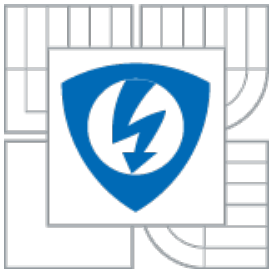
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV MIKROELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF MICROELECTRONICS

VYHODNOCOVÁNÍ ELEKTROCHEMICKÝCH SIGNÁLŮ NEURONOVOU SÍTÍ

RECOGNITION OF ELECTROCHEMICAL SIGNALS USING ARTIFICIAL NEURAL NETWORK

DIPLOMOVÁ PRÁCE
MASTER´S THESIS

AUTOR PRÁCE — BC. JAN ŠÍLENÝ
AUTHOR

VEDOUCÍ PRÁCE — DOC. ING. JAROMÍR HUBÁLEK, PH.D.
SUPERVISOR

BRNO 2011

Abstrakt:

Automatizovaná elektrochemická měření jsou zdrojem velkého množství dat určených k následnému vyhodnocování. Tato práce se zabývá problémem zpracování, klasifikace a vyhodnocování elektrochemických signálů pomocí neuronových sítí. Kvůli vysoké dimenzionalitě analyzovaných dat je v této práci využita autoasociativní neuronová síť (AANN). Tento typ sítě provádí redukci dimenzionality filtrováním analyzovaných dat a extrahuje relativně nízký počet význačných parametrů na výstupu svého krčku. Pomocí extrahovaných parametrů je možné provést klasifikaci, vyhodnocení a následně modelovat analyzovaný experiment díky rekonstrukční části naučené sítě. Dále se tato práce zabývá implementací dopředných neuronových sítí v jazyku OpenCL.

Abstract:

Automatical electrochemical measurements are sources of large data sets intended for further analysis. This work deals with classification, evaluation and processing of electrochemical signals using artificial neural networks. Due to high dimensionality of input data, an autoassociative neural network (AANN) is used in this work. This type of network performs dimensionality reduction via filtering the input data into relatively small number of principal parameters at the bottleneck output. These extracted parameters can be used for classification, evaluation and additional modelling of analyzed data trough the reconstructive part of this network. Furthermore, this work deals with implementation of a feedforward neural network in OpenCL language.

Klíčová slova:

neuronové sítě, klasifikace dat, elektrochemické signály, GPU programování, OpenCL, NLPCA

Keywords:

artificial neural networks, data classification, electrochemical signals, GPU programming, OpenCL, NLPCA

Bibliografická citace díla:

ŠÍLENÝ, J. *Vyhodnocování elektrochemických signálů neuronovou sítí.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2011. 65 s. Vedoucí diplomové práce doc. Ing. Jaromír Hubálek, Ph.D..

Prohlášení autora o původnosti díla:

Prohlašuji, že jsem tuto vysokoškolskou kvalifikační práci vypracoval samostatně pod vedením vedoucího diplomové práce, s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury. Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne 26. 5. 2011

…….

Poděkování:

Hlavní poděkování patří mému vedoucímu doc. Ing. Jaromíru Hubálkovi, Ph.D. za trpělivost a důvěru, kterou ve mne vložil a také za volnost, kterou mi poskytl při realizaci této práce. Dále bych chtěl poděkovat celému týmu z Laboratoře molekulární biochemie a biolelektrochemie z Mendelovy zemědělské a lesnické univerzity v Brně, kde jsem tuto práci začal zpracovávat. Další díky patří mé rodině, která mi byla vždy oporou jak v osobním, tak profesním životě. Nakonec děkuji všem, kteří byli ochotni naslouchat problémům, které jsem řešil v průběhu realizace této práce.

Obsah

Úvod

V dnešní době jsou datové analýzy jednou z neopominutelných oblastí vědy. S rozvíjející se výpočetní technikou navíc dochází k nasazování stále složitějších metod, které by jinak nebylo možné efektivně využít. Jednou z těchto metod jsou i neuronové sítě. Přestože se jedná o relativně starou techniku, její využití získalo velký význam právě s nástupem výkonné výpočetní techniky. V dnešní době je tak většina experimentů doprovázena právě rozborem analyzovaných dat pomocí neuronových sítí. Využití této metodiky umožňuje získat další vhled do vnitřních jevů experimentu, provádět klastrování či vyhodnocování vybraných sledovaných parametrů. S ohledem na tyto skutečnosti se v této práci budeme zabývat rozborem elektrochemických dat. Díky své miniaturizovatelnosti a související možností nasazení těchto metod v terénu nebo přímo u koncového zákazníka, má oblast elektrochemických analýz velkou výhodu oproti dalším metodám. Realizace miniaturních elektrod na jedno použití a analyzátorů, které jsou snadno přenosné, je metodika využití jasně definována. Oproti ostatním metodám se v elektrochemii bohužel mnohdy setkáváme s nepřesnostmi a nedostatečnou definicí sledovaných parametrů ve změřených datech. Otevírá se zde tedy prostor pro navržení matematických modelů, mezi nimi i neuronových sítí, které tyto nedostatky odstraní nebo naopak odkryjí jejich podstatu. Proto dalším z cílů této práce bude výběr vhodného modelu neuronové sítě a následná podrobná analýza dat.

Množství nástrojů, které se využívají v souvislosti s analýzou dat neuronovými sítěmi, je velmi velké. Existují samozřejmě jak komerční nástroje, tak otevřené softwarové celky. Jednotlivé možnosti se pokusíme v této práci okrajově jmenovat a následně provedeme návrh vlastního programu na zpracování dat s maximálním ohledem na rychlost a dnešní dostupné technologie. Proto v teorii rozebereme OpenCL jako jednu z programovacích technologií budoucnosti, v průběhu práce pak bude implementován model neuronové sítě právě pomocí této technologie.

1 Zpracování elektrochemických signálů

Zdrojem dat v elektrochemických analýzách jsou převážně potenciostaty nebo chromatografy. [1] [2] Dnes jsou tyto přístroje již téměř výhradně připojeny k PC a ovládány z příslušného programového rozhraní, ve kterém je možno výstupní data přímo sledovat a ručně analyzovat. Manuální zpracování je však časově velmi náročné, v případě automatickéch měření velkého množství vzorků je navíc nevhodné. V takovém případě je nutné využít jiných specializovaných vyhodnocovacích nástrojů.

1.1 Možnosti zpracování a klasifikace dat

Ke zpracování dat jsou většinou využívány některé z dotupných nástrojů, kterými jsou např:

- GAUSS - Aptech Systems Inc.
- Maple - Waterloo Maple Software Inc.
- Mathematica - Wolfram Research Inc.
- Matlab -The Mathworks Inc.
- O-Matrix - Harmonic Software
- OxMetrics(Ox Prof.) - Timberlake Consultants Ltd.
- Scilab – INRIA
- Octave – GNU

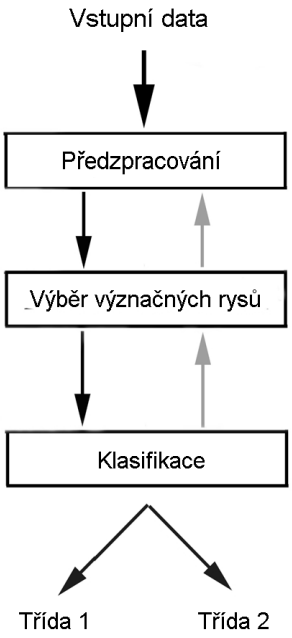
Další variantou je vývoj vlastní specializované aplikace vhodné pro řešení zadaného problému. V případě simulací a testování je obvykle výhodnější a rychlejší využít některý z uvedených nástrojů. Každý nástroj však má jiné možnosti, takže výsledný výběr je nakonec určen klasifikační metodou, kterou chceme na analýzu využít.

Díky rozvíjejícím se oborům a činostem jako je bioinformatika, data mining, rozpoznávaní řeči nebo obrazu, automatizace a dalších, je dostupné velké množství klasifikačních metod. [3] Nejdůležitější klasifikační metody jsou shrnuty v následující tabulce:

Tabulka 1: Stručný seznam a rozdělení dostupných klasifikačních metod

Založené na pravděpodobnosti	**Nezaložené na pravděpodobnosti**
Fisherův lineární diskriminant	Algoritmus k-nejbližších sousedů
Naivní Bayesův klasifikátor	Fuzzy logika
Rozhodovací stromy	Support vector automaty
Bayesovské sítě	**Neuronové sítě**
Markovovy modely	

1.2 Klasifikace

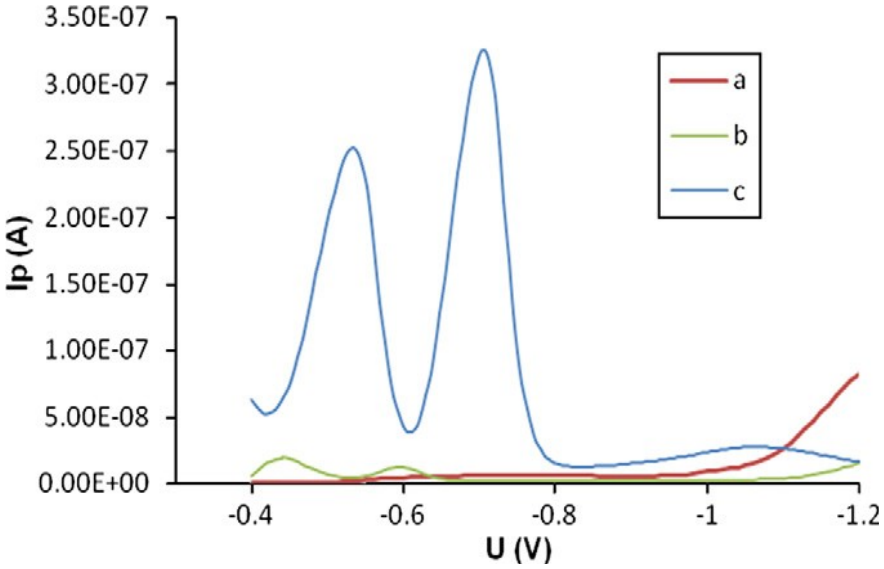


Obrázek 1: Schéma postupu při klasifikaci dat.

Základním předpokladem pro klasifikaci jsou rozdíly mezi vzorky, které chceme analyzovat. V případě, že jsou mezi vzorky rozdíly, říkáme, že mají rozdílné modely – jiné popisy, které jsou typicky vyjádřené v matematické podobě nebo je lze na matematický popis převést. Hlavním cílem v klasifikaci vzorků je odhadnout jejich třídu, tomu ale předchází několik úkonů. Obvykle je nejprve nutné vstupní data předzpracovat (eliminace rušení apod.) a následně provést výběr význačných rysů (redukce vstupní dimenzionality, odstranění zbytečných částí). [4] Na obrázku (Obrázek 1) je schéma typického postupu při klasifikaci. Jak je naznačeno, je nutné dodržovat zpětnou vazbu mezi jednotlivými kroky. Například v případě, kdy zvolíme nevhodné metody v kroku předzpracování dat, budou touto chybou zatíženy i následné kroky.

1. Vstupní data

Vstupní data jsou vždy určena prováděným experimentem, u elektrochemických signálů jsou to zpravidla závislosti *I=f(t)* nebo *I=f(E)*, viz. Obrázek 2. Většinou se jedná o přímý a nezpracovaný výstup z analyzátorů. Je tedy nezbytná prvotní vizualizace a zhodnocení dat operátorem. Křivky se obvykle skládají z velkého množství bodů, proto jsou nutné další úpravy viz. následující bod. [5]



Obrázek 2: Ukázková data z voltametrických měření

2. Předzpracování

   Předzpracování úzce souvisí se vstupními daty a jejich úpravou. V tomto bodě jsou využívány operace typu převzorkování, transformace do jiných rovin (např. fourierova transformace), ořezy, vyhlazování atd. V případě elektrochemických dat se jedná nejčastěji o vyhlazení průběhů (smooth), korekci základní linie (baseline correction) a jiné úpravy při kterých nedochází ke ztrátě důležitých informací.

3. **Výběr charakteristických rysů**

   Z každého analyzovaného vzorku jsou obvylke vybírány charakteristické rysy nebo hlavní příznaky tak, aby byl co nejvíce zredukován soubor analyzovaných dat. Tento krok úzce souvisí s redukcí dimenzionality vstupních dat pro klasifikátor a s tzv. prokletím dimenzionality.[6] Navíc narůstají paměťové nároky a čas potřebný k provedení analýzy. Tento krok nejvíce závisí na vybrané klasifikační metodě, některé metody totiž mohou provádějí redukci dat samy o sobě.

4. **Klasifikátor**

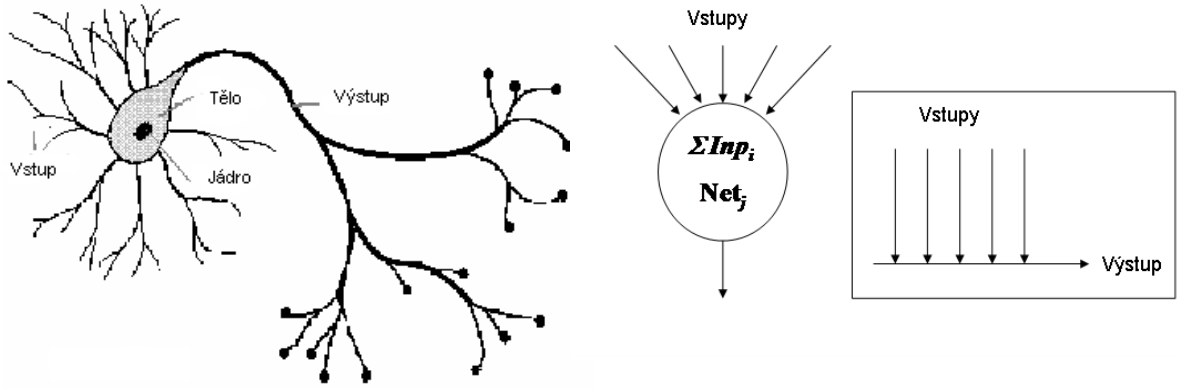
   Hlavním účelem klasifikátoru je samotné zpracování připravených analyzovaných dat. Jeho výstupem může být zařazení vzorků do tříd v případě klasifikace nebo jiná v experimentu sledovaná číselná hodnota při využití klasifikátoru k vyhodnocení parametrů. Nutno dodat, že klasifikace je velmi úzce spjata právě s vyhodnocováním. V případě, že jsou výstupní hodnoty dostatečně jemně ocejchovány (např. velký počet výstupních tříd), lze touto metodou pomocí klasifikace provést vyhodnocení.

Jak již bylo uvedeno, volbou vhodné klasifikační metody lze eliminovat některé z výše uvedených kroků a zjednodušit tak celý proces. Z tohoto pohledu jsou velmi silným nástrojem neuronové sítě. Existuje mnoho typů neuronových sítí, které automaticky provádějí některé z výše uvedených úkonů. V následující kapitole se tedy hlouběji seznámíme s principem jejich funkce a základním systémem jejich učení.

2 Umělé neuronové sítě

2.1 Úvod

Neuronové sítě nebo umělé neuronové sítě („ANN“ z ang. „artificial“ neural networks) byly zpočátku vyvíjeny se záměrem modelovat procesy v lidském mozku, jakými jsou například učení a paměť. Přestože existuje jistá podoba mezi ANN a mozkem, je patrné, že umělé neurony jsou pouhým zjednodušením [7] skutečných biologických neuronů viz. Obrázek 3. Vývoj ANN následně vedl k rozvoji praktických využití v nejrůznějších vědních oborech. Neuronové sítě mají dnes velmi široké uplatnění. Využívají se například při aproximaci funkcí, regresních analýzách, predikci , klasifikaci a zpracování dat. [8] [9] [10]

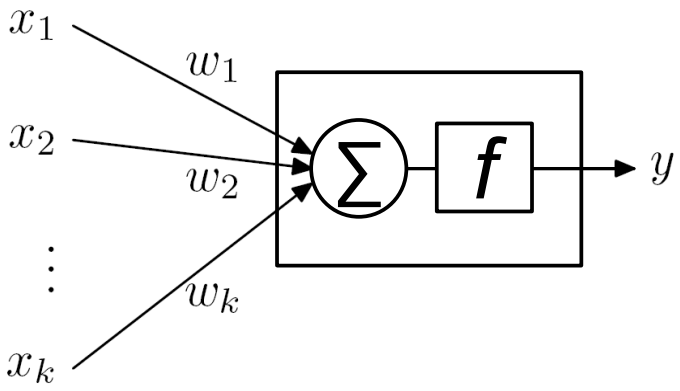


Obrázek 3: Obrázek biologického neuronu (vlevo) a zjednodušený umělý model biologického neuronu (vpravo).

2.2 Základní model neuronu - perceptron

Perceptron byl původně navržen jako pravděpodobnostní model pro objasnění dějů probíhajících v mozku, mezi které patří paměť, úsudek, zjednodušování a přemýšlení. Tento problém se týkal také senzorického vnímání biologických systémů, formem ukládání informací a způsobů, jakými uložená informace ovlivňuje vybavování. [11] Perceptron se tak stal základním stavebním kamenem neuronových sítí. Jejich nejjednodušší formou jsou takzvané jedno-jednotkové perceptrony viz. Obrázek 4. Takovéto zapojení lze využít pouze pro řešení nejjednodušších problémů, které lze lineárně separovat. Perceptron provádí v podstatě váženou sumu vstupních dat a její následnou transformaci do jednorozměrného prostoru. Složitější sítě jsou pak obvykle tvořené sériovým a paralelním zapojováním těchto jednotek do vrstev a mohou provádět složitější transformace. [12] Nejprve se však seznámíme se vztahy a funkcemi, které popisují jednoduchý uzel.

2.2.1 Základní princip perceptronu



Obrázek 4: Jedno-jednotkový perceptron.

Tyto jednotky se sestávají z K vstupních hodnot $x_1, \ldots, x_K$ a výstupní hodnoty y. Vstupní hodnoty x_k jsou váhovány vahami ω_i $(i = 1, \ldots, k)$ tak, že výstupní předpokládaná odezva y je určena vstupním vektorem proměnných $x = (x_1, \ldots, x_K)$ a to dle vztahu [13]:

$$y = \omega_0 + \sum_{i=1}^{k} x_i * \omega_i = \omega_0 + \boldsymbol{x}'\boldsymbol{\omega} \tag{1}$$

Jedná se tedy o jednoduchý matematický model reprezentující váženou sumaci vstupních hodnot. Perceptrony jako takové jsou hluboce spojeny s regresní a diskriminační analýzou. Nutno zmínit, že výstup nemusí být pouze váženým součtem vstupů, ale může být ještě následně transformován. Dostupné transformace budou popsány v následující podkapitole. V případě jednoduchého perceptronu se obvykle využívá prahová přenosová funkce 'hardlim'. Vztah mezi vstupem a výstupem lze potom obecněji zapsat jako:

$$y = f(\omega_0 + \boldsymbol{x}'\boldsymbol{\omega}) \tag{2}$$

Kde $f(x,\omega)$ je předem definovaná funkce nazývaná také aktivační či přenosová funkce.

Předpokládejme nyní, že máme k dispozici soubor analyzovaných dat, které chceme zařadit do dvou skupin. Prahová přenosová funkce s výstupy *[-1; 1]* je pro tento problém postačující. Uvažovaná vstupní data nechť jsou párována s požadovanými výstupy $((x_1, y_1), \ldots, (x_n, y_n))$ do n párů, jedná se o takzvanou trénovací sadu. Dále předpokládejme, že po inicializaci perceptronu jsou v sadě dat chybně klasifikované páry. Abychom mohli upravit váhy a eliminovat chyně zařazené vzorky, potřebujeme navrhnout chybovou funkci. Nejjednodušší používaná funkce se nazývá kritériem perceptronu [14] [15] [16] a má tvar:

$$E(\omega) = -\sum_{i \in \text{chybné}} f(\omega_0 + \boldsymbol{x_i}'\boldsymbol{\omega}) * y_i \tag{3}$$

Tato chyba je vyjádřením celkové chyby, takže abychom nalezli ideální váhový vektor $\boldsymbol{\omega}$ pro

analyzovaná data, musíme $E(\omega)$ minimalizovat.

Z kritéria (3) tak lze sestavit nejjednodušší systém na úpravu váhového vektoru, který se nazývá perceptronové pravidlo, pro chybně zařazené vzorky platí [16] [17]:

$$\Delta \boldsymbol{\omega} = \eta * y_i * \boldsymbol{x_i} \tag{4}$$

V případě využití perceptronového pravidla a prahové přenosové funkce se aplikuje opakování tohoto postupu:

- Správně zařazený vzorek: pokračujeme dalším vzorkem
- Špatně zařazený vzorek ze skupiny $y_i = 1$: přičteme vstup k váhovému vektoru
- Špatně zařazený vzorek ze skupiny $y_i = -1$: odečteme vstup od váhového vektoru

Ve vztahu (4) je navíc uvedena učící konstanta η, která váhuje vypočtenou úpravu. V případě využití spojité přenosové funkce je obvykle využívána jiná chybová funkce, jako například metoda nejmenších čtverců („LMS“ z ang. Least mean squares) [14]:

$$E(\omega) = \frac{1}{2} \sum_{i \in chybné} (f(\omega_0 + \boldsymbol{x_i}'\boldsymbol{\omega}) - y_i)^2 \quad . \tag{5}$$

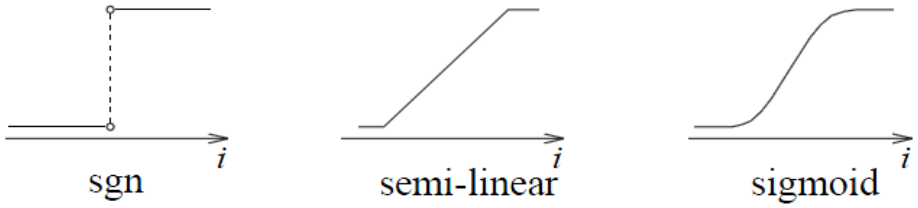
Z této chybové funkce pak vychází LMS pravidlo:

$$\Delta \boldsymbol{\omega} = -\eta (f(\omega_0 + \boldsymbol{x_i}'\boldsymbol{\omega}) - y_i) * \boldsymbol{x_i} \tag{6}$$

Pokud položíme učící konstantu ve vzorci (4) $\eta = 0{,}5$ a využijeme prahovou přenosovou funkci, zjistíme, že výpočty (4) a (6) mají pro chybně zařazené vzorky stejné výsledky. V případě prahové přenosové funkce tak jde o identitu mezi perceptronovým a LMS pravidlem. Pomocí libovolného z těchto postupů získáme pro každý špatně klasifikovaný vstupní vzorek hledaný vektor úprav vah. Aplikací úpravy vah a opakováním tohoto postupu se obvykle dostáváme k požadovanému výstupu perceptronu.

2.2.2 Aktivační funkce a jejich druhy

Důležitou a zásadní částí pro složitější sítě je aktivační neboli přenosová funkce neuronu, je to funkce, která transformuje celkový vstup neuronu tak jak bylo uvedeno ve vztahu (2). Mimo speciální případy jsou aktivační funkce většinou neklesající. Obrázek 5 zobrazuje nejčastěji využívané přenosové funkce [16] [18] [19].



Obrázek 5: Nejčastěji používané typy přenosových funkcí

1. **Lineární přenosová funkce - „purelin“:**

$$f(x) = x \quad . \tag{7}$$

2. **Saturovaná lineární přenosová funkce - „sat-lin“, „semi-lin“**

   Při této funkci jsou definovány dva prahy t_1 a t_2, které přiřazují vstupní hodnotě různé výstupní hodnoty. Následně jsou pro krajní intervaly stanoveny výstupní hodnoty v_1 a v_2. Výstup se pak řídí dle následujícího pravidla:

$$f(x) = \begin{Bmatrix} v_1 \, když \; x < t_1 \\ x \;\; když \; x \geq t_1 a \; x \leq t_2 \\ v_2 \;\; když \; x > t_2 \end{Bmatrix} \quad . \tag{8}$$

3. **Prahová přenosová funkce - „hardlim“ či „sgn“**

   V tomto případě je stanoven práh na 0, jsou tedy pouze dva případy do kterých může výstup spadat. Výstupy jsou voleny buď symetrické nebo binární. Symetrická:

$$f(x) = \begin{Bmatrix} -1 \, když \; x < 0 \\ 1 \;\; když \; x \geq 0 \end{Bmatrix} \tag{9}$$

   a nesymetrická:

$$f(x) = \begin{Bmatrix} 0 \, když \; x < 0 \\ 1 \;\; když \; x \geq 0 \end{Bmatrix} \tag{10}$$

4. **Sigmoidní přenosová funkce - „logsig“, „tansig“**

   Logsig:

$$f(x) = \frac{1}{1+e^{-ax}} \tag{11}$$

   Tansig:

$$f(x) = \frac{e^{ax}-e^{-ax}}{e^{ax}+e^{-ax}} \tag{12}$$

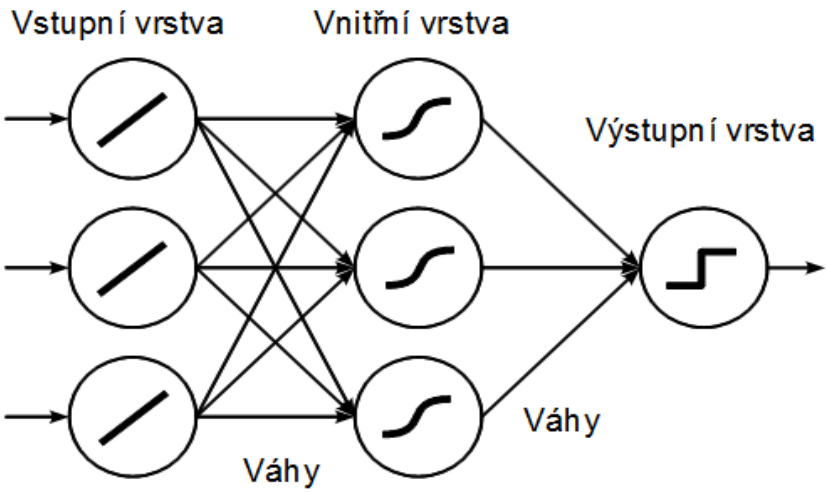
   Parametr *a* je většinou volen *a = 1*.

