VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2018 Bc. Jiří Zmeškal

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

EXTRÉMNÍ UČÍCÍ SE STROJE PRO PŘEDPOVÍDÁNÍ ČASOVÝCH ŘAD

EXTREME LEARNING MACHINES FOR TIME SERIES PREDICTION

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. Jiří Zmeškal

AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. Radim Burget, Ph.D.

SUPERVISOR

BRNO 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

Student: Bc. Jiří Zmeškal ***ID:*** 167729
Ročník: 2 ***Akademický rok:*** 2017/18

NÁZEV TÉMATU:

Extrémní učící se stroje pro předpovídání časových řad

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s nástroji ND4J, které umožňují základní maticové operace nad vektory, maticemi a tensory s použitím akcelerace nad GPU procesory. Seznamte se s problematikou extrémních učících se strojů (pomocnou CPU implementaci poskytne školitel). Implementujte algoritmus extrémního učícího se stroje s výpočtů na GPU. Změřte zrychlení algoritmu v závislosti na charakteru vstupních dat. Dosažené výsledky vhodně prezentujte a popište. Seznamte se s problematikou sítí s ozvěnou stavu (echo state network) a nástroji Tensorflow, DL4J a Keras. Na základě vzorové implementace pro CPU vytvořte akcelerovanou variantu pro grafické procesory. Demonstrujte funkčnost na predikci časové řady finančního indexu a srovnejte výkonnost CPU a GPU varianty. Srovnejte dosažené výsledky s dalšími algoritmy pro analýzu časových řad.

DOPORUČENÁ LITERATURA:

[1] Tang, Jiexiong, Chenwei Deng, and Guang-Bin Huang. "Extreme learning machine for multilayer perceptron." IEEE transactions on neural networks and learning systems 27.4 (2016): 809-821. http://ieeexplore.ieee.org/document/7103337/

[2] N-Dimensional Arrays for Java [Online], http://nd4j.org/

Termín zadání: 5.2.2018 ***Termín odevzdání:*** 21.5.2018

Vedoucí práce: doc. Ing. Radim Burget, Ph.D.
Konzultant:

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

Abstrakt

Práce je zaměřena na možnost využití extrémních učících se strojů a sítí s ozvěnou stavu pro předpověď časových řad s možností akcelerace pomocí grafických procesorů. Takovéto předpovědi jsou v dnešní době každodenní součástí života naprosté většiny lidí, a to vzhledem k jejich využití v předpovědích počasí, vývoje finančního a akciového trhu, spotřeby energie a mnohých dalších věcí. Práce uvádí teoretický podklad extrémních učících se strojů a sítí s ozvěnou stavu, jejichž hlavní výhodou je náhodná volba většiny parametrů neuronové sítě a iterativního postupu dopočtu parametrů, programovací nástroje k jejich realizaci, jako je knihovna ND4J a CUDA toolkit, tvorbu vlastního programu, a nakonec i test doby zpracování a přesnosti.

Klíčová slova

Neuronová síť, Extrémní učící se stroj, ELM, Síť s ozvěnou stavu, ESN, předpověď časových řad, akcelerace GPU procesory

Abstract

Thesis is aimed at the possibility of utilization of extreme learning machines and echo state networks for time series forecasting with possibility of utilizing GPU acceleration. Such predictions are part of nearly everyone's daily lives through utilization in weather forecasting, prediction of regular and stock market, power consumption predictions and many more. Thesis is meant to familiarize reader firstly with theoretical basis of extreme learning machines and echo state networks, taking advantage of randomly generating majority of neural networks parameters and avoiding iterative processes. Secondly thesis demonstrates use of programing tools, such as ND4J and CUDA toolkit, to create very own programs. Finally, prediction capability and convenience of GPU acceleration is tested.

Keywords

Neural network, Extreme learning machine, ELM, Echo state network, ESN, Time series forecasting, GPU acceleration

Bibliografická citace:

ZMEŠKAL, J. Extrémní učící se stroje pro předpovídání časových řad. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2018. 49 s. Vedoucí diplomové práce doc. Ing. Radim Burget, Ph.D..

