).*300 + rand(1,n)*0.2 - rand(1,n)*0.2;
x(2,:) = ones(1,n).*59.985 + rand(1,n)*0.05 - rand(1,n)*0.05;
x(3,:) = ones(1,n).*0.003 + rand(1,n)*0.002 - rand(1,n)*0.002;
for j=1:n
    if j<(n/2)
        x(4,j) = 1.6;
    elseif j<(4*n/3)
        x(4,j) = 3.2;
    else
        x(4,j) = 6.4;
    end
end
y = zeros(3,n);
for i=1:n
    if x(1,i)<299.9 || x(2,i)<59.97 || x(3,i)>0.004 || x(4,i)>3.2
        y(1,i) = 1; %zmetky
    elseif x(1,i)>=299.9 && x(1,i)<=300.1 && x(2,i)<=60.03 &&
x(2,i)>=59.97 && x(3,i)<=0.004 && x(4,i)<=3.2
        y(2,i) = 1; %ok
    else
        y(3,i) = 1; %opravitelne
    end
end
y2 = round(class_acc(x));
y3 = round(classification(x));
a=0;
e=0;
for k=1:n
    if isequal(y(:,k),y2(:,k))==1
    else
        e = e+1;
    end
    if isequal(y(:,k),y3(:,k))==1
    else
        a = a+1;
    end
end
```

Příloha 4

Funkce *clustering* generovaná naučenou neuronovou sítí v softwaru Matlab pro příklad klasifikace v kapitole 5.4.1

```
function [y1] = clustering(x1)
%clustering neural network simulation function.
%
% Generated by Neural Network Toolbox function genFunction, 15-May-
2017 09:57:20.
%
% [y1] = clustering(x1) takes these arguments:
%   x = 2xQ matrix, input #1
% and returns:
%   y = 4xQ matrix, output #1
% where Q is the number of samples.

%#ok<*RPMT0>

% ===== NEURAL NETWORK CONSTANTS =====

% Layer 1
IW1_1 = [-2.0241632086759722 -1.930973393138496;-1.809878105952164
2.1184099492173472;1.9145075856864748 -
2.0757326913997289;2.1311005034238777 2.0275946688499951];

% ===== SIMULATION ========

% Input 1
% no processing

% Layer 1
z1 = negdist_apply(IW1_1,x1);
a1 = compet_apply(z1);

% Output 1
y1 = a1;
end

% ===== MODULE FUNCTIONS ========

% Negative Distance Weight Function
function z = negdist_apply(w,p,~)
[S,R] = size(w);
Q = size(p,2);
if isa(w,'gpuArray')
    z = iNegDistApplyGPU(w,p,R,S,Q);
else
    z = iNegDistApplyCPU(w,p,S,Q);
end
end
function z = iNegDistApplyCPU(w,p,S,Q)
z = zeros(S,Q);
if (Q<S)
    pt = p';
    for q=1:Q
        z(:,q) = sum(bsxfun(@minus,w,pt(q,:)).^2,2);
    end
else
```

```
    wt = w';
    for i=1:S
        z(i,:) = sum(bsxfun(@minus,wt(:,i),p).^2,1);
    end
end
z = -sqrt(z);
end
function z = iNegDistApplyGPU(w,p,R,S,Q)
p = reshape(p,1,R,Q);
sd = arrayfun(@iNegDistApplyGPUHelper,w,p);
z = -sqrt(reshape(sum(sd,2),S,Q));
end
function sd = iNegDistApplyGPUHelper(w,p)
sd = (w-p) .^ 2;
end

% Competitive Transfer Function
function a = compet_apply(n,~)
if isempty(n)
    a = n;
else
    [S,Q] = size(n);
    nanInd = any(isnan(n),1);
    a = zeros(S,Q,'like',n);
    [~,maxRows] = max(n,[],1);
    onesInd = maxRows + S*(0:(Q-1));
    a(onesInd) = 1;
    a(:,nanInd) = NaN;
end
end
```

Funkce *clustering9* generovaná naučenou neuronovou sítí v softwaru Matlab pro příklad klasifikace v kapitole 5.4.1

```
function [y1] = clustering9(x1)
%clustering9 neural network simulation function.
%
% Generated by Neural Network Toolbox function genFunction, 15-May-
2017 09:58:26.
%
% [y1] = clustering9(x1) takes these arguments:
%   x = 2xQ matrix, input #1
% and returns:
%   y = 9xQ matrix, output #1
% where Q is the number of samples.