2.3 Topologie sítí

Ze základních vlastností uzlu plyne, že je možné sestavit spojováním a vrstvením rozsáhlejší propojené sítě neuronů. Návrh topologie sítě se obvykle liší dle analyzovaných dat, požadovaného výstupu a výpočetních možností dnešního hardware. Vzory spojení jednotek souvisí také s výslednou transformací dat a učením. Typy zapojení jednotek lze nejjednodušeji rodělit na dvě skupiny [17] [20]:

2.3.1 Feed-forward sítě (plněné směrem vpřed, dopředné)

Jedná se o sítě, kde data na vstupu jsou striktně přenášena směrem na výstup. Data mohou být transformována přes několik (vrstev) jednotek, ovšem s absencí jakékoliv zpětné vazby. Žádná data tedy nejsou přenášena z výstupu jednotek na vstupy jednotek ve stejné vrstvě nebo jednotek ve vrstvách předcházejících. Klasickými typy dopředných sítí jsou například Perceptron a AdaLiNe v jedno i vícecevrstvé podobě.

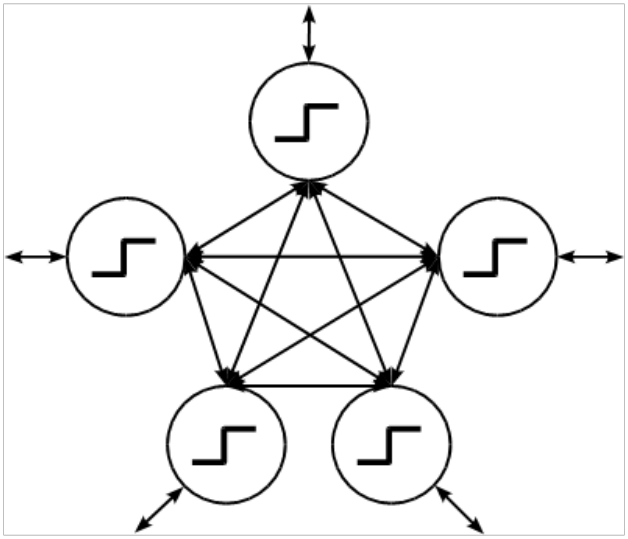


Obrázek 6: Jednoduché schéma zapojení třívrstvé feed-forward neuronové sítě.

Jak je naznačeno, vrstvy se mohou skládat z neuronů s různými přenosovými funkcemi. V uvedeném případě je pomocí všeobecně využívaných symbolů naznačena lineární přenosová funkce vstupní vrstvy, sigmoidální přenosová funkce vnitřní skryté vrstvy a prahová výstupní funkce.

2.3.2 Rekurentní sítě

Rekurentní sítě obsahují zpětnovazebná spojení mezi všemi neurony (Obrázek 7). V těchto sítích hrají nejdůležitější roli dynamické vlastnosti, v některých případech u aktivačních hodnot těchto jednotek probíhají různé relaxační děje, dokud nenastane stabilní stav a tyto hodnoty se neustálí. V jiných případech jsou změny aktivačních hodnot výstupních neuronů tak významné, že právě toto dynamické chování určuje výstup sítě. Obecně jsou tyto sítě složitější na návrh i učení. Typickým případem této třídy jsou pak Andersonovy, Kohonenovy a Hopfieldovy sítě. Kohonenovy sítě se obvykle skládají z neuronů s prahovou přenosovou funkcí. Všechny neurony v síti jsou mezi sebou propojeny a všechny slouží zároveň jako vstupní a výstupní. Tato síť principielně funguje jako autoasociativní a při přiložení vstupů se snaží optimalizovat vnitřní váhy tak, aby byl výstup identický [17].



Obrázek 7: Typické schéma Kohonenovy sítě

2.4 Trénování neuronových sítí

Neuronové sítě musejí být nastaveny tak, aby se po přiložení vstupních dat na výstupech objevily požadované hodnoty. Existuje několik metod používaných pro nastavení a úpravy jednotlivých vah trénované sítě. Je možné nastavit váhy explicitně, zde se vychází ze znalosti analyzovaného problému. Další možností je využití učení neuronové sítě – plnit její vstupy sadou trénovacích dat a nechat měnit váhy mezineuronových propojení podle předem určených pravidel. V bližším pohledu lze učení rozdělit do dvou základních kategorií [17], kterými jsou:

1. Řízené učení neboli asociativní učení – síť je trénována poskytnutím sady vstupních dat a sadou odpovídajících výstupních dat. Tyto vstupně-výstupní páry jsou poskytovány externím „učitelem“ nebo systémem, který síť obsahuje (samořízené).
2. Neřízené učení nebo samoorganizační učení – výstupní jednotky jsou trénovány k reakci na vzory ve vstupech. V této formě učení se předpokládá, že si systém najde ve vstupních datech význačné znaky. Oproti řízenému učení zde neexistuje sada výstupních vzorů, do kterých by vstupy měly být klasifikovány – systém si vytvoří vlastní výstupní reprezentaci vstupů.

2.4.1 Modifikace vah mezineuronových spojení

Při obou typech učení uvedených v předchozí podklapitole je nutné provádět modifikace vah mezi jednotlivými uzly dle předem zvolených pravidel. Všechna pravidla mohou být považována za modifikaci Hebbianského učení. Základní představou je následující

úvaha, když jsou dvě jednotky současně aktivní, musí být jejich propojení zesíleno. Pokud jednotka *j* obdrží vstupní impuls od jednotky *k*, popisuje Hebbianské učení modifikaci váhy ω_{jk} jako [21]:

$$\Delta \omega_{jk} = \gamma x_i y_j \tag{13}$$

kde γ je pozitivní konstanta úměrnosti reprezentující míru učení. Další běžné pravidlo nevyužívá aktuální výstup neuronu, nýbrž rozdíl mezi aktuálním a požadovaným výstupem, což vede k úpravě vah dle [22]:

$$\Delta \omega_{jk} = \gamma (t_j - y_j) x_k \tag{14}$$

kde t_k je požadovaný výstup poskytnutý učitelem. Toto pravidlo je často nazýváno „Widrow-Hoffovo“ pravidlo nebo „Delta pravidlo“. Je nutné dodat, že uvedený vzorec (14) je platný pro neurony s lineární přenosovou funkcí, pro nelineární je nutné toto pravidlo dále rozšířit. Zároveň se jedná o nejjednodušší možný zápis. Delta pravidlo je navíc pouze speciálním případem zpětné propagace [23].

2.4.2 Zpětná propagace

V jednoduché jednovrstvé síti, která obsahuje neurony s lineární přenosovou funkcí popisuje výstup každého neuronu vzorec (1). Takto jednoduchá síť je schopna reprezentovat libovolný lineární vztah mezi hodnotami vstupů a výstupů. Prahováním výstupních hodnot lze pak jednoduše zkonstruovat jednoduchý klasifikátor. V případě složitějších sítí platí, že pokud jsou vstupní data n-dimenzionální a požadovaný výstup m-dimenzionální, reprezentuje výsledná síť transformaci z n-dimenzionálního prostoru do m-dimenzionálního. Chceme-li tedy naučit síť provádět transformaci vzorků z trénovacích vstupních dat $\boldsymbol{X}$ na požadované výstupní hodnoty $\boldsymbol{T}$, budou se v průběhu učení výstupy $\boldsymbol{Y}$ pro každý vstup lišit od cílené hodnoty o $\boldsymbol{e}_p = (\boldsymbol{T}_p - \boldsymbol{Y}_p)$, kde $\boldsymbol{Y}_p$ je výstupní vektor analyzovaného vstupu *p* a $\boldsymbol{e}_p$ je vektor absolutních chyb výstupních neuronů. Modifikace vah spojů v této metodě využívá právě tyto absolutní chyby. Delta pravidlo jako takové je analogií metody nejmenších čtverců (LMS z ang. least mean squares), takže je nutné absolutní chybu výstupu ještě následně upravit na [24]:

$$E = \frac{1}{2}(y_j - t_j)^2 \tag{15}$$

$$E_p = \sum_j \frac{1}{2}(y_j - t_j)^2 \tag{16}$$

Vztah (15) popisuje chybu výstupního neuronu j pro libovolný vstupní vzor. Vztah (16) popisuje celkovou chybu *p*-tého vstupního vzoru u sítě s *j* výstupními neurony. Výstup každého neuronu ve vícevrstvé síti je vnitřním produktem jeho váhového vektoru $\boldsymbol{\omega}$ a vstupního vektoru $\boldsymbol{x}$ tak, že pro výstupní neurony dostáváme [16]:

$$y_j = f(a_j) \equiv f(\boldsymbol{\omega_j}' \boldsymbol{x}) \tag{17}$$

a pro neurony vnitřní vrstvy:

$$h_i = f(b_i) \equiv f(\boldsymbol{\omega_i}' \boldsymbol{x}) \tag{18}$$

Nyní potřebujeme výstupní chybu neuronu rozprostřít mezi jeho vstupní váhy a proporcionálně upravit jejich hodnoty. K tomu potřebujeme vypočítat, jak se jednotlivé váhy podílely na výsledné chybě. K tomu využijeme derivaci výstupní chyby vůči libovolné synapsi jdoucí do neuronu a řetízkové pravidlo. Pro výstupní neurony platí:

$$\frac{\partial E}{\partial \omega_{ij}} = \frac{\partial E}{\partial a_j} \frac{\partial a_j}{\partial \omega_{ij}} \quad . \tag{19}$$

Pro vnitřní neurony platí:

$$\frac{\partial E}{\partial \omega_{hi}} = \frac{\partial E}{\partial b_i} \frac{\partial b_i}{\partial \omega_{hi}} \quad . \tag{20}$$

První člen v rovnicích (19) a (20) reprezentuje derivaci závisející na přenosové funkci neuronu. Druhý člen závisí na vnitřním produktu $\boldsymbol{\omega_j}'\boldsymbol{x}$ respektive $\boldsymbol{\omega_i}'\boldsymbol{x}$, tyto derivace vzhledem k ω_{ij} a ω_{hi} jsou vlastně pouze vstupy sledovaných neuronů. Rozhodující je tedy první člen uvedených vzahů. Deklarujme tedy následující vztahy první části vztahu, pro výstupní neuron:

$$\delta_j = \frac{\partial E}{\partial a_j} \tag{21}$$

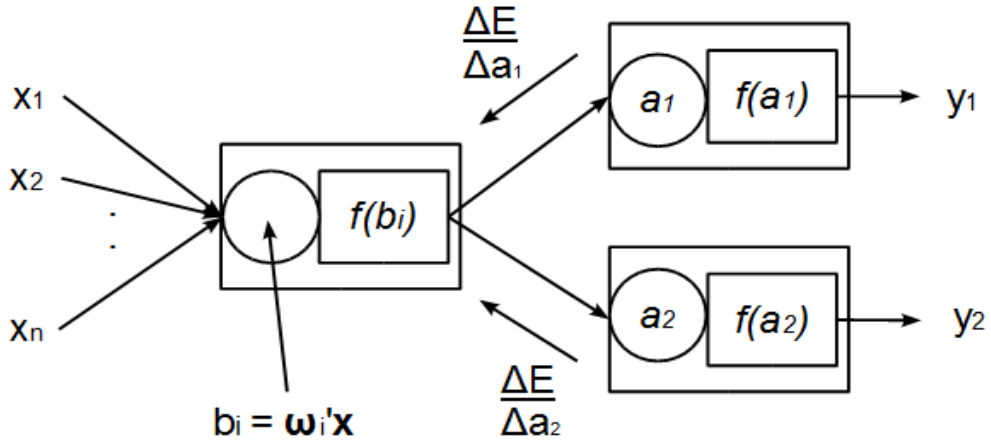
a vnitřní neuron:

$$\delta_i = \frac{\partial E}{\partial b_i} \quad . \tag{22}$$

Rozvedením těchto vztahů a využitím (17) respektive (18) získáme následující tvar:

$$\delta_j = \frac{\partial E}{\partial a_j} = f'(a_j) \frac{\partial E}{\partial y_j} = f'(a_j)(y_j - t_j) \quad . \tag{23}$$

Na následujícím obrázku je zobrazen princip průchodu chyby od výstupního neuronu k vnitřnímu.



Obrázek 8: Propagace chyb z výstupních neuronů na předchozí vrstvu

Pro vnitřní neuron je závislost poměrně komplikovanější, protože člen δ_i záleží na chybách neuronů následující (v diskutovaném případě výstupní) vrstvy, k nimž je připojen. Vyjádříme tedy chyby pomocí řetízkového pravidla a přenos součtem přes všechny váhy jdoucí do následující vrstvy, takto získáme:

$$\delta_i = \frac{\partial E}{\partial b_i} = \sum_{j=1}^{c} \frac{\partial E}{\partial a_j} \frac{\partial a_j}{\partial b_i} \quad . \tag{24}$$

První člen odpovídá v předchozí části (předchozí vrstvě) vypočtené δ_j, druhý člen odpovídá vlivu aktivační funkce na výstup a vlivu vah jdoucích ze sledovaného neuronu (tyto váhy přispívají svým přenosem k chybě na následující vrstvě). Můžeme tak psát:

$$\delta_i = f'(b_i) \sum_{j=1}^{c} \omega_{ij} \delta_j \quad . \tag{25}$$

Prozkoumáme-li podrobněji jak chyby výstupních neuronů figurují ve výpočtu chyb vnitřních neuronů, je název této metody zřejmý. Poslední částí která nám stále chybí je metoda úpravy hodnot vah. Využijeme tedy vzorec (14) , kde nahradíme člen chyby deltou vyjádřenou ve vzorcích (24) a (25). Pro vrstvu spojující vnitřní neurony s výstupními můžeme zapsat:

$$\begin{aligned} \Delta \omega_{ij} &= \eta \delta_j h_i \\ \omega_{ij}^{(t+1)} &= \omega_{ij}^{(t)} - \eta \delta_j h_i \end{aligned} \quad , \tag{26}$$

kde h_i je výstup neuronu v předchozí vrstvě spojený s aktuální modifikovanou synapsí. Pro vnitřní neurony platí:

$$\begin{aligned} \Delta \omega_{hi} &= \eta \delta_i x_h \\ \omega_{hi}^{(t+1)} &= \omega_{hi}^{(t)} - \eta \delta_i x_h \end{aligned} \quad , \tag{27}$$

kde x_h je opět výstup neuronu v předchozí vrstvě (nyní například vstupní). Procedura učení vyžaduje, aby byly změny vah proporcionální k $\partial Ep/\partial \omega$. Pro praktické účely zavádíme

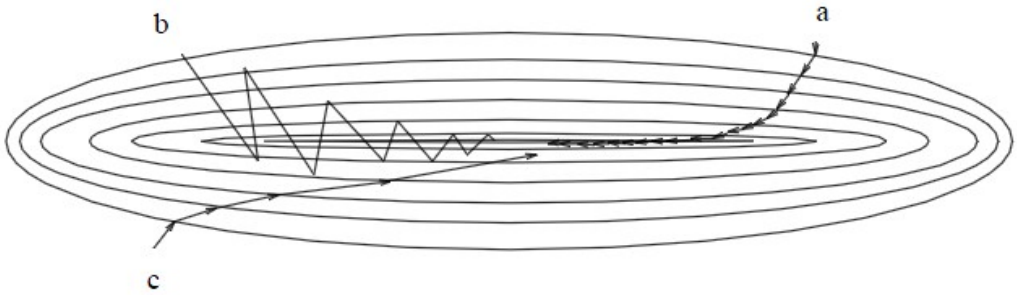
a používáme konstantu učení. Její hodnotu volíme velkou jak je jen možné, aniž by vedla ke vzniku oscilací v systému. Zpětnou propagaci lze dále ještě vylepšit zavedením členu „momentum“. Tento člen zavádí do úprav vah setrvačnost a v mnohých případech zamezí oscilacím po povrchu chyb nebo naopak dovolí uniknout z lokálního minima, modifikací vztahů (26) a (27) dostáváme:

$$\omega_{ij}^{(t+1)} = \omega_{ij}^{(t)} - \eta\,\delta_j\,h_i + \alpha\,\Delta\,\omega_{ij}^{(t)} \quad , \tag{28}$$

pro výstupní vrstvu a

$$\omega_{hi}^{(t+1)} = \omega_{hi}^{(t)} - \eta\,\delta_i\,x_h + \alpha\,\Delta\,\omega_{hi}^{(t)} \tag{29}$$

pro vrstvy vnitřní. Člen „momentum“ navíc urychluje konvergenci k optimálnímu výsledku při učení viz. Obrázek 9. Není-li použit moment, trvá učení dlouhou dobu, než je dosaženo minima. Pro velké učící konstanty nemusí být minima dosaženo nikdy kvůli oscilacím. Po přidání momentu je dosaženo minima podstatně rychleji.



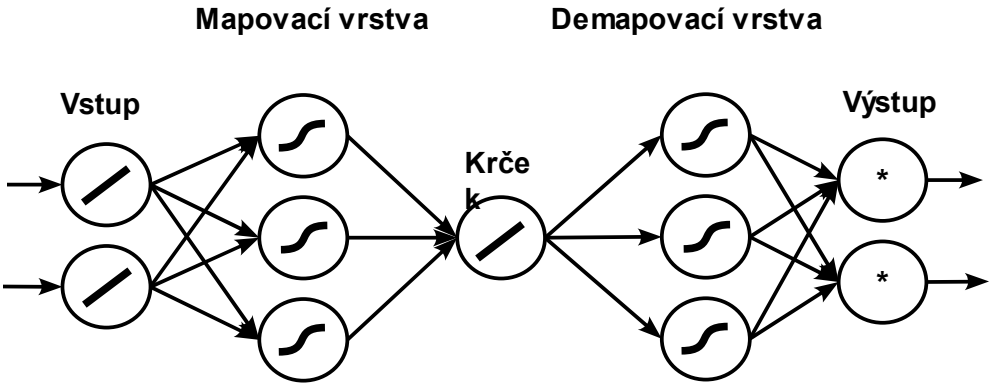
Obrázek 9: Spád do minima v chybovém prostoru. a) malé konstanty učení; b) velké konstanty učení (oscilace); c) velké učící konstanty s přidaným momentem

Uvedené vztahy předpokládají, že modifikace vah bude prováděna pro každý vstupní vzorek odděleně. Mnohdy je však výhodnější provádět tréning po sadách. Tento přístup pak v praxi při zpětné propagaci znamená, že místo vektorů pracujeme s maticemi. Například delty jsou v dávkové propagaci matice o rozměru „počet vzorků v sadě“ x „počet neuronů ve vrstvě“, stejně tak pro výstupní hodnoty neuronů a další vnitřní zpracovávané hodnoty a parametry. Maticový přístup ke zpětné propagaci přímo rozebírat nebudeme, nicméně je vhodné dodat, že maticové operace jsou ideálním prostorem pro nasazení paralelizace (GPU výpočty apod.).

2.5 Autoasociativní neuronové sítě

Autoasociativní neuronové sítě jsou obyčejné dopředné sítě trénované tak, aby provedly identické mapování vstupu na výstup, to znamená, že požadujeme po přiložení vstupního vzoru identický výstup. Učící technikou pak může být libovolné pravidlo, jako například výše zmíněná zpětná propagace. Tato síť se skládá z lichého počtu vrstev, vstupní a výstupní vrstvy, mapovací a demapovací vrstvy a neuronového krčku. Nejdůležitější vlastností této sítě je právě tento dimenzionální krček uprostřed sítě, který je obklopen

mapovací a demapovací vrstvou. Před mapovací vrstvou je obvykle vstup sítě a za demapovací výstup viz. Obrázek 10. Krček má obvykle nižší dimenzionalitu než vstupní respektive výstupní data, takže dochází ke kompresi informace. Mapovací a demapovací vrstvy naopak čítají více neuronů než vstup a výstup. Po naučení této sítě tak získáme korelační model vstupních dat, který je využitelný v mnoha dalších datových analýzách. Rozdíly mezi vstupním a výstupním signálem pak lze považovat za chybu či rušení zdroje dat, čímž lze provádět redukci šumů ve vstupních datech. Výstupní prostor dat je spojitý a velmi hladký, takže může síť predikovat správný výstup i v případě zcela chybějící či chybných vstupů. Obvyklou aplikací této sítě je nelineární analýza hlavních komponent NLPCA, která vyhledává rozložení význačných rysů analyzovaných dat na zakřivených nadrovinách. [25] [26] [27]



Obrázek 10: Schéma autoasociativní neuronové sítě s naznačenými aktivačními funkcemi. (= lineární či sigmoidní)*

2.5.1 Matematické pozadí

Uvažujme abstraktní množinu X, jejíž prvky nemají žádný specifikovaný typ. Definujme abstraktní prostor, který uvažovaná množina vybavuje superponovanými strukturami, definovanými funkcemi či algebraickými operacemi. Dále definujme metrický prostor, kterého abstraktní množina X vybavuje metrikou D [28]:

$$D : x * x' \rightarrow [0, +\infty) \quad , \tag{30}$$

kde x a x' jsou generické elementy v množině X. V metrickém prostoru pak lze nad touto množinou X zkonstruovat topologické struktury, navíc můžeme zavést pojmy jako okolí, otevřená a uzavřená podmnožina, spojitost a konvergence. Nyní nechť X a Y jsou libovolné topologické (metrické) prostory, navíc uvažujme, že jejich elementy jsou body vybrané ze spojítých množin. Definujme mezi těmito prostory homeomorfismus, zobrazení:

$$f : X \rightarrow Y \quad , \tag{31}$$

které je spojité a jeho inverze f^{-1} je také spojitá. Homeomorfismus jako takový, je

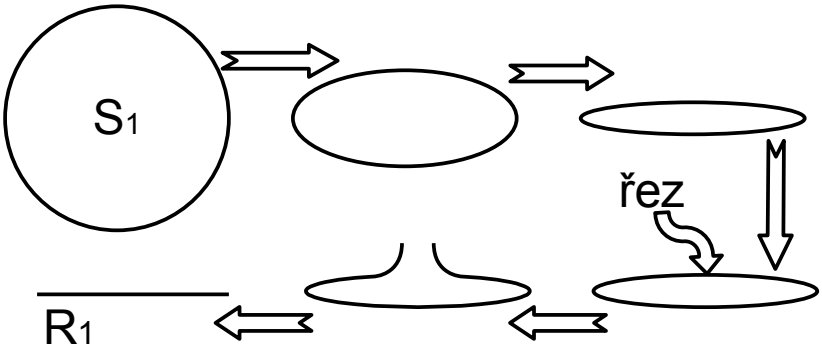
jednoznačné zobrazení mezi dvěma objekty, které mohou být transformovány jeden do druhého a naopak pouze za použití ohybů a průtahů. Žádné destruktivní operace jako řezy nejsou povoleny, pokud nedeojde po transformaci znovu ke spojení přesně tam, kde k řezu došlo. Dva topologické prostory jsou tedy homeomorfní, právě pokud existuje vztah (31).

Nechť X je libovolným topologickým prostorem. Globální popis (vzorkování) prostoru X pomocí n souřadnic je globálním homeomorfismem

$$\varphi : X \to \Re^n \quad , \tag{32}$$

kde $\Re^n$ je topologický lineární vektorový prostor. Homeomorfismus (32) přiřazuje každému elementu $x \in X$ množinu n reálných čísel $\{\varphi(x)_1, \varphi(x)_2, \ldots, \varphi(x)_n\}$, tedy zobrazení elementů $x \in X$ do souřadnic systému $\Re^n$. X budeme nazývat (nelineární) varietou, která reprezentuje zakřivený hyperpovrch, který je lokálně podobný lineárnímu (vektorovému) subprostoru $\Re^n$. Homeomorfismus zobrazení (32) nám garantuje, že přiřazené souřadnice jsou unikátní a je možné je přiřadit bez jakéhokoliv trhání elementů X od sebe. Samozřejmě existuje mnoho způsobů jak provést namapování souřadnic na uvažovaný prostor X. Uvažujme například, že homeomorfismus $\Psi : X \to \Re^m$ je jiný systém m souřadnic pro X. Pak mapování $\gamma(\Psi, \varphi) = \Psi \varphi^{-1}$ je homeomorfismem $\gamma : \Re^n \to \Re^m$.

Prostory $\Re^n$ a $\Re^m$ jsou ale homeomorfní pouze pokud $n = m$. Parametr n je tedy jednoznačně spjat s prostorem X pomocí uvedeného (lokálního) pokrytí souřadnicemi. Parametr n tedy budeme nazývat dimenzí d topologického prostoru X. Nutno dodat, že existují speciální případy prostorů X, které přestože mají dimenzionalitu $d = n$ a jsou lokálně homeomorfní vůči $\Re^n$, globální homeomorfismus vůči $\Re^n$ postrádají. Obrázek 11 zobrazuje podobný případ. Obvod kruhu, který má dimenzionalitou $d = 1$, je plochý v nejbližším okolí každého bodu a tedy lokálně homeomorfní k $\Re^1$, tato vlastnost však neplatí, pokud budeme sledovat obvod jako celek (uzavřenou křivku). Jinými slovy, každý bod na křivce můžeme jednoznačně popsat jedinou souřadnicí, polárním úhlem, globální homeomorfismus však obvod kruhu vůči $\Re^1$ nemá, protože ho není možné bez destruktivní operace (střihu) deformovat do rovné přímky. Toto platí pro libovolnou 1-dimenzionální křivku, která se protíná.



Obrázek 11: Ukázka sytému který má lokální homeomorfismus, ale postrádá globální

Podobný případ lze nalézt u dalších případů, jako například povrchu koule. Povrch koule má dle definice dimenzionalitu $d = 2$, k popisu nám v tomto případě stačí pouze sférická šířka a délka. Přesto tento povrch není globálně homeomorfní k prostoru $\Re^2$, protože ho bez použití řezu opět není možné rozvinout do roviny. Proto všechny planární projekce povrchu Země jsou nějakým způsobem zdeformované. Mercator je například deformován kolem severního pólu.

2.5.2 Geometrické vlastnosti AANN

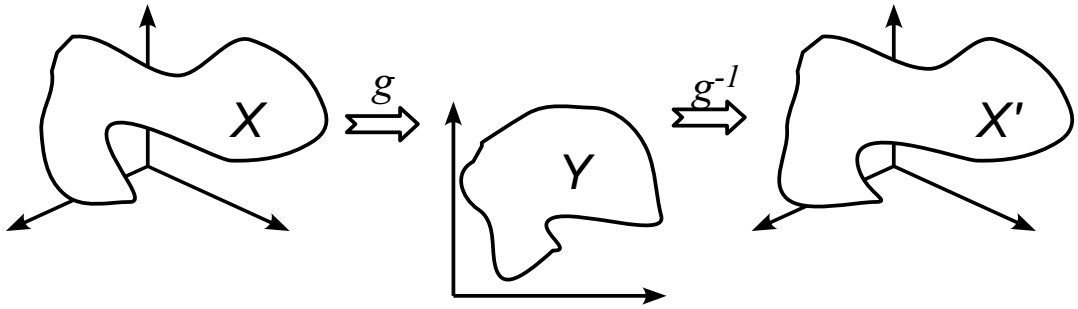
Nyní se pomocí uvedeného teoretického pozadí pokusíme popsat i vlastnosti autoasociativních neuronových sítí viz. Obrázek 10. Mimo vstupní a výstupní vrstvy obsahuje architektura tři vnitřní vrstvy. Jmenovitě mapovací a demapovací vrstvy, které obvykle obsahují stejný počet neuronů. Vstupní a výstupní vrtvy mají také stejný počet neuronů, avšak nižší než mapovací a demapovací vrstva. Aktivační funkce neuronů v mapovací a demapovací vrstvě je obvykle sigmoidální, což způsobuje nelineární charakter transformačních funkcí. Přenosová funkce neuronů v krčku je obvykle lineární. Po naučení by měla síť ideálně provádět identické mapování vstupu na výstup. Data jsou donucena projít zúženým krčkem sítě, takže pokud je nalezeno přijatelné řešení, tedy chyba rekonstruovaných dat je dostatečně malá, potom jsou na výstupu krčku všechny důležité informace o analyzovaných datech. Jinými slovy mapovací vrstva vyfiltrovala všechny důležité nelineární příznaky popisující analyzovaná data. Počet uzlů v krčku f je kritickým parametrem jak pro správné fungování sítě, tak pro přijatelnou rekonstrukci dat a je obvykle vybírán metodou pokus-omyl. Při návrhu a učení tak komplexní struktury jako AANN je nutno optimalizovat více parametrů, jako například učící konstantu, člen setrvačnosti momentum, počet uzlů v mapovací/demapovací vrstvě a šířku krčku. [28] navrhuje metodu na zjištění šířky krčku modelováním fraktální dimenze analyzovaných dat.

Nyní se podívejme na vnitřní vazby sítě z matematického pohledu. Nechť $X \subset \Re^m$ je topologickým prostorem vstupních a výstupních dat a $Y \subset \Re^f$ topologickým prostorem dat na výstupu neuronů v krčku sítě. Podle teorému Cybenka, mohou diskrétní sumy sigmoidních funkcí tvaru:

$$\sum_{j=0}^{N} \alpha_j \sigma \left(\sum_{i=1}^{m} \omega_{ji} x_i + \omega_{j0} \right) \quad , \tag{33}$$

vyjádřit libovolnou nelineární spojitou vektorovou funkci tvaru $\vec{v} = \vec{g}(\vec{x})$, kde $\vec{x} \in \Re^m$ a $\vec{v} \in \Re^n$ pro libovolná m a n. Standartní dopřednou neuronovou síť lze samozřejmě vyjádřit pomocí podobných součtů jako uvádí vzorec (33). Neuronová síť tak provádí spojité mapování ze vstupního prostoru do výstupního. Toto samozřejmě platí pokud je počet neuronů vnitřních vrstev N vhodně zvolen. V případě AANN můžeme rozdělit neuronovou síť do dvou částí nebo podsítí, kde každá z nich zůstane dopřednou neuronovou sítí. První podsíť

bude začínat vstupní vrstvou a končit krčkem, druhá bude začínat krčkem a končit výstupní vrstvou. Pokud byla síť vhodně navržena, provádí nyní první část sítě mapování ze vstupního prostoru X do prostoru krčku Y. Označme tedy mapovací funkci písmenem g, pak tedy $Y = g(X)$. Druhá část sítě provádí mapování z prostoru krčku Y do prostoru aproximovaných vstupních dat X'. Pojmenujme tuto transformaci h. Potom lze psát $X' = h(Y)$. Jelikož je síť učena tak, aby $X' = X$, musí platit $h = g^{-1}$ a tedy, že spojitá mapovací funkce h reprezentuje spojitou inverzní funkci vůči g. Prostory $X (\simeq X') \subset \Re^m$ a $Y \subset \Re^f$ jsou spolu tedy dle definice globálně homeomorfní. Fakt, že X a Y jsou spolu homeomorfní, nám zaručuje (i když jen pomyslně), že nejsou ztraceny žádné informace při projekci do variety krčku Y, protože se vždy pomocí druhé části sítě, která provádí inverzní transformaci, můžeme vrátit na varietu původní.



Obrázek 12: Reprezentace mapování, ke kterým dochází uvnitř AANN, zprava vstupní prostor X, prostor krčku Y a rekonstruovaný prostor X'

Obrázek 12 je vizuální reprezentací mapování z 3-D vstupního prostoru do 2-D prostoru krčku a následně inverzní transformací pomocí druhé části sítě zpět do 3-D prostoru rekonstruovaných vstupů. Naznačená změna tvaru výstupních dat se snaží simulovat chybu rekonstrukce. Tím je popis sítě a její vnitřní fungování dokončeno. Předešlý rozbor lze jednodušeji slovně definovat tak, že AANN hledá základní pratvar analyzovaných dat. Aby ho našla, musí musí být šířka krčku správně velká, ale ne menší ani větší. V případě, že je šířka krčku příliš malá, dochází ke ztrátě informace a extrahovaná data na výstupu krčku mohou být zcela nesmyslná. Pokud je šířka krčku moc velká, získáváme transformaci vstupních dat do prostorů o nižší dimenzi, ve kterých ale pravděpodobně existuje více možných řešení (tvarů rozložení) s akceptovatelnou rekonstrukcí. Tyto extrahované parametry pak jsou obvykle opět nesmyslné. Obvyklou cestou k určení šířky krčku je metoda pokus-omyl.

3 OpenCL

OpenCL™ je prvním otevřeným standartem pro multiplatformní praralelní programování moderních procesorů, kterými jsou vybaveny dnešní PC, servery a další zařízení. OpenCL (Open Computing Language) výrazně zvyšuje rychlost a odezvu širokého spektra software v mnoha zájmových odvětvích od her a zábavy přes vědecké aplikace a aplikace ve zdravotnictví.

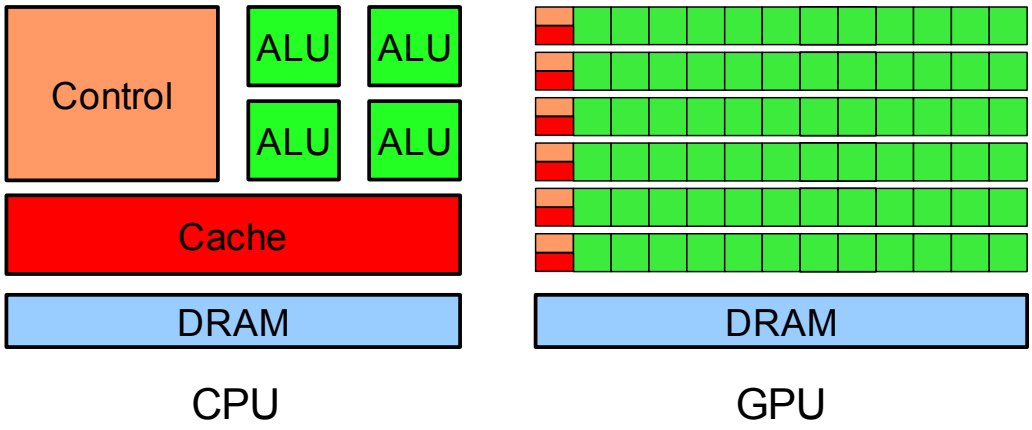
OpenCL má velmi širokou aplikovatelnost, od uživatelského software po HPC (High-performance computing), díky své nízkoúrovňové abstrakci. Vytvořením efektivního a výkoného nízkoúrovňového programového rozhraní formuje OpenCL vrstvu ekosystému paralelního programování platformě nezávislých nástrojů a aplikací. Z pohledu programátora definuje OpenCL abstraktní hardwarové zařízení a k němu unifikované ovládací softwarové rozhraní, pomocí kterého aplikace přistupují ke konkrétním výpočetním možnostem různých hardwarových platforem.

Skupina Khronos group vytváří tento standart za pomoci dalších celosvětových společností a institucí, mezi které patří 3DLABS, Activision Blizzard, AMD, Apple, ARM, Broadcom, Codeplay, Electronic Arts, Ericsson, Freescale, Fujitsu, GE, Graphic Remedy, HI, IBM, Intel, Imagination Technologies, Los Alamos National Laboratory, Motorola, Movidius, Nokia, NVIDIA, Petapath, QNX, Qualcomm, RapidMind, Samsung, Seaweed, S3, ST Microelectronics, Takumi, Texas Instruments, Toshiba a Vivante. [29]

3.1 Nástup paralelizace

Paralelizace se v poslední době stává jedinou možnou cestou k navýšení výkonu. Díky poptávce po stále výkonějších realtime HD 3-D grafických akcelerátorech, se vyvinuly dnešní GPU do vysoce paralelizovaných, multivláknových, mnohajádrových procesorů s ohromnou výpočetní silou a velmi vysokou paměťovou propustností. Nasazení a využití této síly v ostatních aplikacích je tedy v podstatě až druhotným jevem. Principielní rozdíl v architektuře GPU oproti CPU naznačuje Obrázek 13. [30]

Pro upřesnění, GPU je speciálně navržena na řešení problémů, které mohou být popsány jako datově-paralelní výpočty. Při těchto výpočtech je prováděn stejný program na mnoha datových elementech zároveň, s vysokou aritmetickou intenzitou, tedy vysokým poměrem aritmetických operací k paměťovým operacím. Protože je stejný program prováděn v každém datovém elementu, není potřeba provádět příliš složitou kontrolu toku a větvení. Navíc díky vysoké aritmetické intenzitě je pomocí výpočtů možné eliminovat zpoždění přístupu do paměti, které nastává při přesunech velkých objemů dat.



Obrázek 13: Principielní rozdíl architektury CPU a GPU

Grafické karty jsou tedy velmi specializované procesory, nelze očekávat, že by z jejich výkonu těžily všechny programy. Ve skutečnosti to mnohdy ani není potřeba, protože například pro programy typu kanacelářských balíků Office jsou již dnes plně postačující klasické procesory. Proto jak již bylo zmíněno v úvodu, nasazení OpenCL je momentálně vhodné spíše v multimediálních aplikacích, ve vědeckém software či v medicíně. Pro zajímavost, existují i nástavby pro programy typu Matlab a Maple, které tak mohou profitovat z jejich výkonu v případě různých matematických výpočtů.

Záměrem vzniku OpenCL tedy není ani tak nasazení ve všech aplikacích, jako spíš možnost využití všeobecně dostupných výkoných grafických karet v efektivně paralelizovatelných výpočtech. Paralelní programování však provází mnoho obtížných překážek, které často znesnadňují jeho nasazení tam, kde bychom ho obvykle očekávali. Většina současných aplikací je totiž vytvořena pomocí starého typu tzv. sekvenčního (postupného) programování. Proto je nutná reimplementace některých částí kódu a mnohdy dokonce i celých programů. Pro programátora je navíc paralelní programování spojeno s novou programovací metodiku.

3.2 Architektura OpenCL

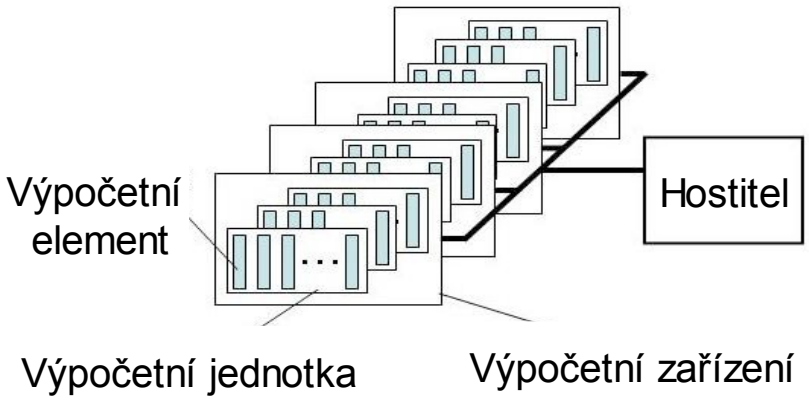
OpenCL je průmyslový standart na programování souborů heterogeních CPU, GPU a jiných diskrétních výpočetních zařízení, organizovaný a sloučený do jediné platformy. Jedná se o více než programovací jazyk. Jedná se o strukturu určenou k paralelnímu programování a vývoji software, která obsahuje vlastní jazyk, API (application programming interface), knihovny a exekuční systém. Příkladem využití OpenCL může být vývoj libovolných náročných algoritmů, které jsou prováděny na GPU bez potřeby jejich mapovaní na 3D API typu OpenGL nebo DirectX. Skutečným cílem je umožnit programátorovi psát efektivní a přenositelný kód. Do této skupiny lze zařadit například vývoj knihoven a aplikací orientovaných na výkon. [31]

Pro popis hlavních principů tohoto standartu se používá tato hierarchie modelů:

- Model platformy
- Paměťový model
- Exekuční model
- Programovací model

3.2.1 Model platformy

Model platformy OpenCL je definován na obrázku (Obrázek 14). Model obsahuje hostitelský systém (např. PC), ke kterému je připojeno jedno nebo více OpenCL zařízení. Zařízení OpenCL je dále rozděleno na více výpočetních jednotek, které jsou dále děleny na jeden nebo více výpočetních elementů. Samotný výpočet probíha až ve výpočetních elementech. Aplikace využívající OpenCL běží na hostitelském systému v souladu s modely nativními jeho platformě. Z hostitele jsou odesílány řídící příkazy k provedení výpočtů na jednotlivých výpočetních elementech připojeného zařízení. Každý výpočetní element v jedné výpočetní jednotce může zpracovávat jediný tok instrukcí v módech SIMD („single instruction multiple data“) nebo SPMD („single process multiple data“).

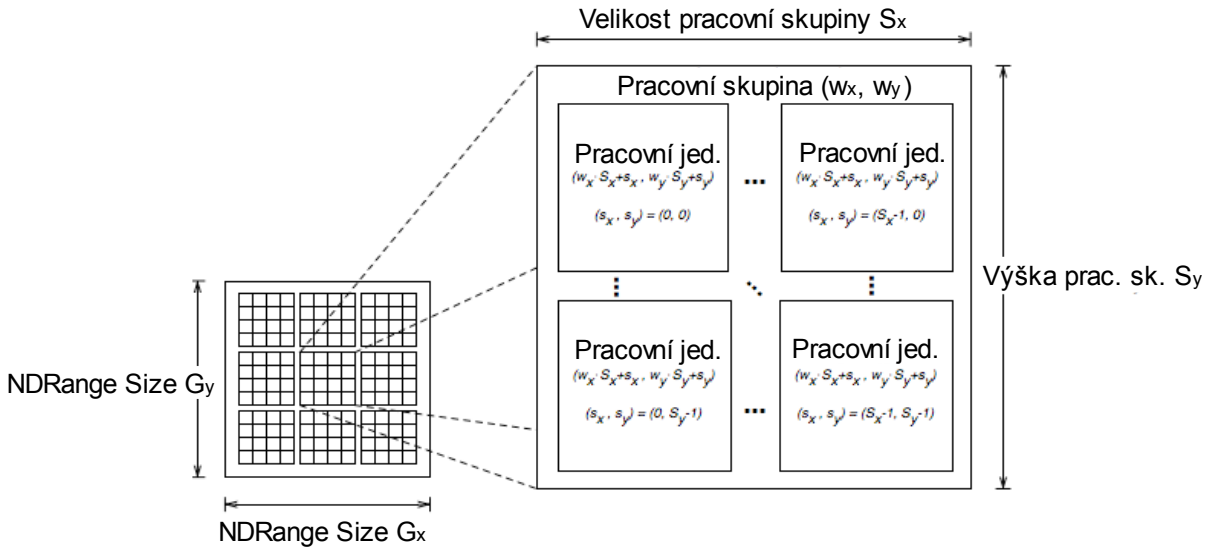


Obrázek 14: Model platformy OpenCL

3.2.2 Exekuční model

Provádění OpenCL programu se skládá ze dvou částí. První částí jsou jádra, která se provádějí na jednom nebo více OpenCL zařízeních a hostitelský program, který je spuštěn na hostiteli. Hostitelský program vytváří a definuje kontext, ve kterém jsou jádra spouštěna a kontroluje jejich provádění. Základ exekučního modelu je definován systémem, jakým probíhá spouštění jader. Když hostitel předloží jádro k provedení, je nejprve definována matice či prostor indexů. Pro každý index v této matici je spuštěna jedna instance předloženého jádra. Tato instance se nazývá pracovní jednotka a je globálně identifikována svou souřadnicí ve zmíněném prostoru indexů. Každá jednotka provádí stejný kód, ale zpracovávaná data a průchod kódem se obvykle liší. Pracovní jednotky jsou organizovány do

pracovních skupin, které rozdělují prostor indexů na větší celky. Každá skupina má také přiřazen svůj identifikátor se stejnou dimenzionalitou jako v případě pracovních jednotek. Každá pracovní jednotka má navíc unikátní lokální identifikátor v rámci pracovní skupiny. Pro unikátní identifikaci pracovní jednotky je tedy možné využít buď její globální souřadnici nebo kombinaci jejího lokálního indexu a indexu pracovní skupiny. Z hardwarového pohledu jsou pracovní jednotky v dané pracovní skupině prováděny paralelně na výpočetních elementech jedné výpočetní jednotky. Indexový prostor v rámci OpenCL je nazýván „NDRange“. NDRange je N-dimenzionální indexový prostor, kde *N* může nabývat hodnot *[1, 2, 3]*.



Obrázek 15: Příklad NDRange indexového prostoru zobrazujícísystematiku popisu souřadnic pracovních skupin a jednotek

Obrázek 15 ukazuje rozložení pracovních skupin v indexovém prostoru a rozložení pracovních jednotek ve skupinách.

Další důležitou součástí exekučního modelu jsou kontexty a fronty. Hostitel pro provádění jader vždy vytváří kontext. OpenCL kontext obsahuje následující prostředky:

- Kolekci zařízení, které se pro provádění budou využívat.
- Jádra, která se budou provádět na OpenCL zařízeních.
- Programové objekty, zdrojové kódy jader a strojové kódy, které je implementují.
- Paměťové objekty, které jsou přístupné hostiteli a OpenCL zařízení. Paměťové objekty obsahují hodnoty, které jádra zpracovávají v průběhu vykonávání.

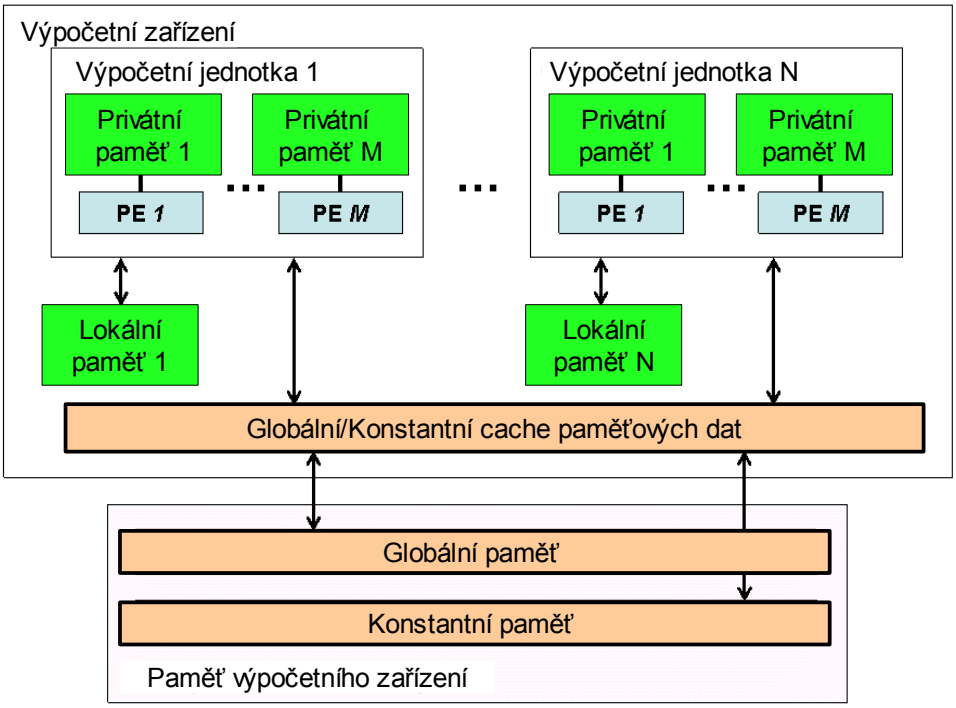
Vytvoření kontextu je prováděno na hostiteli pomocí OpenCL API. Následně je vytvořena struktura nazývaná fronta příkazů, pomocí které je řízeno pořadí zpracovávání specifických jader na jednotlivých zařízeních.

3.2.3 Paměťový model

Pracovní jednotky mají při provádění jádra přístup ke čtyřem druhům paměti:

- Globální paměť: v tomto regionu je povolen zápis i čtení všem jednotkám ve všech pracovních skupinách. Čtení i zápis v globální paměti může být podle vlastností zařízení prováděno přes cache.
- Paměť konstant: je region globální paměti, který zůstává konstantní v celém průběhu provádění jádra. Paměťové objekty jsou alokovány a inicializovány hostitelem.
- Lokální paměť: tento region je přístupný pouze pro jednotky ze stejné pracovní skupiny. Tato paměť je využívána k alokování proměnných, které jsou používány v rámci celé pracovní skupiny.
- Privátní paměť: tento region je privátní pro samotnou pracovní jednotku. Proměnné definované v této paměti nemohou být sdíleny ostatním pracovním jednotkám.

Aplikace prováděná na hostiteli využívá OpenCL API na vytváření paměťových objektů v globální paměti a na zařazování příkazů, které tuto paměť zpracovávají. Paměťové modely hostitele a OpenCL zařízení jsou na sobě nezávislé, což je způsobeno tím, že hostitel se nachází zcela mimo OpenCL. Čas od času však spolu tyto dvě části potřebují interagovat. Tuto interakci je možné provést buďto explicitním nakopírováním dat nebo namapováním a demapováním částí paměťových objektů opět pomocí OpenCL API. Schéma paměti uvnitř OpenCL zařízení je zobrazena na obrázku (Obrázek 16).



Obrázek 16: Diagram rozložení paměťových regionů

3.2.4 Programovací model

Exekuční model OpenCL podporuje datově paralelní a úlohově paralelní programovací modely a také hybridy těchto dvou modelů. Primárním využívaným modelem je datově paralelní model.

1. **Datově paralelní model**

   Tento model definuje výpočty ve smyslu sekvenčního provádění instrukcí aplikovaných na více elementů paměťového objektu. Indexový prostor asociovaný s prováděním OpenCL definuje pracovní jednotky a systém, jakým jsou na ně mapována data. Ve striktně datově paralelním modelu je namapován na každou pracovní jednotku jeden zpracovávaný element paměti, kterou jádra paralelně zpracovávají. OpenCL však implementuje relaxovanou verzi, ve které není takto striktní mapování nutné.

   OpenCL tedy poskytuje hierarchický datově paralelní programovací model. Existují dvě cesty jak specifikovat tuto hierarchii. V explicitním modelu nejprve programátor definuje celkový počet pracovních jednotek, které budou pracovat paralelně a také jakým způsobem jsou pracovní jednotky rozděleny do pracovních skupin. V implicitním modelu programátor specifikuje pouze množství pracovních jednotek a rozdělení pracovních skupin je obstaráno samotnou implementací OpenCL.

2. **Úlohově paralelní model**

   V tomto modelu je prováděna pouze jediná instance jádra nezávisle na indexovém prostoru. To je ekvivalentní provedení jádra na výpočetní jednotce pomocí pracovní skupiny, která má pouze jedinou pracovní jednotku. V tomto modelu uživatel popisuje paralelismus:

   - využíváním vektorových datových typů implementovaných v zařízení,
   - zařazováním více úloh do fronty příkazů nebo
   - zařazováním nativních jader vyvinutých v programovacích jazycích ortogonálních vůči OpenCL

3.3 PyOpenCL

Jak již bylo zmíněno, OpenCL se snaží zjednodušit a sjednotit vývoj na různých typech zařízení, mezi které lze zařadit:

- Vícejádrové procesory,
- různé architektury GPU
- a další vestavná a přenosná zařízení.

Programovácí jazyk OpenCL C je založen na normě „ISO/IEC 9899:1999 - Specifikace jazyka C“ (C99), liší se však množstvím rozšíření a některými omezeními, která obsahuje. [32] Jak je uvedeno, C99 je poměrně starý, avšak všeobecně známý standart, takže zvládnutí

programování není příliš složité a omezuje se pouze na zvládnutí nových rozšíření. Návrh programů jader prováděných na GPU, je obvykle psán v nativním kódu OpenCL. Interakce s OpenCL API však není omezeno pouze na programovací jazyk C. V případě dnešních vysokoúrovňových programovacích jazyků jsou obvykle k dispozici knihovny, které zprostředkovávají vazby na aplikační programovací rozhraní OpenCL, zapouzdřují a vystavují jeho prostředky v podobě různých knihoven tříd a dalších nástrojů. Abychom využili možnosti vysokoúrovňových programovacích jazyků, nabízí se využití platforem Microsoft .net, Java, Python a dalších. Pokud chceme využít multiplatformnosti a možnosti využívat hostitelský program na co nejširší škále zařízení, nabízí se jako nejlepší volba Python či Java. Python oproti Javě přináší některé další výhody, takže se v našem případě zaměříme na jazyk Python a jeho vazbu do API OpenCL, která se jmenuje PyOpenCL.[33] [34]

Tento balíček obsahu vše potřebné pro vývoj v OpenCL, mezi hlavní přednosti patří:

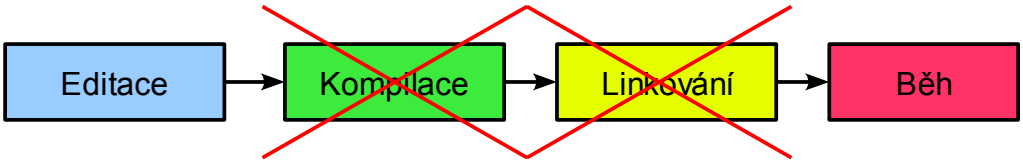
- podpora téměř všech operačních systémů (Linux, OS X, Windows),
- kompletní podpora OpenCL 1.1,
- podporuje interaktivní mód,
- automatická správa prostředků,
- automatické testování chyb a chybové reporty,
- integrace s balíčkem NumPy.