%#ok<*RPMT0>

% ===== NEURAL NETWORK CONSTANTS =====

% Layer 1
IW1_1 = [-2.0152337564033767 -1.8926408204462537;-1.8325513984910224
0.12920102318062043;-1.8250761336292847
2.2333110746956137;0.10702160867070512 -
2.0081217907073814;0.070527299211460726
1.9946450808890608;0.070527299211460726
1.9946450808890608;1.9466432083532692 -
1.8758323636454466;2.0101003063731882 -
0.14267773299894138;2.1105461658878495 1.93257794503453];
```

```
% ===== SIMULATION ========

% Input 1
% no processing

% Layer 1
z1 = negdist_apply(IW1_1,x1);
a1 = compet_apply(z1);

% Output 1
y1 = a1;
end

% ===== MODULE FUNCTIONS ========

% Negative Distance Weight Function
function z = negdist_apply(w,p,~)
[S,R] = size(w);
Q = size(p,2);
if isa(w,'gpuArray')
    z = iNegDistApplyGPU(w,p,R,S,Q);
else
    z = iNegDistApplyCPU(w,p,S,Q);
end
end
function z = iNegDistApplyCPU(w,p,S,Q)
z = zeros(S,Q);
if (Q<S)
    pt = p';
    for q=1:Q
        z(:,q) = sum(bsxfun(@minus,w,pt(q,:)).^2,2);
    end
else
    wt = w';
    for i=1:S
        z(i,:) = sum(bsxfun(@minus,wt(:,i),p).^2,1);
    end
end
z = -sqrt(z);
end
function z = iNegDistApplyGPU(w,p,R,S,Q)
p = reshape(p,1,R,Q);
sd = arrayfun(@iNegDistApplyGPUHelper,w,p);
z = -sqrt(reshape(sum(sd,2),S,Q));
end
function sd = iNegDistApplyGPUHelper(w,p)
sd = (w-p) .^ 2;
end

% Competitive Transfer Function
function a = compet_apply(n,~)
if isempty(n)
    a = n;
else
    [S,Q] = size(n);
    nanInd = any(isnan(n),1);
    a = zeros(S,Q,'like',n);
    [~,maxRows] = max(n,[],1);
    onesInd = maxRows + S*(0:(Q-1));
    a(onesInd) = 1;
```

```
    a(:,nanInd) = NaN;
end
end
```

Skript v softwaru Matlab volající tyto funkce a vykreslující potřebné grafy závislostí. Tento skript také generoval náhodná data pro naučení, kterých je příliš mnoho, aby zde byly vypisovány.

```
clear all, clc;
x1 = 2*ones(1,10) + [rand(1,10) - rand(1,10)];
y1 = 2*ones(1,10) + [rand(1,10) - rand(1,10)];

x2 = -2*ones(1,10) + [rand(1,10) - rand(1,10)];
y2 = 2*ones(1,10) + [rand(1,10) - rand(1,10)];

x3 = -2*ones(1,10) + [rand(1,10) - rand(1,10)];
y3 = -2*ones(1,10) + [rand(1,10) - rand(1,10)];

x4 = 2*ones(1,10) + [rand(1,10) - rand(1,10)];
y4 = -2*ones(1,10) + [rand(1,10) - rand(1,10)];

plot(x1,y1,'g+',x2,y2,'r+',x3,y3,'b+',x4,y4,'m+')
title('Shlukování')
xlabel('x')
ylabel('y')
legend('Shluk 1','Shluk 2','Shluk 3','Shluk 4')
grid on
input = [x1 x2 x3 x4; y1 y2 y3 y4];
```