Podíváme-li se podrobněji na programovací jazyk Python, jedná se o multiparadigmatický programovací jazyk. Tento jazyk je imperativní, objektově orientovaný, funkcionální, dynamicky typovaný s automatickou správou paměti. Python se také velmi často používá jako prototypovací jazyk, protože vývoj je v něm velmi rychlý a pohodlný. Tato vlastnost je velmi užitečná při tvorbě experimentálních aplikací. Pokud je výsledný kód otestován, provádí se většinou přepis kritických částí kódu do efektivnějších jazyků. Mezi další velké výhody patří také rozsáhlá aktivní komunita a rozsáhlý ekosystém vědeckých balíčků. Výše zmiňovaný balíček NumPy je pak rošířením Pythonu pro rychlé vědecké a matematické výpočty, jeho jádro je napsané v jazyku C právě pro využití plného výkonu platformy. Nabízí podporu pro práci s velkými multidimenzionálními poli a navíc obsahuje knihovnu mnoha matematických funkcí. Při používání v Pythonu je jeho rychlost téměř porovnatelná s jazykem C. V našem případě se však z největší části nemusíme starat o rychlost provádění kódu na hostiteli, protože hlavní výpočty jsou prováděny na GPU. Ukázka jednoduchého kódu pro součet dvou náhodných vektorů pomocí kombinace OpenCL a Pythonu je uvedena ve skriptu (Skript 1). Tento skript je přejat a upraven o komentáře z ukázkových kódů PyOpenCL. Nutno ještě zmínit další hojně využívané balíčky, mezi které patří například MatPlotLib, který je určen k vytváření grafů publikační kvality, jeho možnosti jsou srovnatelné, ne-li širší v porovnání s programy typu Matlab či Mathematica. V neposlední řadě nutno dodat, že tyto nástroje jsou poskytované pod open source licencemi.

```
#import používaných knihoven
import pyopencl as cl
import numpy
import numpy.linalg as la

#generace dvou náhodných vektorů pomocí balíčku numpy
a = numpy.random.rand(50000).astype(numpy.float32)
b = numpy.random.rand(50000).astype(numpy.float32)

#vytvoření OpenCL kontextu a vytvoření fronty
ctx = cl.create_some_context()
queue = cl.CommandQueue(ctx)

#nastavení paměťových objektů pro jádro
mf = cl.mem_flags
a_buf = cl.Buffer(ctx, mf.READ_ONLY | mf.COPY_HOST_PTR, hostbuf=a)
b_buf = cl.Buffer(ctx, mf.READ_ONLY | mf.COPY_HOST_PTR, hostbuf=b)
dest_buf = cl.Buffer(ctx, mf.WRITE_ONLY, b.nbytes)

#sestavení a kompilace jádra prováděného v pracovních jednotkách
prg = cl.Program(ctx, """
    __kernel void sum(__global const float *a,
    __global const float *b, __global float *c)
    {
      int gid = get_global_id(0);
      c[gid] = a[gid] + b[gid];
    }
    """).build()

#spuštění provádění jádra
prg.sum(queue, a.shape, None, a_buf, b_buf, dest_buf)

#načtení výsledného vektoru z paměti a jeho výpis
a_plus_b = numpy.empty_like(a)
cl.enqueue_read_buffer(queue, dest_buf, a_plus_b).wait()
print(la.norm(a_plus_b - (a+b)), la.norm(a_plus_b))
```

Skript 1: Zdrojový kód jednoduchého skriptu, který provádí součet dvou vektorů

PyOpenCL vlastně zapouzdřuje všechny konstrukty z OpenCL API do objektů, čímž se velmi zjednodušuje práce a zrychluje psaní programů. Při skriptování GPU nemusí být kód jader nutně konstantní, protože dochází k jeho kompilaci v průběhu hostitelského programu. Jelikož v Pythonu platí kód = data, můžeme v průběhu programu bez problémů kódy jader měnit. Externí změny kódů lze navíc provést bez rekompilace, která je v případě jiných jazyků nutná. Skriptování s PyOpenCL je velmi přehledné, protože čitelnost kódu je díky základní syntaxi Pythonu vynikající. Obrázek 17 popisuje průběh práce při vývoji v Pythonu.



Obrázek 17: Workflow v průběhu tvorby a provádění aplikace v Pythonu

4 Cíle práce

- Zvolit vhodný typ a model neuronové sítě pro zpracování elektrochemických dat
- Navrhnout a ověřit funkčnost sítě na modelovém případu v prostředí Matlab
- Implementovat neuronovou síť ve zvoleném programovacím jazyku a provést její ověření
- Zpracovat a vyhodnotit reálná data pomocí navrženého modelu

5 Praktická část

5.1 Materiál a metody

5.1.1 Analyzovaná data

Analyzovaná elektrochemická data byla změřena na pracovišti v Laboratoři molekulární biologie a biochemie na Mendelově zemědělské a lesnické univerzitě v Brně. Pocházejí z odebraných vzorků krví a tkání pacientů. Vzorky byly dodány z Kliniky otorinolaryngologie a chirurgie hlavy a krku, Fakultní nemocnice u sv. Anny v Brně, Pekařská 53, CZ-656 91 Brno. Veškeré vzorky pocházely od pacientů se zhoubným nádorem v oblasti hlavy a krku (krve podle lokalizace – ústní část hltanu (orofarynx) n = 69, dutina ústní n = 18, hrtanová část hltanu (hypofarynx) n = 14, paranasální dutiny n = 4, hrtan (larynx) n = 42, příušní slinná žláza (glandula parotis) n = 4; nádorové tkáně podle lokalizace – ústní část hltanu (orofarynx) n = 22, hrtan (larynx) n = 25, hrtanová část hltanu (hypofarynx) n = 5, dutina ústní n = 3). Tumorové tkáně byly odebrány při operaci, která je ve většině případů volena jako první typ léčby. Vzorky krví kontrolní skupiny pocházely z Ústavu sportovní medicíny, Křižkovského 22, Brno, v celkovém počtu n = 58. Vzorky byly analyzovány metodou diferenční pulzní voltametrie na přístroji 747 VA Stand ve spojení s 746 VA Trace Analyzer a 695 Autosampler (Metrohm, Švýcarsko) v klasickém tříelektrodovém uspořádání. Pracovní elektrodou byla visící rtuťová kapková elektroda (HMDE) s plochou kapky 0,4 mm^2; referentní elektrodou byla Ag/AgCl/3M KCl a pomocnou platinová elektroda. V analyzovaných datech se objevují tři pozorovatelné signály Cat1 - Ep = - 1,35 V, Cat2 - Ep = - 1,48 V a RS_2Co Ep = -1,20 V. Pro vnitřní potřeby neuronové sítě byla celá sada dat normalizována na rozsah 0-1 a oříznuta tak, aby zachycovala stejný rozsah potenciálu. Posledním krokem pak bylo převzorkování těchto dat na shodný počet vzorků. Obrázek 30 zobrazuje analyzovaná data ve své původní podobě.

5.1.2 Programové a hardwarové vybavení

K základnímu návrhu a ověření funkčnosti neuronové sítě byl využit program MATLAB® 7.9.0 (R2009b), jedná se o prostředí určené pro výpočetní, analyzační a vizualizační úkoly od společnosti MathWorks®, Inc. Nejdůležitější použitou komponentou z Matlabu je bezesporu Neural Network Toolbox. Matlab byl provozován na PC vybaveným dvoujádrovým procesorem AMD Turion II 2.3 Ghz s 3 GB operační paměti, běžícím pod operačním systémem Windows 7. Pro implementaci vlastního modelu neuronové sítě byl použit programovací jazyk Python 2.7.1 ve spojení s technologií OpenCL 1.1. Mezi nejdůležitější použité balíčky a komponenty z této platformy patří SciPy, NumPy, Matplotlib a PyOpenCL. NumPy je fundamentálním balíčkem k vědecké výpočetní práci v Pythonu, integruje v sobě hlavně rychlou práci s objekty n-dimenzionálních polí. SciPy, který je na

NumPy postaven, pak v sobě integruje mnoho důležitých a nepostradatelných numerických rutin. Matplotlib je knihovnou určenou pro produkci grafů. PyOpenCL je základním kamenem pro provádění výpočtů na procesorech grafických karet z prostředí Python. Pro psaní skriptů bylo využito nativní prostředí Idle a pro provádění skriptů interaktivní příkazový řadek IPython. Vývoj vlastního modelu neuronové sítě a zpracovávání dat bylo prováděno na PC vybaveném procesorem Intel Core 2 E6600 s 2GB operační paměti běžícím na operačním systému ArchLinux. Grafickou kartou určenou k provádění OpenCL výpočtů byla NVidia GTX N470.

5.1.3 Typ a model neuronové sítě

Na analýzu dat byl využit autoasociativní model neuronové AANN sítě využívaný v nelineární analýze hlavních komponent NLPCA. Tímto způsobem je možné provést jak redukci dimenzionality vstupních dat, tak následnou analýzou extrahovaných parametrů klasifikaci jednotlivých vstupních případů. Extrahované parametry by navíc měly obsahovat informaci spojenou s fyzikálními jevy, které jsou zodpovědné za tvary voltametrických křivek. Další výhodou této sítě je, že se vyhneme návrhu skupin, do nichž se snažíme klasifikované případy zařadit.

K určení šířky krčku, jako jednoho z nejdůležitějších parametrů, jsme se pokusili využít metodu z fraktální geometrie, zjištěním Minkowski–Bouligandovy dimenze analyzovancých dat. Tato metoda fungovala při analýze modelových dat, ale při reálných datech její předpovědi nebyly správné, což lze připsat malé sadě analyzovaných dat. V případě experimentů s reálnými daty byla tedy šířka krčku stanovena metodou pokus-omyl. Šířka mapovací a demapovací vrstvy byla určena metodou pokus-omyl. Proto se také určování fraktální dimenze u dalších experimentů neprovádělo.

5.2 Výsledky a diskuze

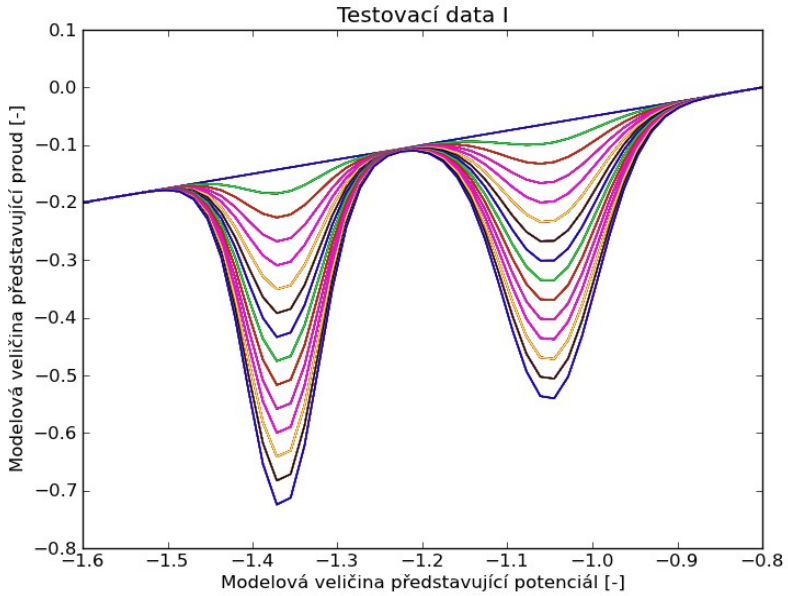
5.2.1 Modelová data pro ověření funkčnosti vybrané neuronové sítě

Pro ověření zvoleného modelu neuronové sítě byl nejprve vygenerován testovací set dat. Jednalo se o dva píky nasuperponované na základní nosnou. Píky byly vygenerovány pomocí gaussovské funkce o parametrech:

Tabulka 2: Hodnoty parametrů generovaných dat.

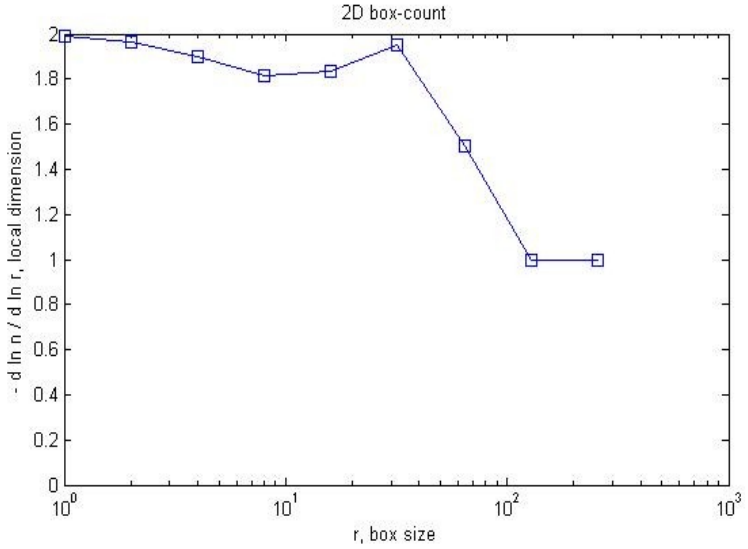
Parametr	a	b	c
Pík 1	$1/(c*\sqrt{2*pi})$	0	$\sqrt{0.4}$
Pík 2	$1/(c*\sqrt{2*pi})$	3.5	$\sqrt{0.6}$

Výška obou píků byla lineárně rozdělena na *15* úrovní z intervalu *[0; maximální výška píku]* a invertována. Takto vytvořené modelové průběhy obou píků byly superponovány ve všech variantách výšek na lineárně stoupající offset z intervalu *[-0,2; 0]*. Výsledný testovací set obsahoval *225* křivek s variacemi obou superponovaných píků. Počet vzorků jednotlivých křivek byl nastaven na *50*. Modelové testovací křivky jsou zobrazeny na obrázku (Obrázek 18). V průběhu experimentů budeme tyto vygenerované křivky nazývat modelovými či testovacími daty typu *I.*



Obrázek 18: Vygenerovaná modelová data.

V další části byla v programu Matlab určena dimenzionalita těchto dat viz. (Obrázek 19), tak aby byla zvolena správná šířka krčku sítě a předešlo se zdlouhavému testování metodou pokus omyl. Následně byl v prostředí Matlab napsán jednoduchý skript na učení autoasociativní neuronové sítě. Tento skript obsahuje úpravu vstupních dat na rozsah *[0; 1],* konstrukci neuronové sítě se zadanými parametry, její učení a vizualizaci výstupních dat. V průběhu analýz bylo zjištěno, že určování šířky krčku pomocí zmíněné metody bylo v případě reálných dat zavádějící. Proto jsme museli v experimentech s reálnými daty tuto metodu opustit a stanovovat šířku krčku pomocí metody pokus-omyl. Proto u dalších experimentů určování šířky krčku tento postup nenajdeme. S ohledem na Matlab a prověření funkcionality vybraného designu sítě pak byl kvůli časové náročnosti učení sítě proveden pouze jeden exeriment a to právě s výše uvedenými modelovými daty typu I.

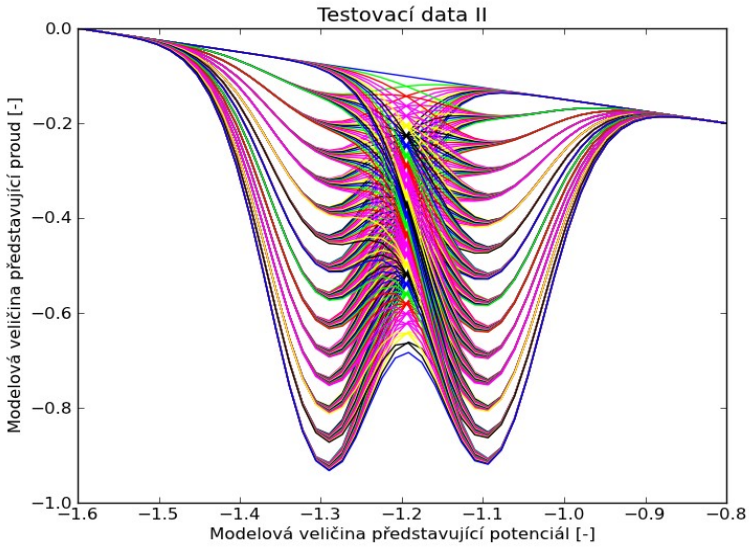


Obrázek 19: Určení šířky krčku sítě.

Druhým testovacím experimentem byla data s překrývajícími se píky. Pro simulaci tvaru píků byly opět využity gaussovské funkce. Jejich parametry jsou shrnuty v tabulce (Tabulka 3). Počet křivek byl opět *225*, tedy kombinace patnácti úrovní výšek píků. Tak aby tvar křivek lépe odpovídal reálným datům, byl upraven offset základní nosné oproti předchozím modelovým datům na interval *[0; -0,2]*.

Tabulka 3: Hodnoty parametrů generovaných dat.

Parametr	a	b	c
Pík 1	1/(c*√2*pi)	1.05	√0.95
Pík 2	1/(c*√2*pi)	2.85	√1.1



Obrázek 20: Testovací data s prolínajícími se píky

Obrázek 20 zobrazuje výsledné modelové křivky typu II. Na těchto vygenerovaných testovacích datech byla provedena rozšířená analýza a ověření zvoleného modelu neuronové sítě pro prolnuté píky. Tato data byla zpracována už pouze námi navrženým a implementovaným modelem neuronových sítí.

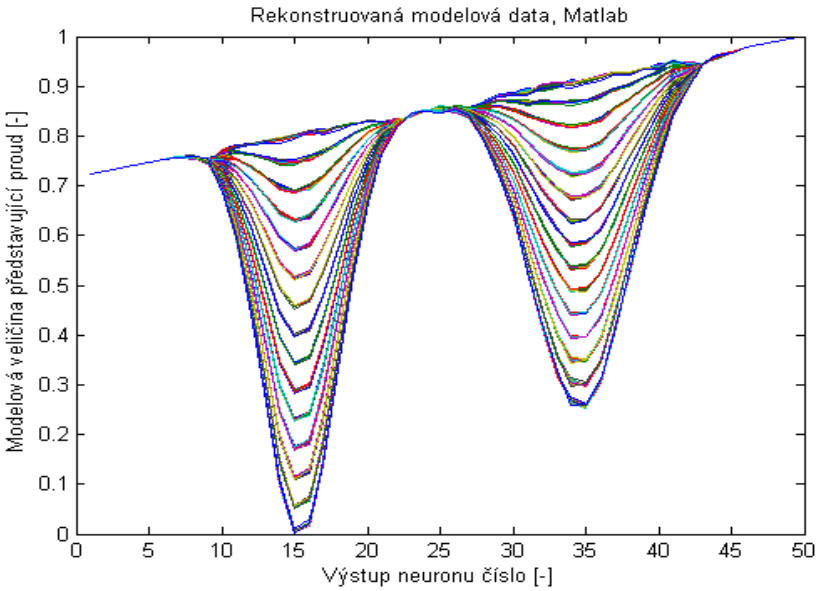
5.2.2 Neuronová síť v prostředí Matlab

Parametry neuronové sítě v Matlabu byly vybrány následujícím způsobem. Šířka krčku $w_n = 2$ neurony, jenž byla určena z dimenzionality vstupních dat, viz. Obrázek 19 . Tento parametr teoreticky odpovídá skutečnosti, v modelových datech jsou přítomny dva parametry ovlivňující jejich tvar – dvě superponované varianty píků stejného rozložení různé výšky. Šířky mapovací a demapovací vrstvy jsou voleny stejné, a obvykle je tato jednotná šířka volena vyšší než je počet vstupních neuronů, pro náš případ *50* vstupních neuronů byla tedy zvolena šířka $w_d = w_m = 100$. Po několika experimentech metodou pokus-omyl se ukázalo, že je výhodnější mapovací i demapovací vrstvu rozdělit na dvě vrstvy o poloviční šířce a to jak z hlediska rychlosti učení tak výsledné účinnosti sítě. Výsledkem jsou tedy dvě mapovací a demapovací vrstvy o šířce $w_d = w_m = 50$ neuronů. Jelikož jsou simulace neuronových sítí časově náročné, nebylo možné se přímo zabývat dopady změn šířky mapovací a demapovací vrstvy a jejich počtu na rychlosti učení a na kvalitu rekonstrukce vstupních dat. Nicméně jako velmi dobrá se jeví technika využití více vnitřních vrstev pro mapovací i demapovací vrstvy, zde lze vycházet ze známého předpokladu, že více vrstev je schopno komplexněji transformovat data. Jsem toho názoru, že studiem rozložení mapovacích a demapovacích vrstev, jejich tvarů a propojení by bylo vhodné se zabývat hlouběji. Další důležitý parametr, typ trénovací funkce, byl nastaven na 'traingdx' (“Gradient descent with momentum & adaptive learning rate backpropagation”), pomocí této metody se vyhneme nastavování a optimalizaci hodnot učící konstanty, která je jedním z rozhodujících parametrů pro rychlost učení tohoto typu sítě a její konečné výkonnosti. Výběr této učící techniky byl navíc podmíněn velikostí sítě, která u komplexnějších metod souvisí s paměťovou náročností. Navíc vzhledem k implementaci vlastní neuronové sítě bylo vhodné využít porovnatelnou techniku. Počet epoch nebyl limitován, trénink byl zastaven až po dosažení akceptovatelné chyby, či po splnění předpokladů při vizuální kontrole výsledků rekonstrukce. Přenosové funkce jednotlivých vrstev byly nastaveny:

- '**purelin**' pro neurony vstupní a neurony krčku, podle experimentu i výstupní
- '**logsig**' pro neurony mapovací, demapovací a dle experimentu i výstupní vrstvy

5.2.3 Výstupy a extrahované parametry - Matlab

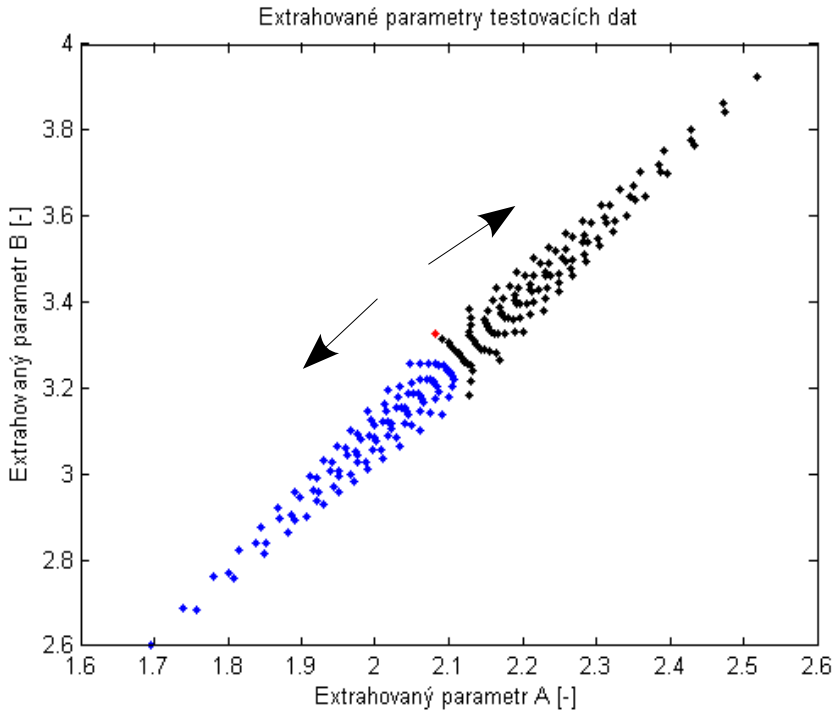
Prozkoumejme nyní rekonstrukci vstupních dat a extrahované parametry sítě navržené v prostředí Matlab. Obrázek 21 zobrazuje rekonstruovaná modelová data, po jejichž zhodnocení se blíže podíváme na parametry extrahované touto sítí.



Obrázek 21: Rekonstruovaná standardizovaná data, Matlab

Chyba rekonstruovaných dat vyjádřená pomocí střední kvadratické odchylky byla *E = 7.0226E-4*. Síť byla učena po dobu několika dnů. Z rekonstruovaných dat je patrné, že kolem hlavní nosné linie křivky jsou výstupní data zvlněná, síť by bylo pravděpodobně potřeba ještě učit, ale vzhledem k tomu, že učení již tak trvalo velmi dlouho byl pokus zastaven. Chyba nicméně není příliš vysoká, takže lze očekávat dobře uskupené extrahované parametry.

Na výstupu v ideálním případě očekáváme parametry rozložené v uzlech dvoudimenzionální pravoúhlé mřížky, ve které změna směru pohybu v první dimenzi souvisí se změnou jednoho z píků a analogicky změna pohybu ve druhé druhé dimenzi s opačným píkem. Jak je patrné, výsledná mřížka je výrazně zkosena, což však není chybné. Uzly této mřížky navíc leží na zvlněné ploše, která mřížku deformuje, tento stav úzce souvisí s chybou sítě. Obrázek 22 Zobrazuje výsledný výstup. Jak je patrné, rozložení parametrů separuje křivky do dvou regionů dle výšek superponovaných píků. Rozložení jednotlivých zobrazených parametrů ale odpovídá předpokladu a jsou rozmístěny logicky. Extrahované parametry tak tvoří zdeformovanou mřížku 15 x 15 bodů, to samozřejmě odpovídá objemu modelových dat. Analyzované křivky jsou rozděleny a barevně rozlišeny do tří skupin dle majoritního píku. Modrou barvou jsou naznačeny křivky s majoritním píkem 1, černě s majoritním píkem 2 a červenou barvou pak křivka bez píků. Šipkami je naznačen směr nárůstu jednotlivých píků. Tyto směry popisují souvislost parametrů s jednotlivými píky, přestože z následujícího obrázku není mnoho jasné, v následujících částech experimentu se vše vyjasní.



Obrázek 22: Extrahované parametry z krčku neuronové sítě v prostředí Matlab

Návrh dodatečné operace na transformaci parametrů do pravoúhlé mřížky přímo z těchto parametrů je možný, muselo by se však jednat o další nelineární transformaci. Jinou variantou je naučení nové sítě, která může mít ve výsledku nižší chybu a také vykazovat vyšší linearitu v rovině do které se parametry promítají. Poslední možností je pak další učení a minimalizace chyby, lepší rozprostření parametrů však není zaručeno. Jsem toho názoru, že za zvlnění prostoru parametrů je částečně zodpovědná i adaptace biasů, ke které v prostředí Matlab dochází, a tedy jejich různé hodnoty. Učení takto rozměrné sítě, jak již bylo zmíněno, je časově velmi náročné, proto jsme v Matlabu již další experimenty neprováděli. Proto se přesuneme k našemu programu, kde chceme využít paralelizaci a zvýšit rychlost učení a tedy i zajistit vyšší množství experimentů. Zajímavostí je, že v prostředí Matlab se nezdařilo učení sítě s nelineárními přenosovými funkcemi výstupních neuronů. Bylo tedy nutné využít lineární výstupní funkce. Učení modelových dat s prolínajícími se píky jsme v případě Matlabu zcela vypustili právě kvůli době potřebné k učení sítě. K opuštění Matlabu vedly také úspěchy při provádění pokusů s naší implementací neuronové sítě.

5.2.4 Vlastní implementace neuronové sítě

Implementace naší neuronové sítě byla provedena v jazyku Python, největší důraz pak byl kladen na využití paralelizačních schopností GPU pomocí rozhraní OpenCL, což bylo realizováno pomocí programového balíčku PyOpenCL. Hlavními navrženými používanými jádry jsou součet matic, transpozice matic, jádra na aplikaci aktivační funkce neuronu a na

úpravu vah. Nejdůležitějším používaným jádrem je násobení matic, kde bylo využito a upraveno optimalizované jádro ze vzorových kódů od společnosti NVIDIA. Tyto kódy jsou k dispozici v příloze na CD.

Použitý typ sítě je klasická dopředná vícevrstvá síť. Na učení bylo využito nejjednodušší pravidlo zpětné propagace. Abychom co nejlépe využili potenciál paralelizace, byl využit jednodávkový systém učení. Zvolený model tříd je velmi jednoduchý a pro rozsah a základ této práce zcela ideální. Skládá se z těchto tříd:

- **Network**

  Hlavní třída zprostředkovávající konstrukci, design, učení a dopřednou transformaci sítě. Inicializuje a vlastní kontext OpenCL, který je nutný pro práci s GPU. V neposlední řadě se stará o inicializaci zpětné a dopředné propagace a samozřejmě inicializaci jednotlivých vrstev. Dále také například ukládá vrstvy do souborů po dosažení nastavené chyby.

- **Layer**

  Třída, která představuje jednu vrstvu neuronů. Nese na sobě všechny informace o vrstvě, počet neuronů, hodnoty biasů, hodnoty jejích vah, typy svých aktivačních funkcí a některé další. Zároveň obstarává přesuny, nahrávání a čtení dat mezi pamětí grafické karty a pamětí hostitelského PC. Tato třída také provádí inicializaci a mutaci jednotlivých polí, které obsahuje. Hlavními parametry při vytváření instance jsou počet neuronů, typ neuronů a v případě existující sítě již známé biasy a váhy.

- **ForwardPropagation**

  Třída obstarávající dopředný průchod dat sítí, majoritní součástí je využití výše zmíněného jádra násobení matic a aplikace přenosových funkcí neuronu. V podstatě provede maticové násobení dat z výstupu předchozí vrstvy a aplikaci aktivační funkce neuronu. Vstupem při vytváření instance je OpenCL kontext a velikost pracovní skupiny.

- **BackPropagation**

  Hlavní činností je zpětný průchod chyb sítí, tedy výpočet chyb, delt, derivací přenosové funkce neuronu a aplikace modifikací vah. Tato třída využívá ponejvíce jádra násobení matic, součet matic a transpozice. Vstupem při vytváření instance je opět OpenCL kontext a velikost pracovní skupiny.

Další součástí programu jsou využívané skripty:

- **learn.py**

  Jedná se o skript využívaný k nastavení parametrů sítě, tedy načtení vstupních hodnot, nastavení parametrů vrstev, nastavení učící konstanty, faktoru momentum, spouštění učení a výpis výsledků do příkazové řádky, případně vyzualizace výstupních dat.

- **loader.py**

  Skript na transformaci dat pomocí již naučené neuronové sítě a jejich uložení. Provádí načtení dat vrstev a inicializaci této sítě. Dále se pomocí tohoto skriptu provádí extrakce a vizualizace extrahovaných komponent pomocí komponenty matplotlib.

- **prover.py**

  Skript k manipulaci s demapovací částí neuronové sítě, používaný pro sledování vlivu parametrů na výstup sítě. Pomocí rozmítání parametrů je možné analyzovat a vizualizovat jak působí parametry v prostoru extraktů na rekonstruované výstupní křivky.

- **plotter.py**

  Skript určený k obecné vizualizaci dat. Jednoduchou modifikací tohoto skriptu tak můžeme vizualizovat rekonstruovaná data, vstupní data a jiné.

- **generator.py**

  Skript využitý ke generování modelových testovacích vstupních dat, jejich popisu a uložení těchto údajů.

- **neck.py**

  Testovací skript k určení potřebné šířky krčku pomocí Bouligandovy-Minkowského dimenze pro analyzovaná data.

- **reader.py**

  Skript na načtení analyzovaných dat, která jsou většinou uložena v jednotlivých souborech, provádí také převzorkování, určení společného intervalu osy x a s tím spojeného oříznutí. Následně pak uložení těchto dat tak, aby možné je snadno načíst pomocí matematického balíčku NumPy.

Poslední částí programu je definiční soubor:

- **kernels.py**

Tento soubor obsahuje zdrojové kódy všech zmíněných jader použitých pro výpočty na GPU.

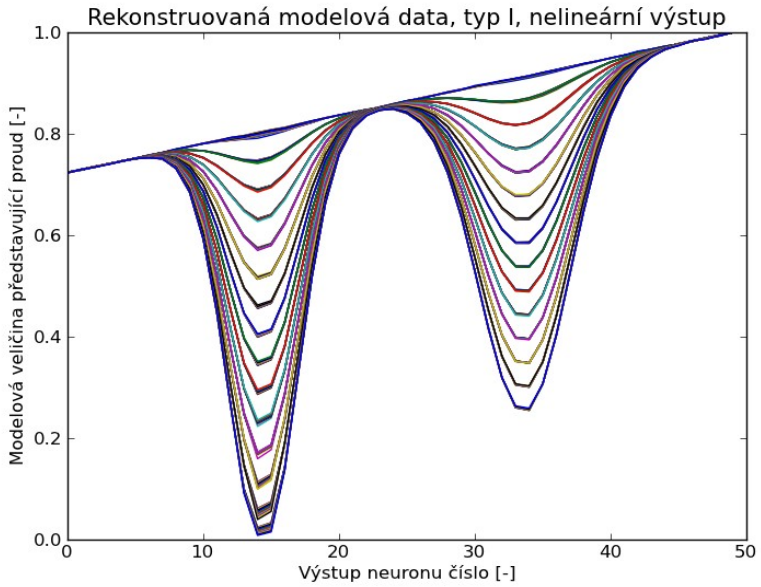
Práce s programem je pak prováděna následujícím způsobem:

1. Příprava dat pomocí skriptu reader.py
2. Určení potřebné šířky krčku – neck.py
3. Nastavení parametrů a spuštění učení sítě – learn.py
4. Produkce výstupů a extrakce – loader.py (případně vizualizace extraktů)
5. Zhodnocení a posouzení parametrů – prover.py
6. Výstupy do grafů – plotter.py

5.2.5 Analýza modelových dat – vlastní implementace

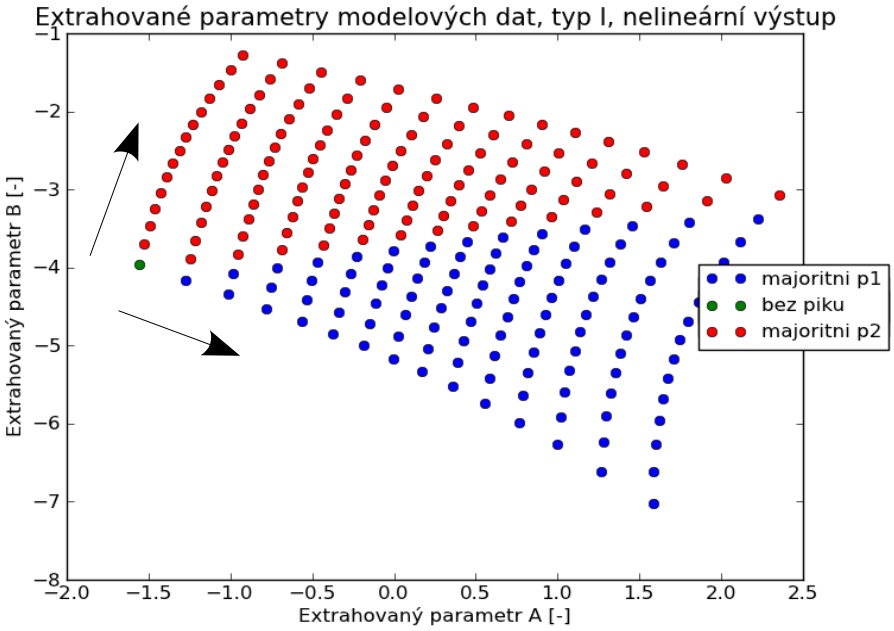
Nelineární výstupní aktivační funkce, modelová data typ I

Podívejme se tedy na stejný pokus zopakovaný v námi naprogramovaném systému a zdali je tato metoda efektivní. Budeme se tedy opět zabývat stejnými modelovými daty. Dle teorie je možno v autoasociativním modelu neuronové sítě využít různé typy přenosových funkcí pro výstupní neurony. Proto jsme v první části provedli rozbor toho, jak se tyto funkce projevují na kvalitu výstupních dat. Na obrázku (Obrázek 23) jsou zobrazena rekonstruovaná standardizovaná data modelu neuronové sítě s nelineárním výstupem.



Obrázek 23: Rekonstruovaná standardizovaná data, síť s nelineárním výstupem

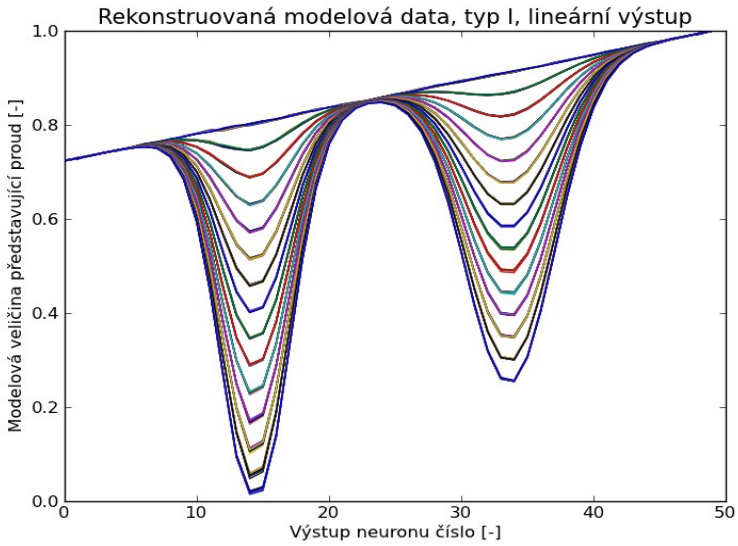
Chyba rekonstruovaných dat vyjádřená pomocí střední kvadratické odchylky byla *E = 8.641e-05*. Doba, po kterou byla síť učena byla půl dne, minimální požadované chyby však dosaženo nebylo. Jedná se tedy o jedno z předběžných zastavení. Jak je patrné, největší chyba se nachází v údolí prvního píku zleva. Tato chyba se promítá samozřejmě i do extrahovaných parametrů, viz. Obrázek 24. Další zjevnou skutečností je daleko lepší rekonstrukce kolem základní nosné a také kolem pat obou píků oproti prostředí Matlab.



Obrázek 24: Vizualizace extrahovaných parametrů modelových dat s nelineárními výstupy

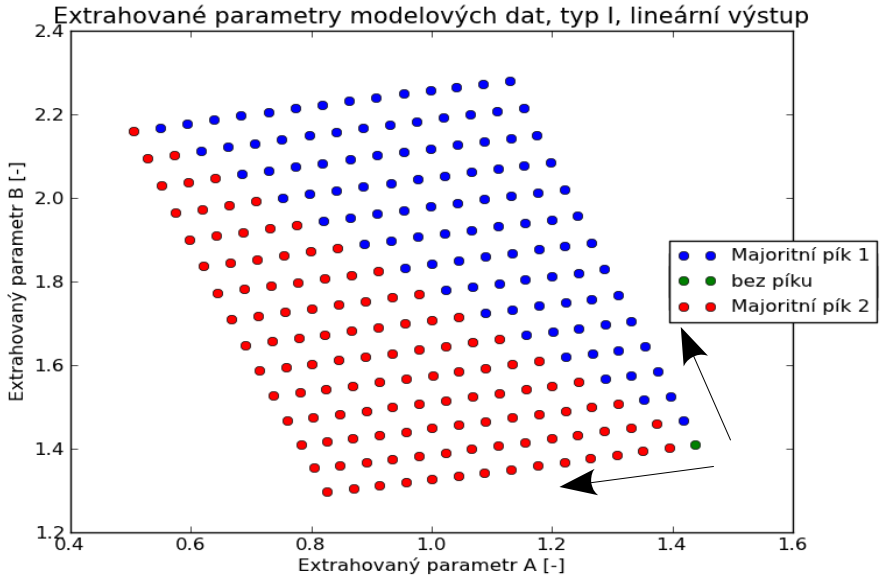
Z obrázku je zřejmé, že extrahované parametry jsou opět rozloženy do předpokládané mřížky. Nyní je rozložení vizuálně daleko přesnější než v předchozím pokusu. Data jsou opět zobrazena s ohledem na majoritu píků tak, aby bylo možné jasně určit směr rozmítání parametru s ohledem na ovlivňovaný pík. Tyto směry jsou naznačeny šipkami. Za chybu prvního píku zodpovídá zvlnění v modře značených parametrech, které je způsobeno nedostatečným natrénováním sítě a chybou normalizace vstupních dat. Podívejme se na chybu návrhu, je možné si všimnout, že první pík je ve svém maximu (údolí) roven nule, takže vstupní hodnoty pro neuron se sigmoidní přenosovou funkcí by musely mít vstup na hodnotě -∞, zároveň její derivace, která se uplatňuje při přenosu chyby, je v oblastech kdy se požadovaný výstup blíží k nule také téměř nulová. Rozptyl a zvlnění parametrů druhého píku je pravidelnější, tomu odpovídá i lepší rekontrukce v jeho oblasti. Natočení mřížky může souviset se stoupající tendencí nosné v modelových datech. Dalším pravděpodobnějším vysvětlením je, že rotace je zcela náhodná, síť totiž po inicializaci nemá žádné znalosti o analyzovaných datech, výsledná rotace rozložení parametrů tedy může záviset pouze na inicializaci. Oproti předchozímu experimentu jsou extrahované parametry zřetelně usazené na méně zvlněné ploše. Tato data by bylo nyní možné pomocí dodatečných transformací převést na pravoúhlou mřížku a následně stanovit převodní charakteristiky mezi parametry a výškou píku daleko lépe než v předchozím případě. Pojďme ale raději k dalšímu pokusu s lineární výstupní vrstvou, kde jsou extrahované parametry ještě daleko lepší.

Lineární výstupní aktivační funkce, modelová data typ I



Obrázek 25: Rekonstruovaná standardizovaná data, síť s lineárním výstupem

Chyba rekonstruovaných dat vyjádřená pomocí střední kvadratické odchylky je *E = 4.027e-05*. Počet cyklů nutných k dokončení učení byl 852 mutací sítě, každá o 2000 epochách. Při průměrné době trvání 2000 epoch 24.6 sekund byl celkový čas 5 hodin a 50 minut. Nutno dodat, že tato chyba je opravdu minimální a výstup byl akceptovatelný daleko dříve, učení však bylo automaticky zastaveno až po dosažení této chyby. Daleko jasnější je pak pohled na extrahované parametry, viz. Obrázek 26.



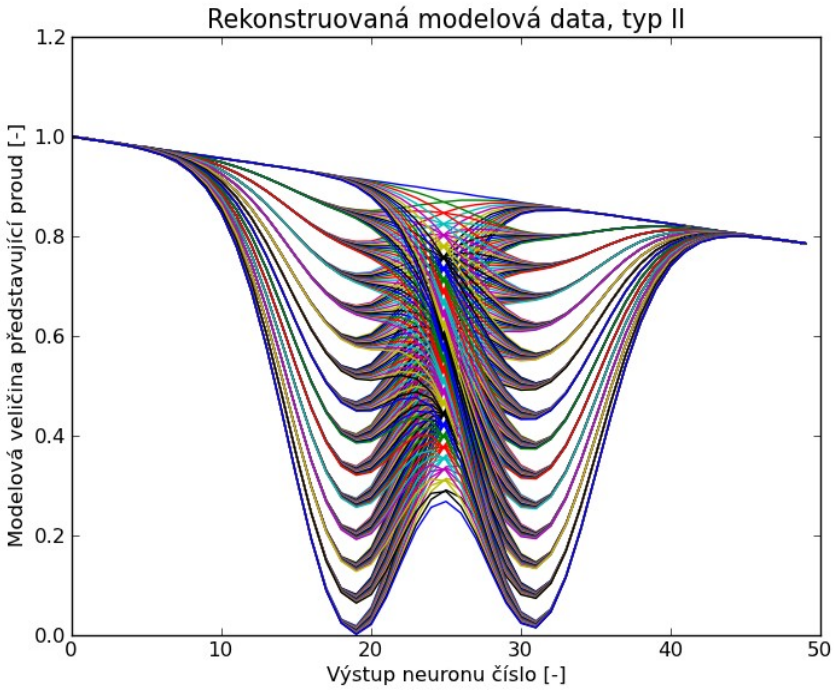
Obrázek 26: Vizualizace extrahovaných parametrů modelových dat s lineárními výstupy

Na tomto obrázku je zřejmá, téměř ideální plocha, na které leží mřížka parametrů. Pro ujasnění jsou naznačeny šipkami směry pohybů nutných pro změnu výšky píků. Směr šipky

I naznačuje nárůst parametru *A* a šipka *II* naznačuje nárůst parametru *B*. Pokud bychom tyto parametry chtěli vyhodnotit, stačí aplikovat afinní transformaci rotace a zkosení, tyto transformace bychom samozřejmě mohli navrhnout jako dvě lineární vrstvy neuronové sítě. Lze předpokládat, že deformace mřížky souvisí s lineárním posunem v modelových datech a s rozdílným rozložením píků. Jistě zde bude vhodné zmínit rozdíl v natočení a umístění extrahovaných parametrů v rovině krčku v porovnání s předchozí sítí. Toto je opět v souladu s předpokladem, že síť na začátku cvičení nemá naprosto žádné znalosti o extrahovaných parametrech a obecně o transformacích jenž své mapovací a demapovací vrstvy učí. Proto je možné, že v jedné verzi naučené sítě bude parametr prvního neuronu druhým a opačně. Navíc uvědomíme-li si komplexnost transformace, kterou mapovací a demapovací vrstvy provádí, je naprosto pochopitelné zrcadlení, otáčení a další transformace v rovině extrahovaných parametrů. Hlubším vyhodnocením se však budeme zabývat až v případě dalších modelových dat a následně reálných dat. Výšky generovaných píků známe a jejich spojitost s extrahovanými hodnotami je nyní zcela zřejmá. Využití lineární přenosové funkce u výstupních neuronů se tedy prokázalo jako výhodnější. Důvodem je schopnost snadno zrekonstruovat i nulové hodnoty křivek.

Lineární výstupní aktivační funkce, modelová data typ II

Tato data jsou navržena s ohledem na tvar reálných dat zpracovávaných v následujícím experimentu a snaží se simulovat jejich strukturu.

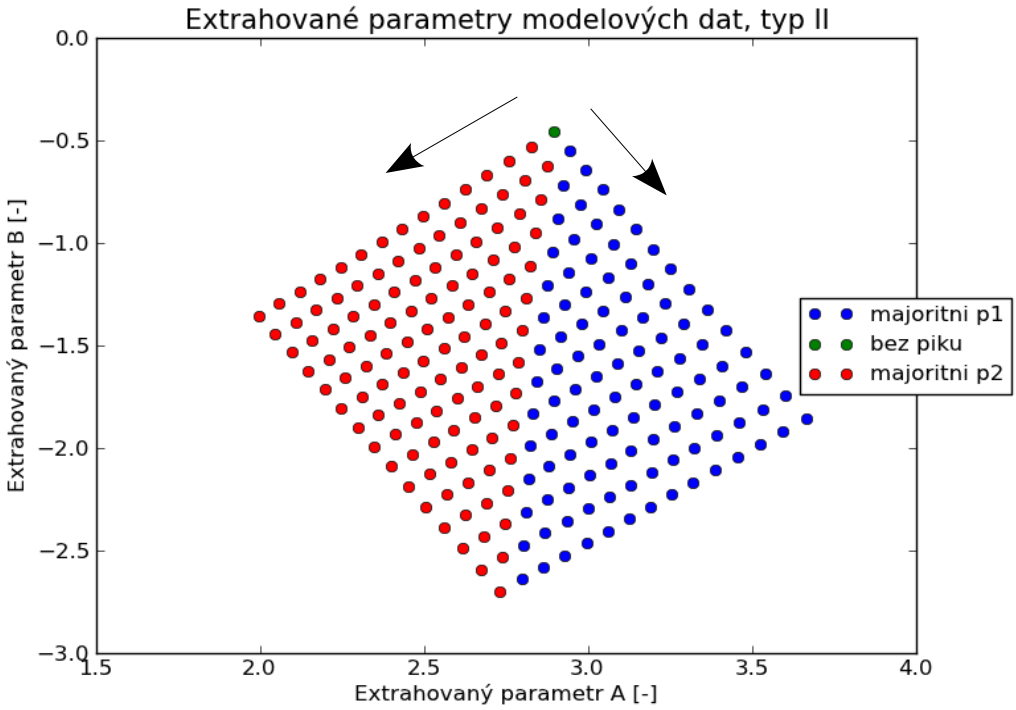


Obrázek 27: Vizualizace rekonstruovaných dat modelových dat typu II

Chyba rekonstruovaných dat vyjádřená pomocí střední kvadratické odchylky je

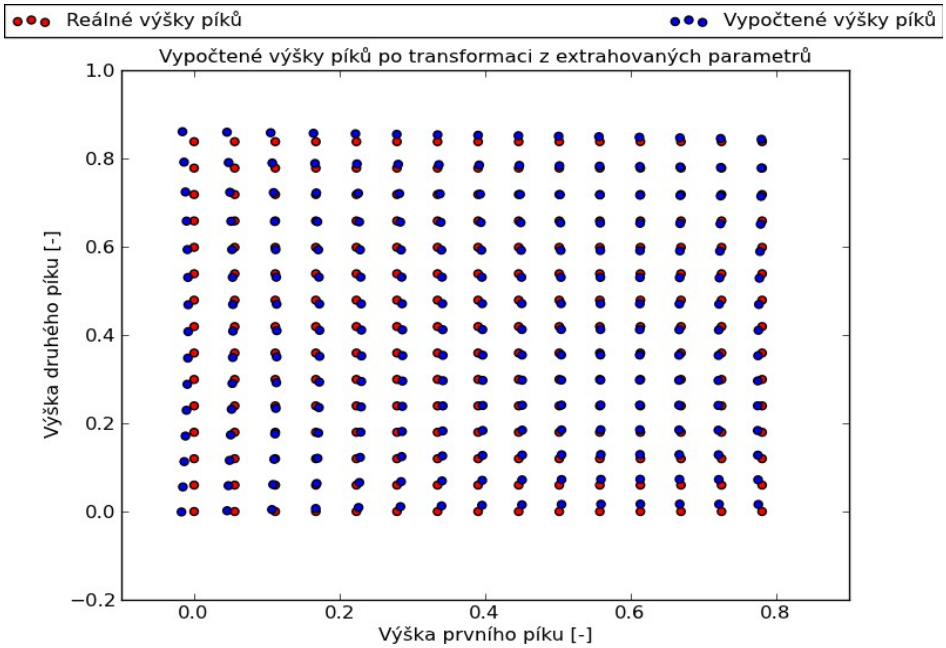
*E = 4.768*E-05. Rekonstrukce je v tomto případě opět vynikající. Rozprostření extrahovaných parametrů by tedy podle dřívějších poznatků mělo být opět ve dvourozměrné mřížce.

Rozložení parametrů (viz. Obrázek 28) odpovídá předpokladům. Pomocí šipek jsou znázorněny směry změn parametrů, které odpovídají nárůstu k nim přidružených píků. Natočení mřížky je opět jiné než v předchozích dvou experimentech. Lze tedy předpokládat, že jde opravdu o náhodné transformace související spíše s počáteční inicializací sítě. Pro tento typ modelových dat jsme se pokusili demonstrovat metodu, pomocí které je možné transformovat extrahované parametry přímo na hodnoty výšek píků v křivkách. Pro realizaci této transformace bylo nutné navrhnout jednoduchou síť se třemi vrstvami. Jak je z extrahovaných parametrů patrné, bylo zapotřebí tří affiních transformací. Jedna vrstva musí provést rotaci parametrů, druhá zkosení a třetí roztažení.



Obrázek 28: Extrahované parametry modelových dat s prolínajícími se píky

Extrahované parametry po transformaci sítí, která uskutečnila požadované affinní transformace jsou uvedeny na obrázku (Obrázek 29).

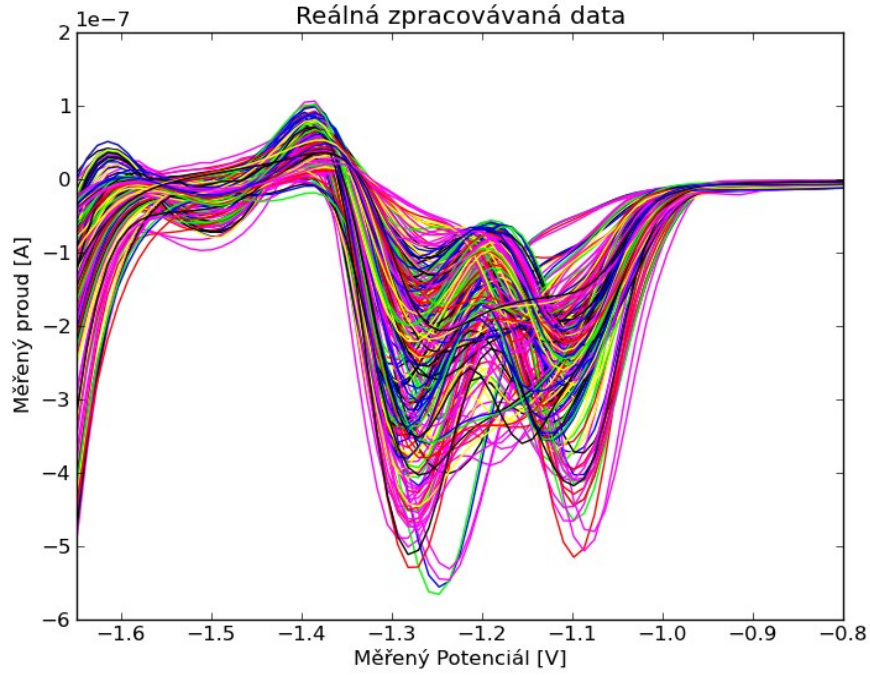


Obrázek 29: Odečtené výšky píků po transformaci vs. reálné výšky píků

Chyba takto odečtených výšek píků vyjádřená pomocí střední kvadratické odchylky je *E = 9,191e-05*. Je tedy jasné, že se jedná o velmi mocnou techniku vyhodnocování a klasifikování dat. Vyhodnocení je provedeno odečtením hodnot z os transformované mřížky. Pokud bychom chtěli křivky klasifikovat, je nutný předpoklad rozdílů výšek píků mezi druhy. Přesněji křivky stejného druhu by se měly shlukovat ve stejné části výsledné transformované mřížky a tvořit klastry. Vzhledem k tomu, že jsou výsledná data vynesena do roviny, je snadná i vizualizace a optická inspekce. Samotné rozložení reálných píků vůči vypočteným vykazuje zajímavé vlastnosti. Přibližně na souřadnicích *[0,6; 0,3]* , *[0,1; 0,7]* a *[0,15]* ; *[0,1]* si můžeme všimnout atraktorů, které k sobě přitahují a kolem sebe formují zvláštní gradienty. První jmenovaný má největší přitažlivost. Jsem toho názoru, že tyto toky modelují konečné zvlnění roviny, do které jsou extrahované parametry zobrazeny a které již nebylo možné dalším učením vyhladit.

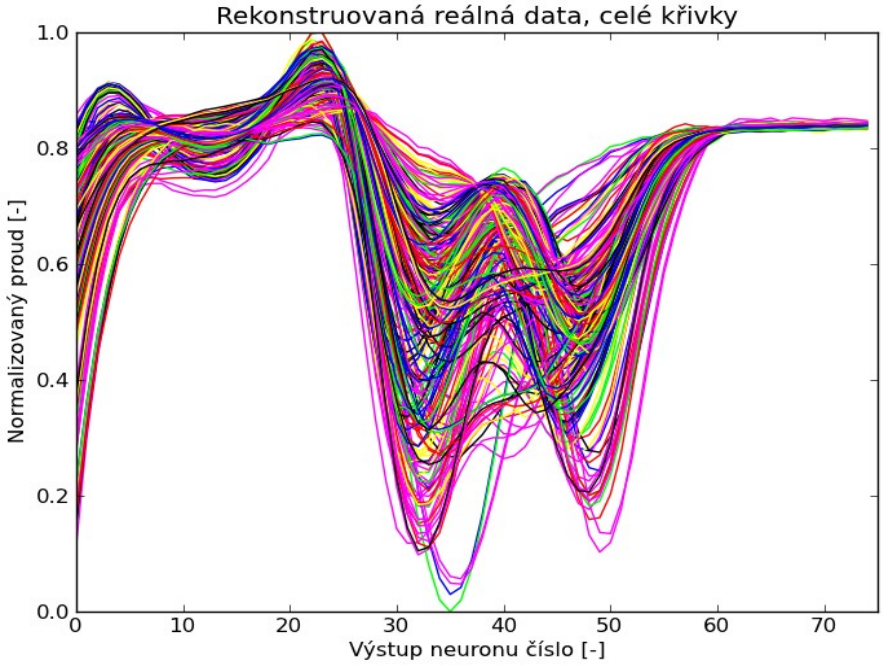
5.2.6 Reálná data, vzorky krví a tkání z tumorů

Jak vyplývá z předchozích experimentů, na testovacích datech zvolený model sítě funguje velmi dobře, podívejme se tedy na reálná data. Nejprve se podíváme jak taková data vypadají, v dalším bodě se blíže seznámíme s chybou sítě a podobou rekonstruovaných dat, poté rozebereme extrahované parametry. V tomto experimentu byla data navzorkována na 75 vzorků na křivku. V mapovací a demapovací vrstvě bylo využito 2 x 75 neuronů.



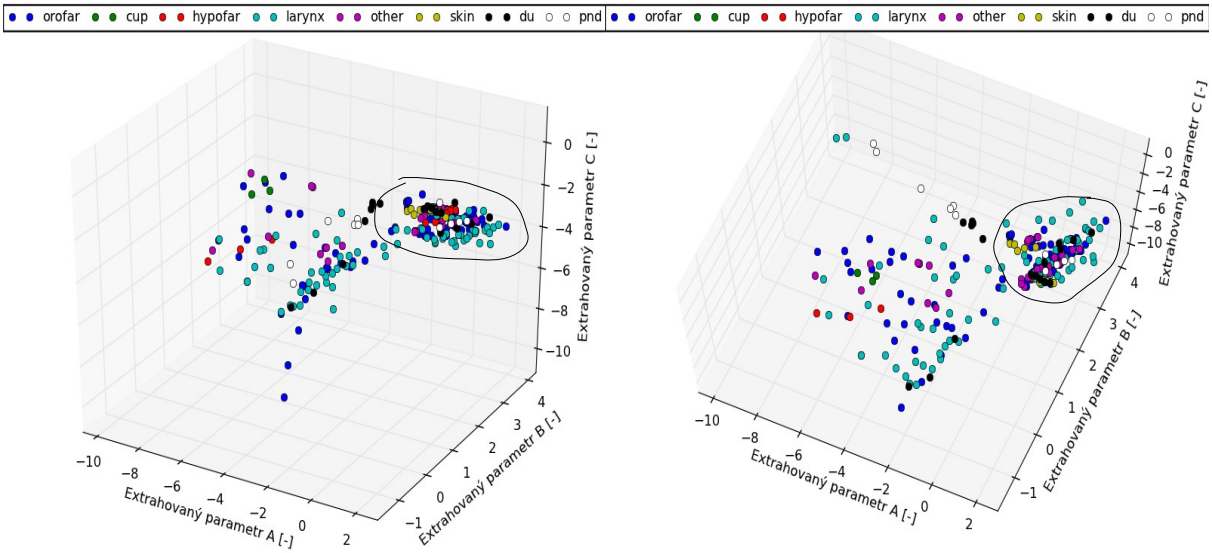
Obrázek 30: Celé křivky reálných analyzovaných dat

Ze zkušeností získaných na modelových datech lze očekávat šířku krčku tři neurony díky třem přítomným píkům. Pomocí „box-counting dimension“ jsme však došli opět k šířce 2. Pomocí metody pokus-omyl se ale následně ukázalo, že je opravdu nutné využít šířku 3, jinak je chyba rekonstrukce nepřijatelná. Následující obrázek zobrazuje rekonstruovaná data.



Obrázek 31: Rekonstruovaná data, celé křivky

Data vykazují neobyčejnou podobu s reálnými daty, chyba rekonstruovaných dat vyjádřená pomocí střední kvadratické odchylky je $E = 0.0096$. Vizuální inspekce extrahovaných parametrů je však v tomto případě složitější, neboť musíme použít 3D zobrazení. Učení této sítě trvalo půl dne, požadované minimální chyby nebylo dosaženo, takže se jedná opět o jedno z předběžných zastavení. Rozložení parametrů v prostoru (viz.Obrázek 33).

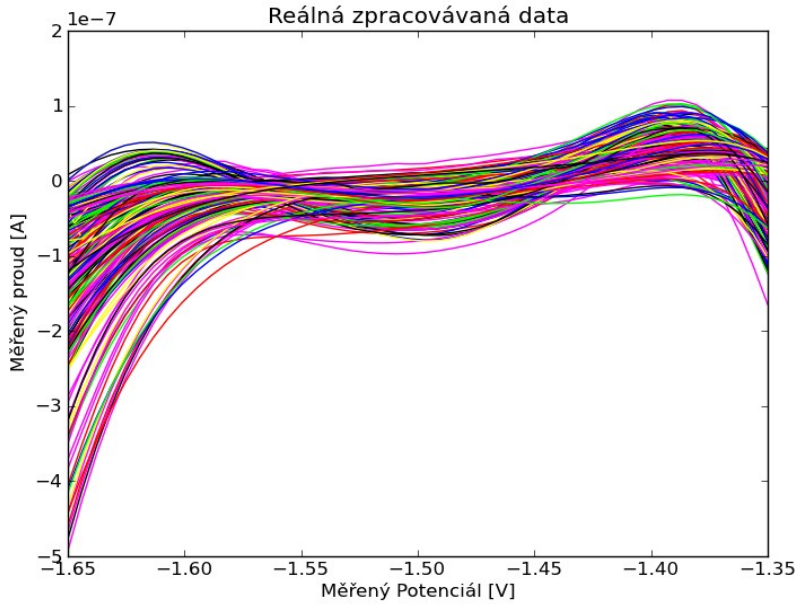


Obrázek 32: Grafy extrahovaných parametrů, 2 různé pohledy

Jak je naznačeno, v datech je identifikovatelný jediný výrazný klastr, rozborem jeho vlastností se však zabývat nebudeme a uskutečníme experiment na redukovaných ořezaných datech tak, aby byl dostačující dvoudimenzionální krček, snazší vizualizace a rozbor parametrů. Nejprve provedeme analýzu na nejdůležitějším píku Cat2, poté se pokusíme prozkoumat zbytek křivky.

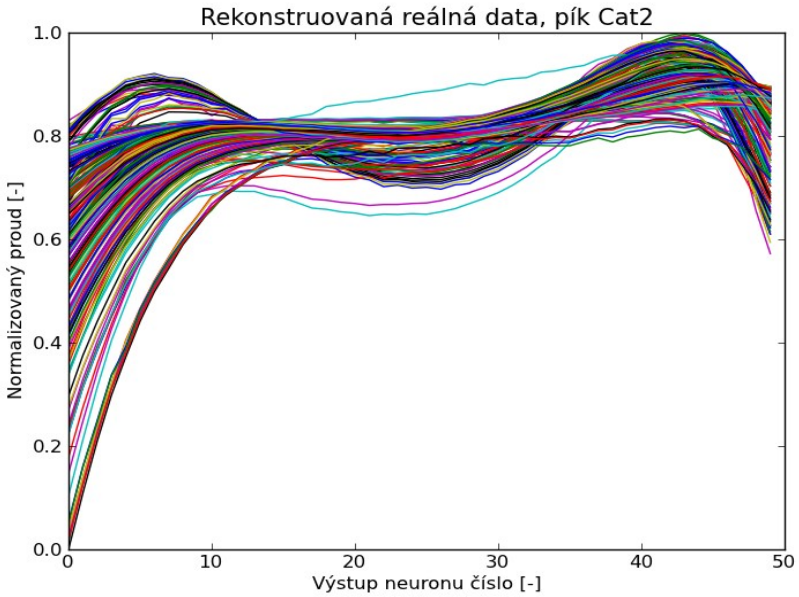
Analýza první části křivek

V analyzovaných datech je nejdůležitější první pík Cat2, který má přímou souvislost s koncentrací sledované látky, potencionálního nádorového markeru. Bylo tedy provedeno vyříznutí úseku v intervalu potenciálů *[-1,65; -1,35] V*. Data byla převzorkována na 50 vzorků. K analýze byla využita síť se dvěmi vrstvami o 50 neuronech jak v mapovací tak demapovací části. Tento experiment zabral více času, neboť z počátku jsme předpokládali, že dimenzionalita analyzovaných dat je 1. Kvůli velmi špatné rekonstrukci bylo tedy přikročeno k metodě pokus-omyl.



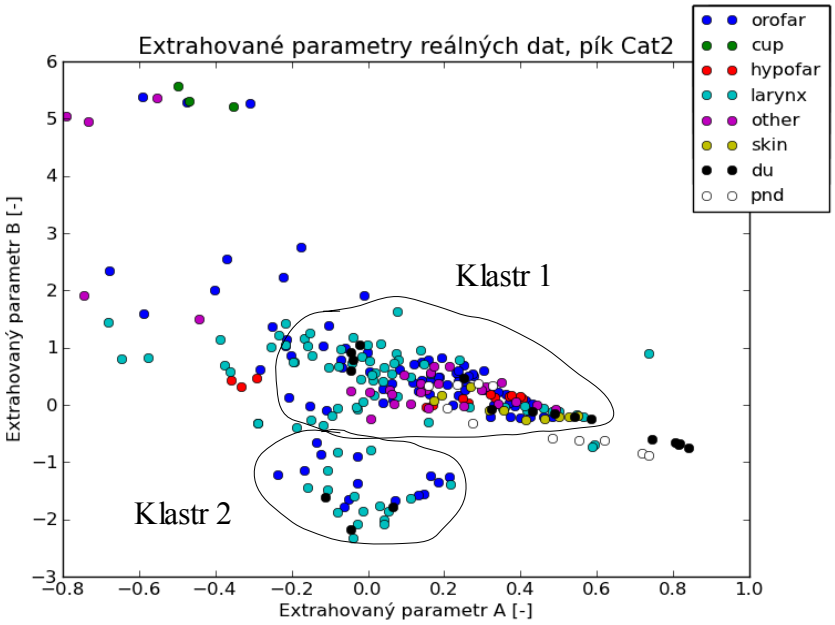
Obrázek 33: Reálná zpracovávaná data, výřez píku Cat2

Šířka krčku byla metodou pokus-omyl stanovena na 2 neurony. Tato skutečnost je pravděpodobně způsobena dvěma vlivy, které určují tvar křivek. Jedná se o náběh po vylučování vodíku na elektrodě a samotnou výšku píku Cat2. V porovnání s celkovými křivkami probíhalo učení této sítě delší dobu a oproti prvotním předpokladům byla dosažená minimální chyba vyšší.

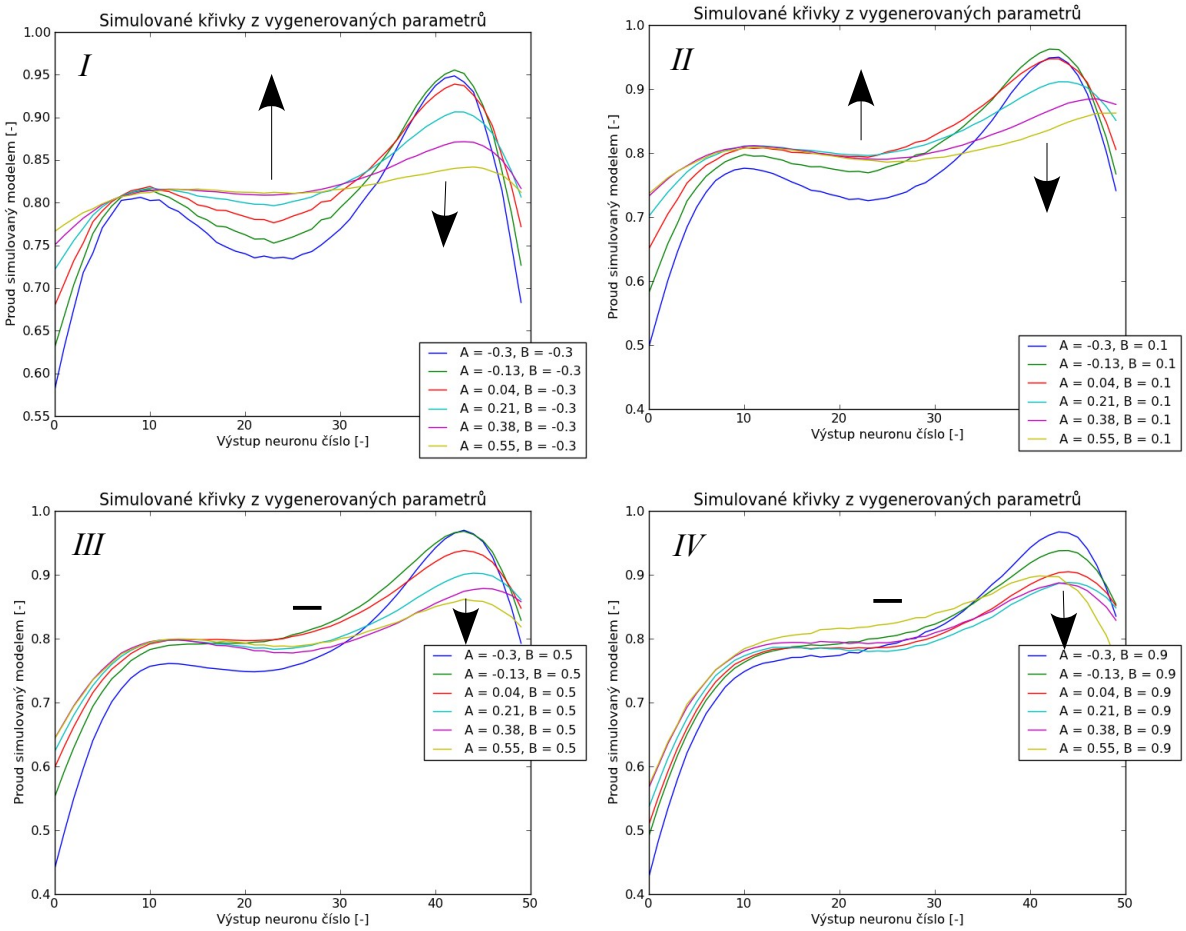


Obrázek 34: Rekonstruovaná data, analýza píku Cat2

Chyba rekonstruovaných dat vyjádřená pomocí střední kvadratické odchylky byla $E = 0.0113$. Obrázek 35 je zobrazením extrahovaných parametry výřezu píku Cat2.



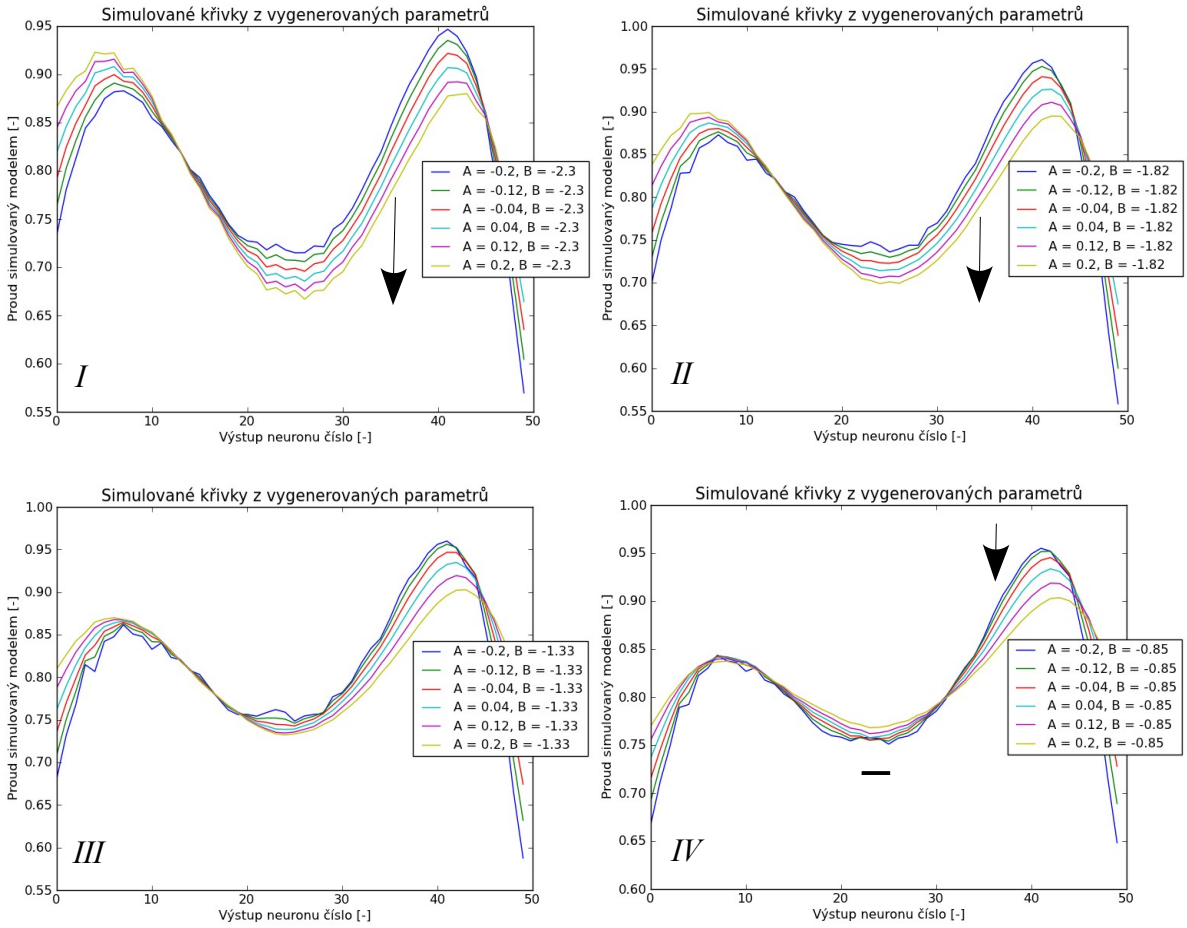
Obrázek 35: Extrahované parametry z analýzy píku Cat2



Obrázek 36: Křivky vygenerované rozmítáním parametrů ve sledovaném klastru 1. Každý graf obsahuje rozmítání parametru A při různé konstantní hodnotě parametru B.

Zařazení křivek do sledovaných kategorií není pomocí tvaru tohoto píku průkazné. Lze si však například povšimnout, že naprostá většina křivek ze skupiny larynx má parametr $A < 0,3$. Jiné závislosti jsou však obtížně pozorovatelné. V datech si můžeme vybrat dvě zajímavé oblasti, tak jak je naznačeno, a prozkoumat simulací parametrů důkladněji přibližné prototypy křivek, které obsahují.

Z pohledu smysluplnosti simulovaných křivek jsou nejdůležitější oblasti I a II, jelikož v ostatních oblastech se při vyšších hodnotách parametru A dostáváme do neobsazené části v prostoru extrahovaných parametrů. Čím vyšší je hodnota parametru B, tím horší je definice sledovaného píku Cat2. Z grafů je patrné, že rozmítání parametru B je zodpovědné za maximální výšku vlny vylučování vodíku. Tento jev má pak evidentně souvislost s výškou tohoto píku a vlnou předcházející druhou část celé křivky. Je tedy možné, že množství vyloučeného vodíku má souvislost s výškou sledovaného píku nebo obráceně, čím vyšší je pík a tedy koncentrace sledované látky (je prováděna akumulace na elektrodu), tím vyšší je vylučování vodíku. Parametr A pak nepřímo souvisí s výškou píku v oblasti daného parametru B. Pro automatické vyhodnocení výšky píku by v tomto klastru byl vhodný pouze



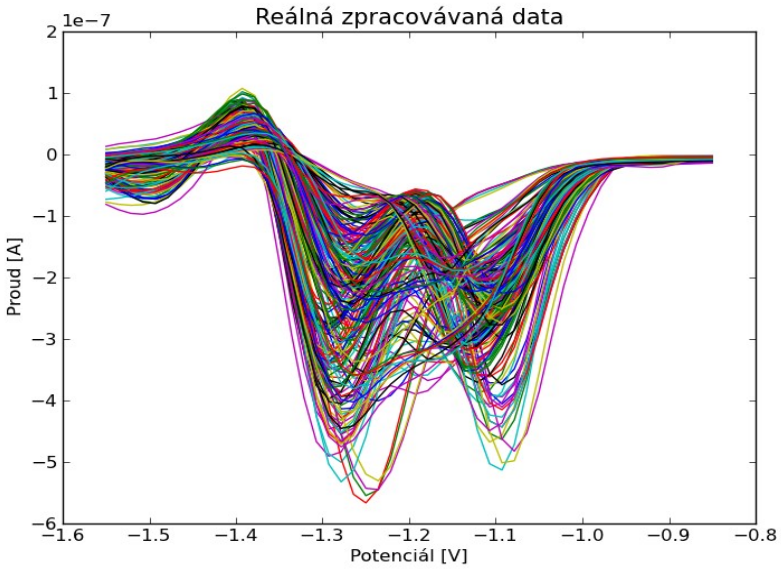
Obrázek 37: Křivky vygenerované rozmítáním parametrů ve sledovaném klastru 2. Každý graf obsahuje rozmítání parametru A při různé konstantní hodnotě parametru B.

spodní levý trojúhelník z rozprostřené mřížky simulovaných parametrů. To odpovídá skutečnosti, protože v tomto klastru v pravé horní části nejsou umístěny téměř žádné extrahované parametry.

Na první pohled je jasné, že klastr 2 obsahuje křivky s vyšším a lépe definovaným píkem Cat2. Parametr *B* se v tomto klastru projevuje na maximální výšce sledovaného píku. V oblastech *I* a *II* dochází ke snižování první patky píku zleva a zvyšování druhé patky v pravé části grafů. Tuto skutečnost lze interpretovat tak, že se parametr *A* projevuje rotací rekonstruované křivky po směru hodinových ručiček vzhledem ke středu křivky. V oblastech *III* a *IV* je pak první patka stabilní a parametr *A* nepřímo ovlivňuje téměř výhradně výšku píku pouze pomocí změny druhé patky. Tento klastr obsahuje jinou třídu sledovaných křivek, návrh automatického vyhodnocení výšky píku by zde pak bylo snadnější než v klastru předešlém, větší význam mají ale pravděpodobně provedené simulace a zobrazení prototypů křivek.

Analýza druhé části křivek reálných dat, nelineární výstup

Jelikož v první části dat nebyly nalezeny požadované souvislosti, byla provedena analýza druhé části křivky. Podoba dat po ořezu je zobrazena na následujícím obrázku. Tato část křivky nese majoritní část informace o měřených vzorcích, souvislosti jejích píků s mechanismy na elektrodě a měřeným vzorkem však dosud nejsou plně objasněny.

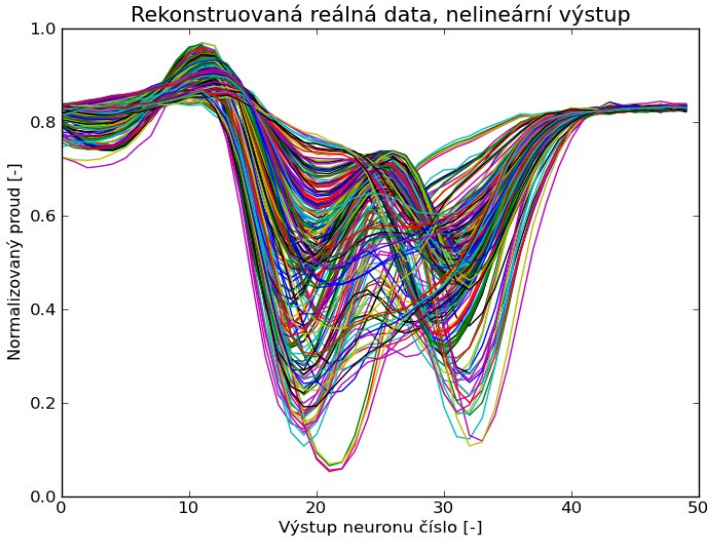


Obrázek 38: Reálná naměřená data

Data byla analyzována stejným modelem jako v prvním pokusu s modelovými daty, tedy dvě vrstvy o 50 neuronech v mapovací a demapovací vrstvě a lineární výstupní přenosová funkce, která se v tomto případě opět ukázala pomocí metody pokus-omyl jako výhodnější.

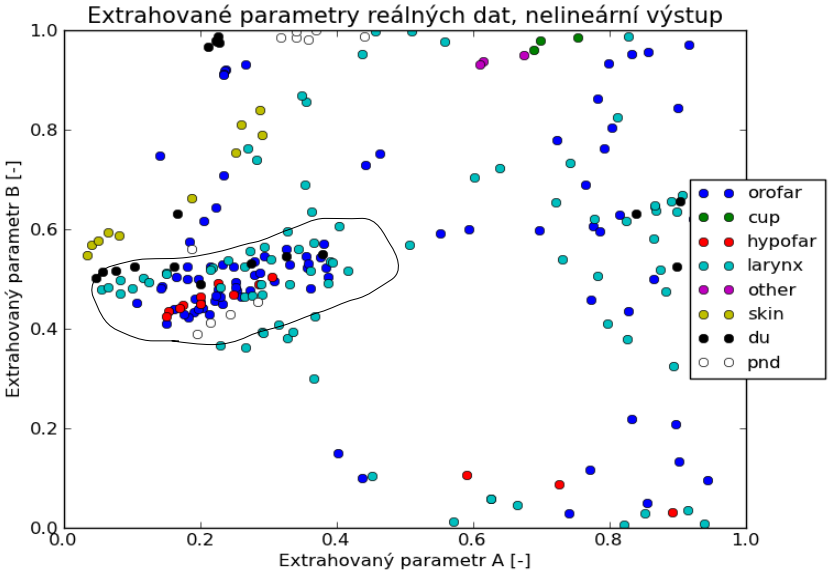
Na první pohled je patrné, že se nejedná o žádný ideální případ dvou superponovaných

píků na základní lineární nosné. Naopak se zde uplatňují různé posuvy a další nelinearity mimo hlavní důležité elektrochemické aktivity ovlivňující přítomnost a tvar samotných píků. Na následujícím obrázku se tedy podívejme jak dopadla rekonstrukce těchto dat po naučení neuronové sítě.



Obrázek 39: Reálná data rekonstruovaná při nelineárních přenosových funkcích výstupních neuronů

Data opět vykazují vynikající podobu s reálnými daty, chyba rekonstruovaných dat vyjádřená pomocí střední kvadratické odchylky je $E = 0.0119$. Chyba je o 4 řády vyšší než u modelových dat, což lze vzhledem k jejich podobě (rušení, chyby apod.) považovat za úspěch. Učení sítě trvalo přes noc. Vskutku je velmi překvapivé jakou kapacitu tato neuronová síť má a co je schopna se naučit. Daleko zajímavější a větší vypovídací hodnotu o samotných datech mají extrahované komponenty.



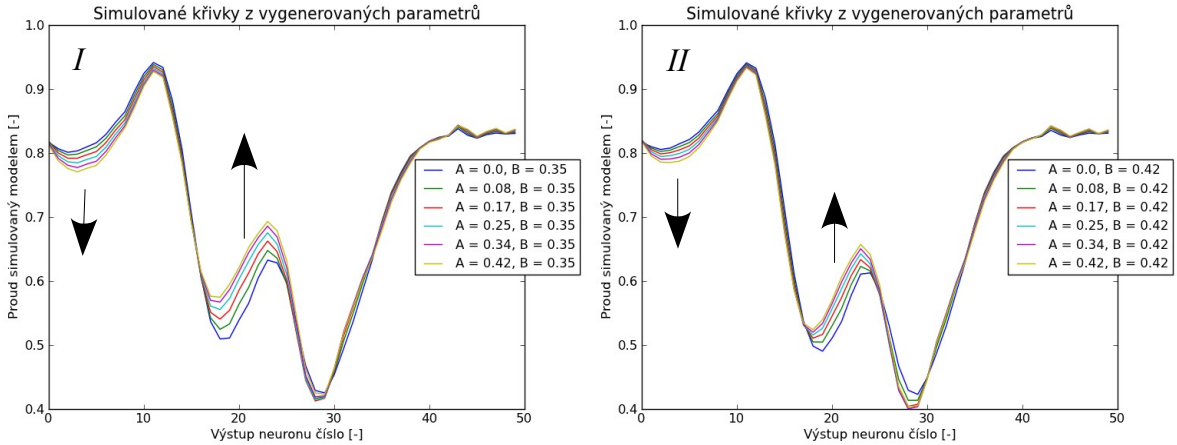
Obrázek 40: Extrahované parametry reálných dat, nelineární výstup

Na první pohled je zřejmé, že klasifikace do požadovaných skupin dle rozložení parametrů extrahovaných z druhé části křivky opět není průkazná. Nicméně i tak jsou z parametrů patrné některé další skutečnosti. V levé střední části grafu je označen jasně definovatelný klastr se spíše filamentární strukturou. Jsou v něm zastoupeny křivky ze skupin orofar, du, hypofar, larynx a pnd. Ostatní data netvoří definovatelné klastry, což může být způsobeno nedostatkem analyzovaných vzorků, případně lze všechna ostatní data zařadit do skupiny druhé. Je zcela jisté, že extrahované parametry o datech nesou téměř kompletní informaci. Abychom zjistili jakou, pojďme se podívat na následující experiment. Pokryjeme oblast roviny parametrů, ve které se pohybují zajímavé extrahované parametry body rozmístěnými ve vhodně zvolené mřížce a prozkoumáme jak tato rekonstruovaná data vypadají a co konkrétně parametry ovlivňují. Stejně tak jako u jiných statistických metod lze předpokládat, že některé analyzované vzorky jsou zavádějící a stejně tak jejich extrahované parametry. Při klasifikaci do skupin tedy očekáváme tvorbu logických shluků, které se dají separovat. Body, které jsou osamocené, nebo leží zcela mimo některé ze skupin nejsou většinou považovány za relevantní. Zaměříme se tedy blíže na místo, které tvoří jediný definovatelný shluk.

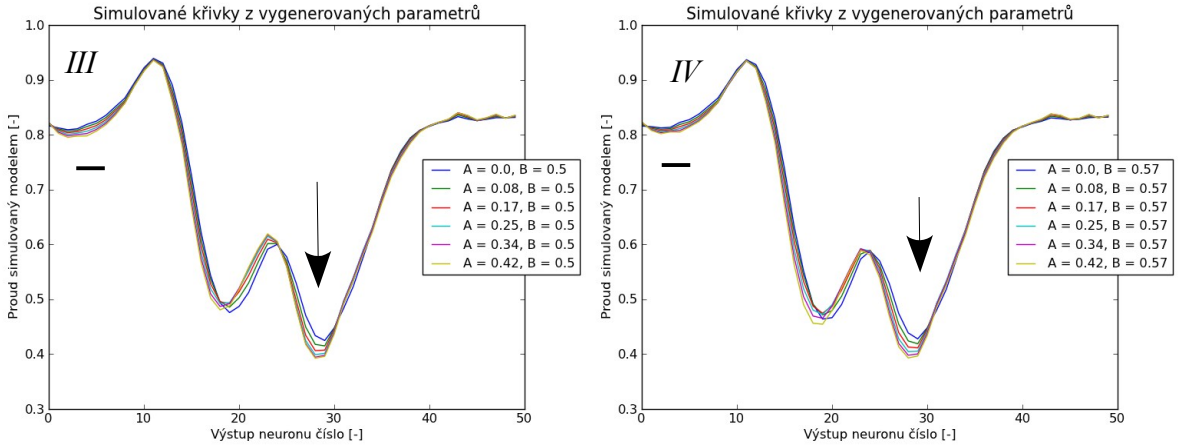
Konkrétně body umístěné v intervalech parametrů:

- *A = [0; 0,42] - B = [0,35; 0,57].*

Pro přehlednost rozdělíme křivky do čtyř grafů, v každém budeme sledovat romítání parametru A při konstantním parametru B. Nasimulované křivky ze sledované oblasti jsou následující:



Obrázek 41: Křivky vygenerované rozmítáním parametrů ve sledovaném klastru. Část I a II.

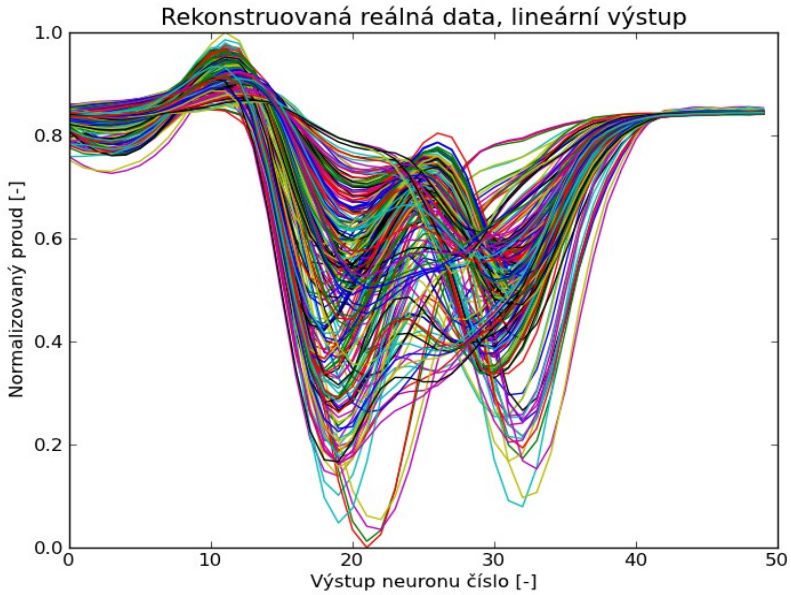


Obrázek 42: Křivky vygenerované rozmítáním parametrů ve sledovaném klastru. Část I a II.

Při hlubším prozkoumání grafů je možné si všimnout jistých pravidelností, které souvisejí s nárůstem či naopak poklesem sledovaných parametrů. Porovnáním vlivů parametru *B* na tvar křivek, tedy všech čtyř grafů naráz, je patrná nepřímá úměrnost nárůstu celkové mohutnosti tělesa sdružujícího oba píky. První pík se evidentně s druhým prolíná. Dále je zřejmé, že čím je vyšší parametr *B*, tím výrazněji je rozmítán pík Cat2. V oblastech *I* a *II* je parametr *A* nepřímo úměrný nárůstu píku Cat1 a přímo úměrný Cat2 tak, jak je naznačeno šipkami. V oblastech *III* a *IV* se pak vliv parametrů týká výhradně druhého píku RS_2Co. Je zřejmé, že na parametrech se projevují některé další transformace (rovina extrahovaných parametrů je zvlněná), a proto tedy vlivy parametrů na tvar křivek nejsou zcela separovány. Sestavováním převodních charakteristik pro tuto hustě obsazenou oblast se nebudeme zabývat, daleko významějším přínosem je opět modelování vlivů parametrů na tvary křivek a jejich další možné podrobnější studium. V případě potřeby přímých odečtů dat by bylo nutné provést daleko podrobnější zmapování roviny parametrů a navrhnout dodatečné transformace.

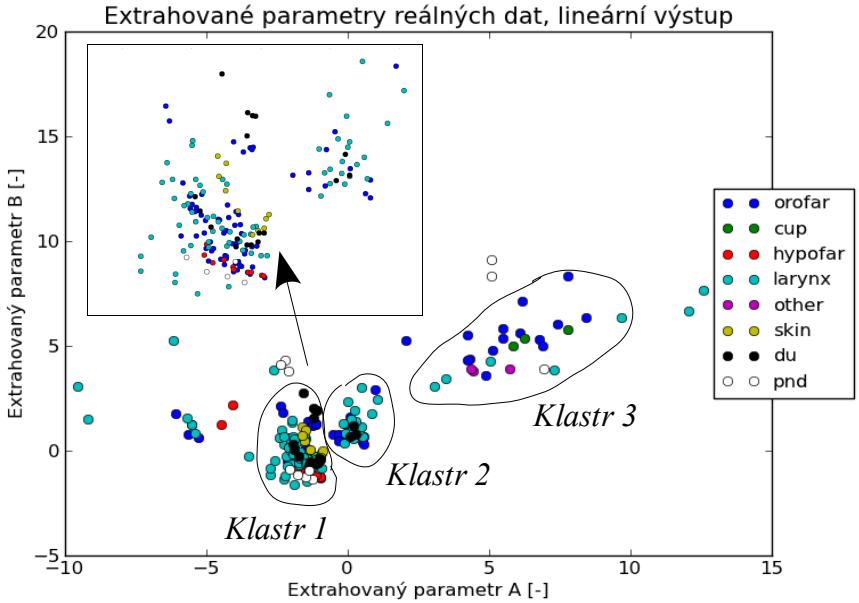
Analýza druhé části křivek reálných dat, lineární výstup

Lepšího rozklastrování bylo opět dosaženo při použití sítě s lineárními výstupy. Rekonstrukci dat touto sítí zobrazuje následující graf.



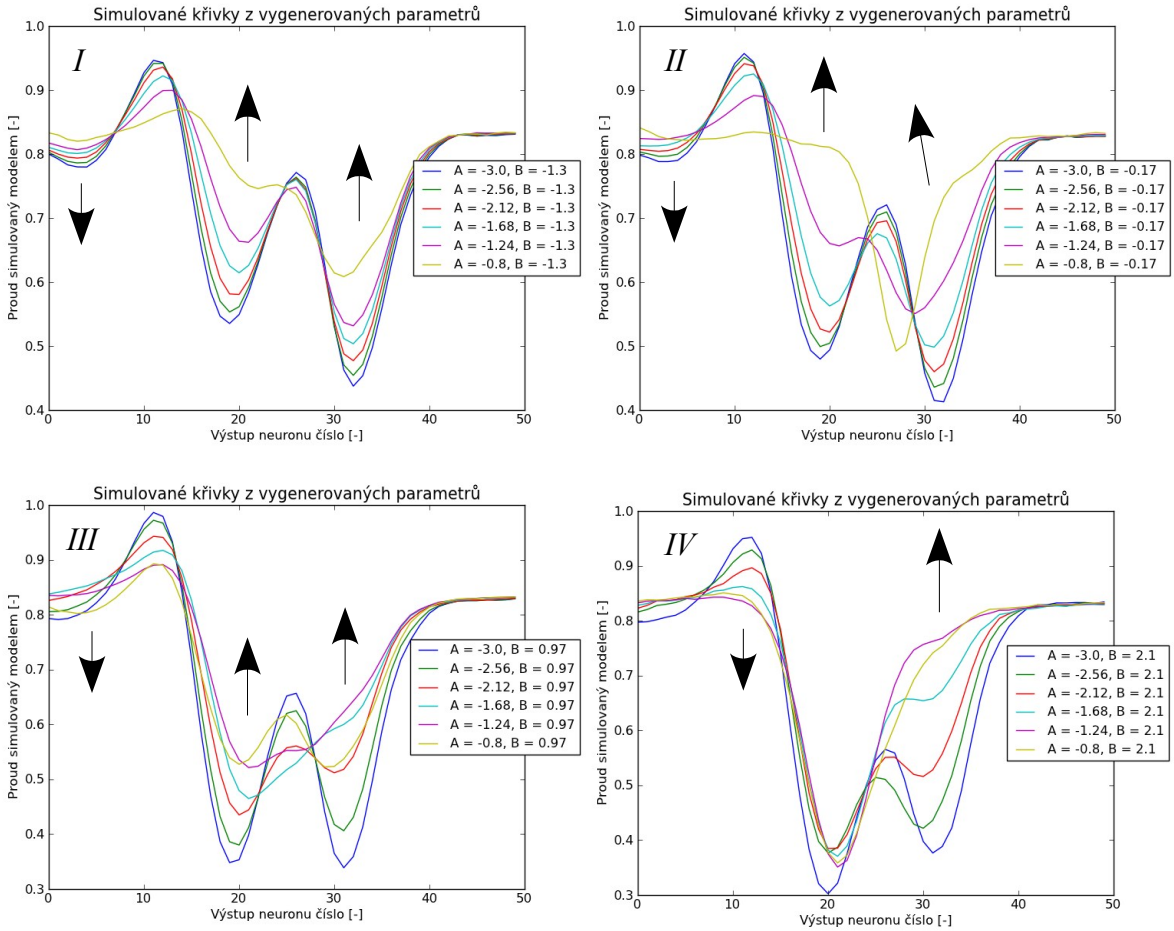
Obrázek 43: Reálná data rekonstruovaná při nelineárních přenosových funkcích výstupních neuronů

Rekonstruovaná data jsou opět vizuálně srovnatelná s reálnými daty, jejich chyba vyjádřená pomocí střední kvadratické odchylky je $E = 0.0126$. Extrahované parametry této sítě jsou:



Obrázek 44: Extrahované parametry reálných dat, lineární výstup

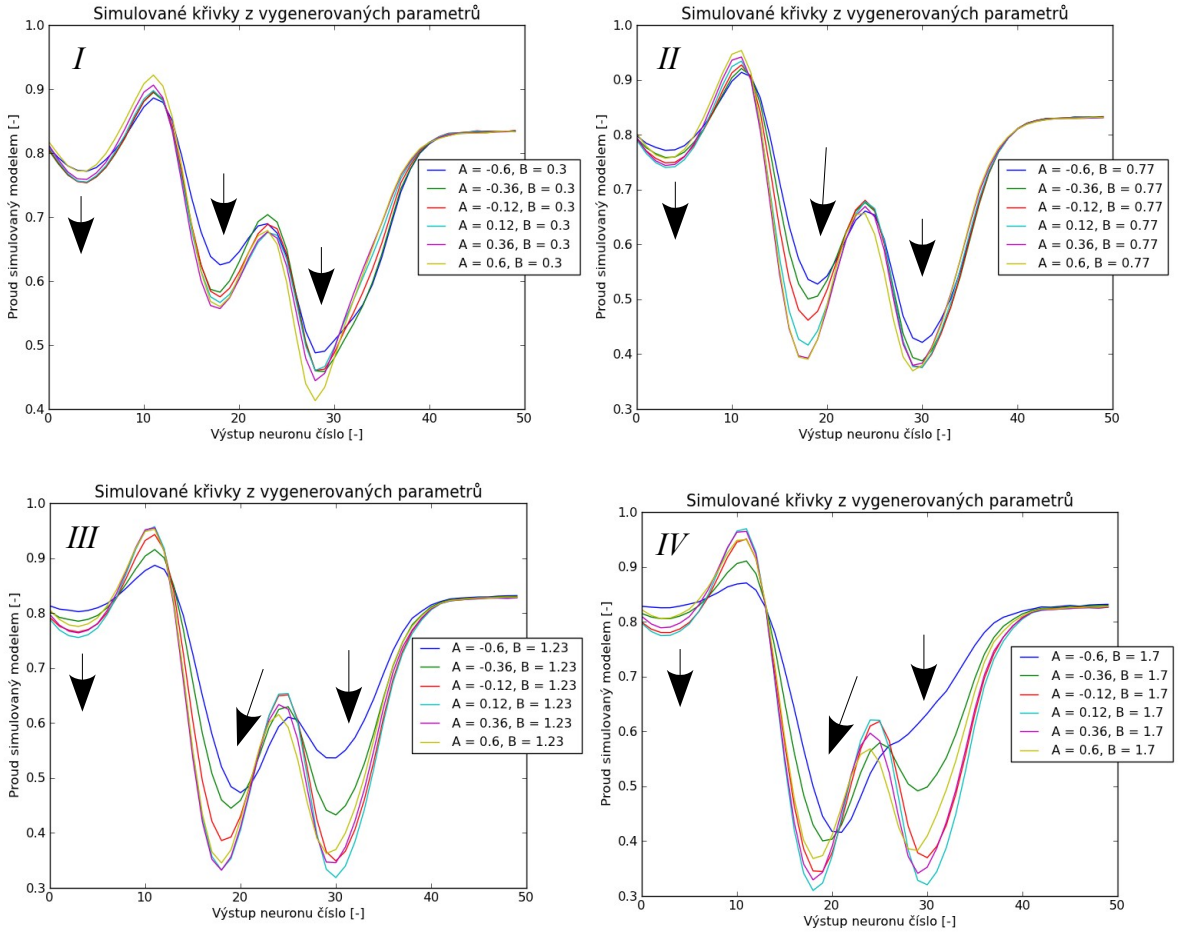
Chyba této sítě je porovnatelná s předchozí, rozložení extrahovaných parametrů je ale na první pohled jiné, což lze očekávat díky použití lineární výstupní přenosové funkce. Prozkoumáme-li však nejdominantnější klastry, zjistíme, že s předchozí sítí sdílí určitou podobu. Pro toto zjištění je nutné otočit rovinu parametrů této sítě o přibližně 90° proti směru hodinových ručiček. Je možné postřehnout, že vrstvy tvořené křivkami ze stejných skupin se v klastrech střídají ve stejném pořadí. Tato skutečnost souhlasí s teoretickými předpoklady, jelikož síť hledá souvislosti ve vnitřním uspořádání analyzovaných dat. Tato síť je však oproti té předchozí výhodnější, neboť jasně definuje dva klastry. Třetí označený klastr obsahuje méně objektů, ale vzhledem k malému počtu vstupních dat, se nemusí jednat o chybu. Prozkoumejme tedy, jaké prototypy křivek se vyskytují v jednotlivých klastrech.



Obrázek 45: Křivky vygenerované rozmítáním parametrů v oblasti klastru 1, rozdělené pro přehlednost do čtyř grafů. Každý graf obsahuje rozmítání parametru A při jiné konstantní hodnotě parametru B.

Nejprve zhodnotíme vliv parametru *B*, tedy porovnáním změn mezi jednotlivými grafy. Opět je zde patrná přímá úměra mezi výškou obou píků a hodnotou parametru *B*. Navíc dochází k výraznějšímu prolnutí obou píků a ke změně vlivu parametru *A* na výšku druhého píku. Souvislost hodnoty parametru *A* s tvarem křivek se projevuje nepřímou úměrností na výškách obou píků. V grafu *IV* je navíc v mezním případě patrný zánik druhého píku. První tři

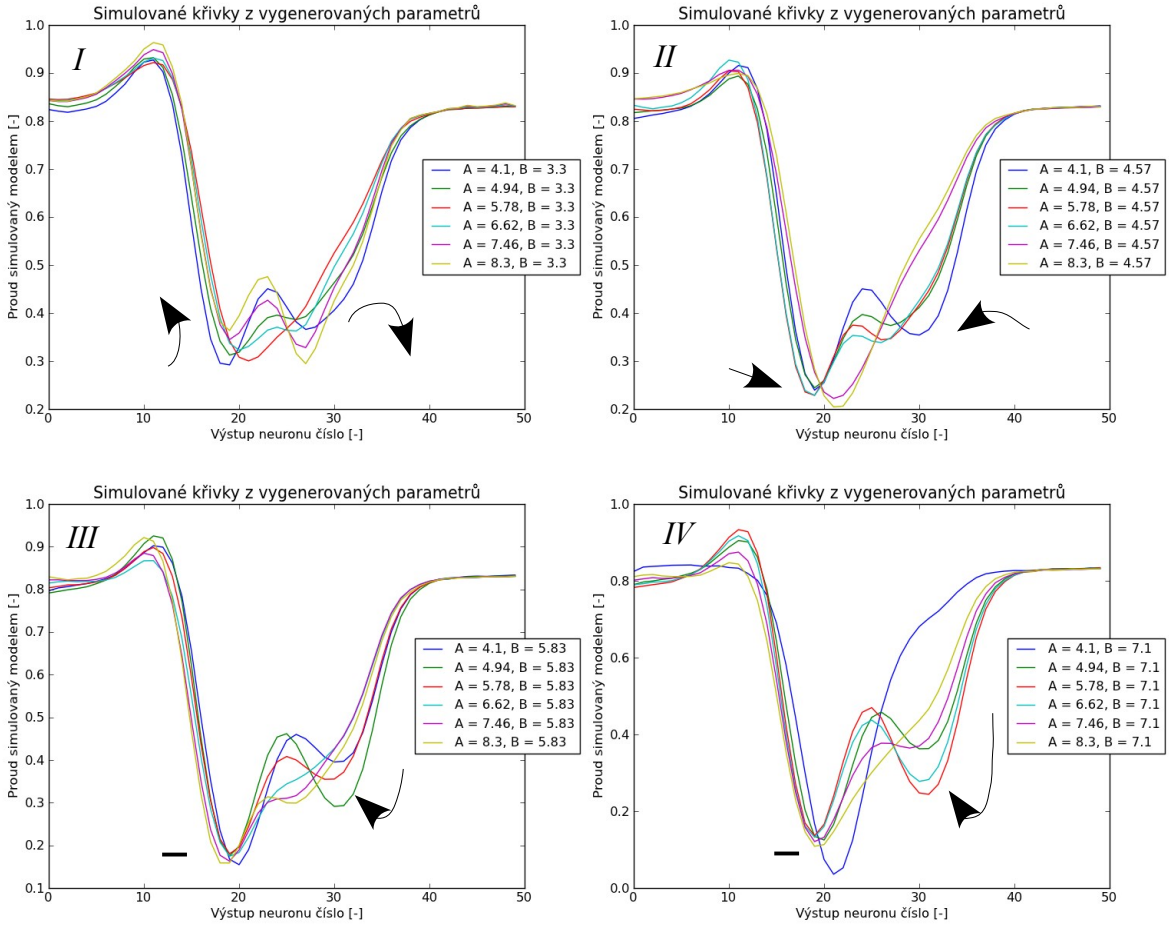
sledované oblasti vykazují podobnost s jediným sledovaným klastrem z nelineární obdoby experimentu. V oblasti *II* lze pozorovat absenci píků Cat2 i Cat1, což se projevuje výrazným posuvem třetího píku do středu křivky. V tomto sledovaném klastru se projevuje více vlivů, které ovlivňují tvar křivek, proto se získané výstupy opět hodí spíše k hlubší analýze souvislostí mezi píky než na přímé vyhodnocení. Stejně jako v předchozím experimentu je patrná vazba mezi jednotlivými píky. Čím lépe je definován pík Cat2, tím více vystupují oba následující píky. V případě, že Cat2 chybí, objevuje se pouze jeden pík, který se navíc posouvá do středu křivky. Vlna u patky píku Cat1 zřetelně ve všech oblastech ovlivňuje výšku píku RS_2Co. Dále se pojďme podívat na prototypy křivkek shluklých ve druhém klastru.



Obrázek 46: Křivky vygenerované rozmítáním parametrů v klastru 2.

Jak je patrné, jevy spojené s rozmítáním parametrů v klastru 2 jsou podobné těm z klastru předchozího. V tomto shluku je však vyšší část píku Cat2. Spojitost parametru *B* s tvarem křivek opět souvisí s celkovou plochou v níž se píky setkávají. Zásadně navíc ovlivňuje průměrnou výšku prvního píku. Se vzrůstajícím parametrem *B* vzrůstá dominance píku Cat1. V krajním případě grafů *III* a *IV* lze pozorovat posun píků na střed křivky v souvislosti s ústupem části píku Cat2. Parametr *A* na rozdíl od prvního klastru přímo ovlivňuje v celé

sledované oblasti výšku všech píků. Výška píku RS_2Co opět ve všech oblastech souvisí s náběhovou vlnou patky píku Cat1, která se navíc odvíjí od píku Cat2. Podrobnější zhodnocení těchto jevů bude nuné ponechat na odborné pracovníky.



Obrázek 47: Křivky vygenerované rozmítáním parametrů v klastru 3.

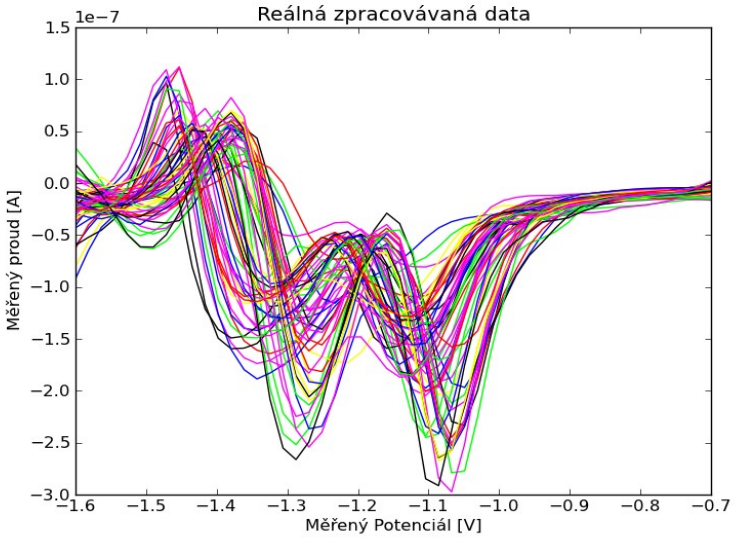
V této oblasti je na první pohled patrné, že se jedná o křivky s nejvyšším píkem Cat1. Parametr *B* se tedy opět promítá do celkového tělesa obou píků. Dále je v této oblasti nejméně definována část píku Cat2. Setkáváme se zde s novým jevem, tvorbou sedlových oblastí ve vrcholech píků. Tato oblast je tedy velmi zvlněná, což odpovídá tomu, že tento klastr je nejhůře definován, což odpovídá jeho nejnižší vypovídací hodnotě. Je rovněž patrné, že parametr *A* se projevuje na pozoci a výšce píku RS_2Co a tedy i jeho prolnutí do Cat1. Při pečlivé prohlídce si lze opět povšimnout, že výška druhého píku souvisí ve všech oblastech s výškou náběhové vlny patky píku Cat1. Tento jev je nejvíce patrný v oblasti *IV*.

Všechny tři klastry popisují různé fyzikální pochody. Jejich zhodnocením by mělo být možné detailně popsat, co se ve změřeném vzorku děje. Tyto analýzy však musí provést odborník. Návrhem sítě na klasifikaci zparametrizovanách křivek do skupin se v tomto případě nebudeme zabývat, jelikož by byl výsledek nepřehledný. V příštím experimentu se pak podíváme na návrh samotné klastrovací sítě, neboť výsledné rozložení parametrů je pro demonstraci jednodušší.

5.2.7 Reálná data, drůbeží krev

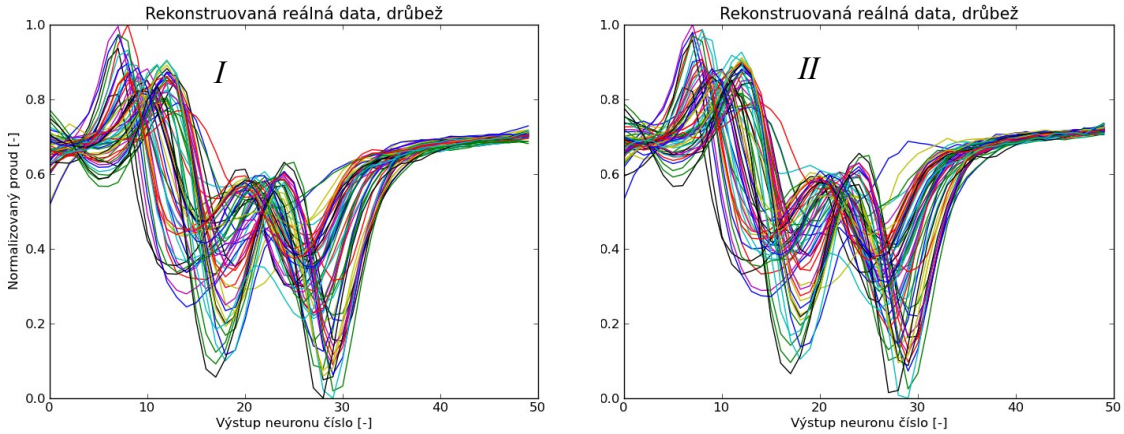
Analýza odlišných reálných dat

V posledním experimentu se podíváme na odlišná reálná data.



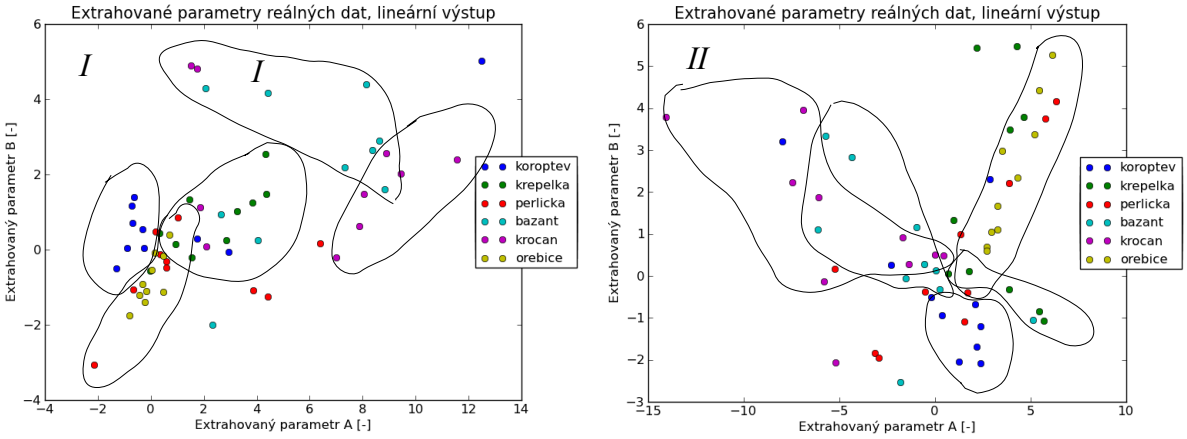
Obrázek 48: Data z měření krví různé drůbeže

K analýze byla využita síť s parametry použitými při analýze modelových dat. Vybrány byly dvě různé sítě s akceptovatelnými chybami. Jejich rekonstrukce jsou zobrazeny na následujícím obrázku.



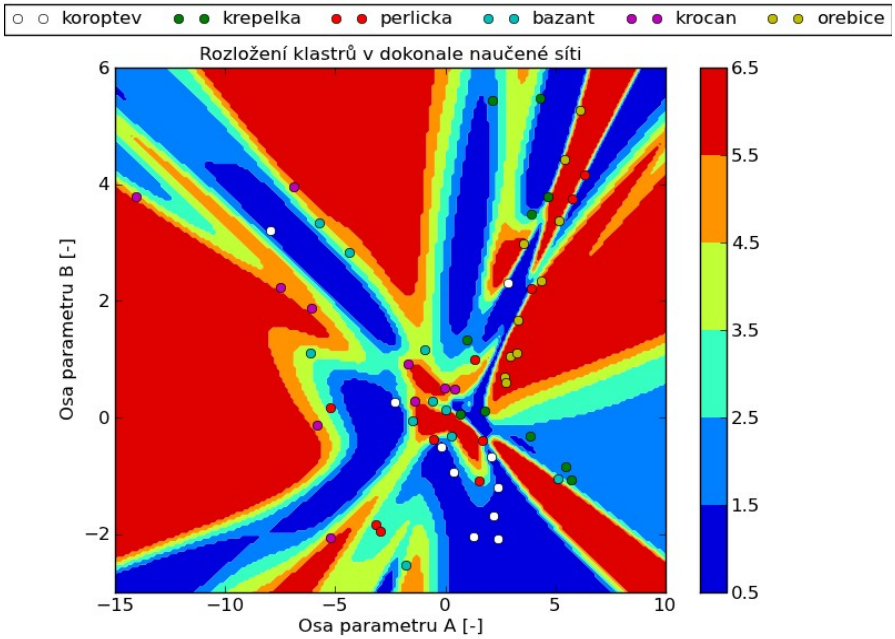
Obrázek 49: Rekonstruovaná data dvou různých sítí

Rekonstrukce sítě I má chybu vyjádřenou pomocí střední kvadratické odchylky $E = 0.0129$, chyba sítě II pak je $E = 0.02019$. Tyto chyby jsou porovnatelné s předchozími experimenty a jsou také velmi malé. Vyšší chyba sítě II je pak patrná při pouhé vizuální inspekci. Podívejme se tedy na souvislost a rozložení extrahovaných parametrů.



Obrázek 50: Rekonstruovaná data dvou různých sítí

V grafech extrahovaných parametrů jsou naznačeny možné podoby klastrů, které jednotlivé skupiny tvoří. Separace u první sítě *I* je výrazně lepší než u sítě *II*. Na obou případech jsou patrné filamentární klastry tvořené skupinou orebice. Tento klastr v sobě obsahuje značnou část křivek ze skupiny perličky. Tento klastr je navíc patrně propojen s perličkou a tedy je od sebe nejde separovat. V případě první sítě by byla úspěšnost zařazení 85% a 10% špatně zařazených případů. Nejlépe definovaný v obou případech je pak klastr obsahující koroptev kde je 70% případů zařazeno správně, v prvním případě dokonce žádný špatně. V případě bažantů a krocanů se opět setkáváme s prolínáním jejich klastrů, úspěšnost přibližně 70%, ale chyba 30%. Nejhůře definovatelný klastr tvoří skupina křepelek a to v obou případech. Jak je patrné, prolínání klusterů se nevyhneme. Analyzovaný set dat je bohužel velmi malý, což může být jednou z příčin chyb špatné separace extrahovaných parametrů. Prostor extrahovaných parametrů může být ve skutečnosti daleko rozlehlejší a momentální rozložení parametrů je v něm poskládáno příliš zhuštěně či zcela chybně. Obě sítě však sdílejí jistou podobu, otočením grafu *II* o 180° proti směru hodinových ručiček bychom dosáhli podobného rozložení parametrů ve výsledné rovině. Dalším učením sítě bychom se tedy pravděpodobně lepšího rozlišení nedočkali. Tyto extrahované parametry by bylo možné využít pro nacvičení sítě, která by prováděla klasifikaci zařazením do skupin. Při dostatečné komplexnosti sítě by mohla být úspěšnost velmi vysoká, vizuelní inspekce nám však ukazuje souvislosti mezi daty, které nenapovídají existenci jiných logických klastrů. Aby byla předchozí demonstrace jasná, podívejme se na jednu z možných síťí dokonale naučenou klasifikovat tato sledovaná data. Záměrně se podíváme na klasifikaci parametrů sítě *II*, kde jejich vizualizace vypovídá o možných tvarech klastrů méně.



Obrázek 51: Klastry, podle dokonale naučené sítě

Pro zařazení výstupů do skupin byla navržena jednoduchá metodika, extrahované parametry křivek ze skupin (koroptev, křepelka, perlička, bažant, krocan, orebice) byly sítí transformovány do číselných skupin (1, 2, 3, 4, 5, 6). Střední kvadratická odchylka výstupu sítě od požadovaných hodnot je $E = 2.397e\text{-}05$, což znamená, že všechny křivky byly klasifikovány dobře. Při učení sítě nebyl zařazen žádný parametr do testovacího ani vyhodnocovacího setu. To, co přesně Obrázek 43 naznačuje, je povrch transformační funkce, jenž zobrazuje extrahované parametry do skupin. Například pro skupinu koroptev = 1 leží všechny parametry v tmavě modré oblasti atd. Mnoho křivek leží správně zařazeno ve své úrovni na příkrých hranách, jejich zařazení je tedy sice správné, ale nelze předpokládat, že jsou v těchto oblastech klastry smysluplné. Zvlnění po stranách hranic klastrů je pak způsobeno zaokrouhlováním simulované mapy. Z tohoto pokusu je patrné, že snaha o naučení neuronové sítě tak, aby řadila analyzované hodnoty do klastrů, které navrhneme, není vhodné. Neuronové sítě sice mají kapacitu a možnost vytvořit velmi komplexní povrchy, ale bohužel je nutné aby v analyzovaných datech byl předem známý systém. K nalezení vnitřních vazeb mezi analyzovanými daty se jako velmi vhodné ukazují autoasociativní neuronové sítě. Po takto komplexním návrhu, který byl v této práci předveden, je možné navrhovat sítě seznalostí jejich transformací, vyhodnocovat pomocí extrahovaných parametrů, dále klasifikovat a zpracovávat.

6 Závěr

V této práci jsme se odchýlili od původního záměru získat co možná nejlepší výsledky při zařazování analyzovaných vzorků do stanovených skupin. V průběhu analýz bylo totiž zjištěno, že skupiny, do kterých jsme se snažili analyzovaná data zařadit, nejsou vhodné. U posledního experimentu bylo navíc naznačeno, že je možné vycvičit dostatečně komplexní neuronovou síť, která zařadí data do libovolných skupin a to i přes to, že jejich rozložení v prostoru netvoří příliš smysluplné klastry. Snaha o zařazení do předem navržených skupin bez hlubšího porozumění analyzovaným datům a jejich maximální simplifikaci se ukázala jako chybná.

Náše implementace neuronové sítě je velmi efektivní a zkracuje časy potřebné pro učení velmi rozsáhlých neuronových sítí. Vývoj v jazyku Python se ukázal jako velmi rychlý a výhodný. Díky využití jazyka OpenCL a výpočetní síly GPU navíc obcházíme ztrátu výkonu, kterou trpí implementace neuronových sítí zpracovávající data přímo na CPU hostitele a při učení tak systém ztrácí mnoho prostředků pro další práci. Dokonce se ukázalo, že je možné v naší implementaci na jednom systému velice efektivně učit více sítí paralelně. V porovnání s Matlabem bylo naučení rychlejší, i když v procesu učení je Matlab stabilnější, naše sít vyžaduje mnoho mutací vah k nalezení řešení. Námi realizovaná neuronová síť by mohla být dobrým základem pro další vývoj a práci. Bezesporu jsou nutné další optimalizace jak kódu jader, kterých by ideálně mělo být méně, ale také by bylo vhodné dále zmenšit a optimalizovat počet přesunů dat mezi pamětí hostitele a GPU.

Zvolený model neuronové sítě se na testovacích datech ukázal jako velmi vhodný. Absence potřeby návrhu výstupních dat je významnou výhodou tohoto modelu neuronové sítě. Při návrhu výstupních dat dochází k vytváření sítí, které nemají žádné hlubší znalosti o analyzovaných datech a jak bylo naznačeno, jejich rozhodovací regiony nemusí být smysluplné. Pro námi zvolený model neuronové sítě je však nutný kvalitní a dobře navržený set dat. Zmiňme téměř perfektní zparametrizování modelových křivek o 50 vzorcích a jejich téměř ideální rozložení do výsledné struktury, která po navržení jednoduché transformace umožnila jednoznačné určení výšky píku. Na reálných datech je situace jiná, extrahované parametry sice tvoří klastry ale neseskupují data tak, jak bychom chtěli. Nicméně analýzou hustějších oblastí byly modelovány zajímavé jevy. V případě druhé analyzované části křivek s píky Cat1 a RS_2Co by vztahy mezi modelovanými křivkami mohly přinést objasnění dosud neznámých souvislostí a mechanismů. Schopnost klasifikace do uvažovaných skupin se ukázala jako neprůkazná, což jistě souvisí také s malým setem dat určeným k analýze. Co se týče přímého vyhodnocení výšek píků z naměřených dat, nejsou tato elektrochemická data bez předchozí úpravy vhodná. A to nejen díky závislosti píků na všech parametrech, ale i množství šumu, nestálostí a absencí více uzlových bodů tak jako u modelových dat. Tohoto poznatku by bylo vhodné využít v případě budoucích elektrochemických měření. Bylo by

vhodné provést analýzu na datech z obdobného pokusu, avšak od začátku přímo s ohledem na podobnou analýzu neuronovými sítěmi. Pomocí optimalizace měřící metody a vhodného sestavení učicího setu dat na tento způsob vyhodnocování by pravděpodobně bylo možné dosáhnout vynikajících výsledků a to ať už při jednoduché klasifikaci, tak i při samotném automatickém vyhodnocování píků z křivek. Ušetření času, který je nutný pro manuální vyhodnocení by bylo neocenitelné. Po naučení podobné neuronové sítě je navíc možné její transformaci provádět velmi rychle i na dnešních běžně dostupných mikrokontrolerech, což by bez problémů umožnilo výrobu přenosných zařízení, schopných naprosto přesně zpracovat naměřená data přímo v terénu či u koncového zákazníka.

7 Seznam použitých zdrojů

[1] "Přístroje PalmSens" [cit. 2011-25-5] Dostupné z WWW: http://www.palmsens.com/

[2] "Přístroje Metrohm Autolab."[cit. 2011-25-5] Dostupné z WWW: http://www.metrohm-autolab.com/

[3] J.M. Santos, "Data classification with neural networks and entropic criteria," University of Porto, 2007.

[4] R.O. Duda, P.E. Hart, and D.G. Stork, *Pattern Classification*, Wiley-Interscience, 2000.

[5] S. Abbasi, K. Khodarahmiyan, and F. Abbasi, "Simultaneous determination of ultra trace amounts of lead and cadmium in food samples by adsorptive stripping voltammetry," *Food Chemistry*, vol. 128, Sep. 2011, pp. 254-257.

[6] "Curse of dimensionality - Wikipedia, the free encyclopedia." [cit. 2011-25-5] Dostupné z WWW: http://en.wikipedia.org/wiki/Curse_of_dimensionality

[7] I.A. Basheer and M. Hajmeer, "Artificial neural networks: fundamentals, computing, design, and application," *Journal of Microbiological Methods*, vol. 43, Dec. 2000, pp. 3-31.

[8] P. Baldi and Sã¸. Brunak, *Bioinformatics: The Machine Learning Approach, Second Edition*, The MIT Press, 2001.

[9] "Neural network - Wikipedia, the free encyclopedia." [cit. 2011-25-5] Dostupné z WWW: http://en.wikipedia.org/wiki/Neural_network

[10] "The Biological Neuron." [cit. 2011-25-5] Dostupné z WWW: http://www.virtualventures.ca/~neil/neural/neuron-a.html

[11] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological Review*, vol. 65, 1958, pp. 386-408.

[12] S. Theodoridis and K. Koutroumbas, *Pattern Recognition, Second Edition*, Academic Press, 2003.

[13] C.R. Rao and H. Toutenburg, *Linear Models: Least Squares and Alternatives*, Springer, 1999.

[14] "Learning in TLUs." [cit. 2011-25-5] Dostupné z WWW: http://www.cs.bham.ac.uk/~jlw/sem2a2/Web/LearningTLU.htm

[15] B.-H. Juang and S. Katagiri, "Discriminative learning for minimum error classification [patternrecognition]," *Signal Processing, IEEE Transactions on*, vol. 40, Dec. 1992, pp. 3043-3054.

[16] J.P.M. de Sá, *Pattern Recognition: Concepts, Methods and Applications*, Springer, 2001.

[17] B. Kröse, B. Krose, P. van der Smagt, and P. Smagt, "An introduction to Neural Networks," 1996.

[18] A.K. Jain, Jianchang Mao, and K.M. Mohiuddin, "Artificial neural networks: a tutorial," *Computer*, vol. 29, Mar. 1996, pp. 31-44.

[19] G. Zhang, B. Eddy Patuwo, and M. Y. Hu, "Forecasting with artificial neural networks:: The state of the art," *International Journal of Forecasting*, vol. 14, Mar. 1998, pp. 35-62.

[20] P.S. Curtiss and P.C.C. Engineers, "NEURAL NETWORK BASICS FOR USE IN BUILDING MECHANICAL SYSTEMS."

[21] M.H. Hassoun, *Fundamentals of artificial neural networks*, MIT Press, 1995.

[22] B. Cheng and D.M. Titterington, "Neural Networks: A Review from a Statistical Perspective," *Statistical Science*, vol. 9, Feb. 1994, pp. 2-30.

[23] "Delta rule - Wikipedia, the free encyclopedia." [cit. 2011-25-5] Dostupné z WWW: http://en.wikipedia.org/wiki/Delta_rule

[24] D. Svozil, V. Kvasnicka, and J. Pospichal, “Introduction to multi-layer feed-forward neural networks,” *Chemometrics and intelligent laboratory systems*, vol. 39, 1997, p. 43–62.
[25] M.A. Kramer, “Nonlinear principal component analysis using autoassociative neural networks,” *AIChE journal*, vol. 37, 1991, p. 233–243.
[26] J.W. Hines, R.E. Uhrig, and D.J. Wrest, “Use of Autoassociative Neural Networks for Signal Validation,” *Journal of Intelligent and Robotic Systems*, vol. 21, Feb. 1998, p. 143–154.
[27] E.C. Malthouse, “Limitations of nonlinear PCA as performed with generic neuralnetworks,” *Neural Networks, IEEE Transactions on*, vol. 9, Jan. 1998, pp. 165-173.
[28] M. Marseguerra and A. Zoia, “The autoassociative neural network in signal analysis: I. The data dimensionality reduction and its geometric interpretation,” *Annals of Nuclear Energy*, vol. 32, Jul. 2005, pp. 1191-1206.
[29] “The Khronos Group: Open Standards, Royalty Free, Dynamic Media Technologies.”
[30] Nvidia, “OpenCL Programming Guide for the CUDA Architecture,” Aug. 2010.
[31] Khronos OpenCL Working Group, “The OpenCL Specification,” Dec. 2008.
[32] “OpenCL – Wikipedie.” [cit. 2011-25-5] Dostupné z WWW: http://cs.wikipedia.org/wiki/OpenCL
[33] “PyCUDA: Even Simpler GPU Programming with Python. Nvidia GPU Technology Conf.”
[34] A. Klöckner, “Scripting GPUs with PyOpenCL,” Jun. 2010.

8 Seznam použitých zkratek a symbolů

ANN	Umělé neuronové sítě
AANN	Autoasociativní neuronové sítě
NLPCA	Nelineární analýza hlavních komponent
LMS	Metoda nejmenších čtverců
purelin	Lineární aktivační funkce neuronu
logsig	logistická sigmoidní aktivační funkce neuronu
tansig	Tangenciální sigmoidní aktivační funkce neuronu
hardlim	Prahová aktivační funkce neuronu
OpenCL	Open computing language
API	Aplikační programovací rozhraní
CPU	Central processing unit
GPU	Graphics processing unit
PC	Osobní počítač
ω_{ij}	Označuje jednu váhu (synapsi) neuronové sítě
$\boldsymbol{\omega}$	Označuje vektor vah
γ, η	Označují konstanty učení