Prohlášení

„Prohlašuji, že svou závěrečnou práci na téma Extrémní učící se stroje pro předpovídání časových řad jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.
Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné 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 části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne 21. května 2018 ……

podpis autora

Poděkování

Děkuji vedoucímu diplomové práce doc. Ing. Radimu Burgetovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne 21. května 2018 ……

podpis autora

SIX research centre
sensor, information and communication systems

Faculty of Electrical Engineering and Communication

Brno University of Technology Purkynova 118, CZ-61200 Brno Czech Republic

http://www.six.feec.vutbr.cz

PODĚKOVÁNÍ

Výzkum popsaný v této diplomové práci byl realizovaný v laboratořích podpořených projektem Centrum senzorických, informačních a komunikačních systémů (SIX); registrační číslo CZ.1.05/2.1.00/03.0072, operačního programu Výzkum a vývoj pro inovace.

Brno ....

Podpis autora

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

2007–13
OP Výzkum a vývoj pro inovace

Tato práce vznikla jako součást klíčové aktivity KA6 - Individuální výuka a zapojení studentů bakalářských a magisterských studijních programů do výzkumu v rámci projektu OP VVV Vytvoření double-degree doktorského studijního programu Elektronika a informační technologie a vytvoření doktorského studijního programu Informační bezpečnost, reg. č. CZ.02.2.69/0.0/0.0/16_018/0002575.

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Projekt je spolufinancován Evropskou unií.

OBSAH

Seznam obrázků

Seznam tabulek

1 ÚVOD

Předpovídání časových řad je velmi důležitou součástí každodenního života většiny lidí a zasahuje do velkého množství různých odvětví. Jeho nejčastější aplikací je například předpověď počasí, která je využívána téměř všemi, nebo například předpověď růstu či poklesu ceny v různých finančních odvětvích. Časové řady samotné jsou v dané problematice chápány jako posloupnosti hodnot, které jsou postupně v čase získávány pomocí měření veličin nebo pozorování a zaznamenávání vlastností.

Obvykle je ovšem k vytvoření efektivního modelu a dostatečně přesné předpovědi zapotřebí zpracování velkého množství dat a provedení náročných matematických operací. Tyto důvody vedly k tomu, že s postupným vývojem výpočetní techniky byl tento úkon více a více přenecháván počítačům, a to hlavně takzvaným umělým neuronovým sítím (Artifical neural network – ANN) [1]. Hlavní výhodou neuronových sítí v oboru předpovědi časových řad je přímé přizpůsobení výpočtu danému typu dat pomocí strojového učení, zvyšující přesnost odhadu.

Neuronové sítě ovšem nepřinesly problematice samozřejmě jen klady. K předpovědi hodnot byly obvykle využívány dopředné neuronové sítě s jednou nebo více skrytými vrstvami [2], učenými pomocí metod s učitelem. Příkladem takového postupu je metoda zpětného šíření (Back Propagation – BP) [1][3] chybové funkce, minimalizované iterativně pomocí gradientního sestupu (Gradient descent). Postup vyžaduje, aby veškeré parametry neuronové sítě, jako jsou vstupní váhy, výstupní váhy a předpojaté hodnoty, byly iterativně upravovány, což znamená zvýšení složitosti výpočtu. Hlavním problémem tedy je potřeba vysokého výpočetního výkonu a dlouhé trvání učebního procesu. Metoda má navíc riziko uvíznutí v lokálním minimu. Postupem času byly metody upraveny, čímž byly eliminovány některé nedostatky jako problém lokálního minima. Dlouhá doba učení je ovšem stále velkým problémem.

S řešením problému rychlosti učení přišli až extrémní učící se stroje (Extreme Learning Machine – ELM) [4][5][6][7]. Jedná se o metodu učení, kde parametry neuronové sítě až na výstupní váhy, tedy vstupní váhy a předpojaté hodnoty neuronů, jsou voleny náhodně na začátku učebního procesu. Tyto hodnoty pak dále nejsou měněny. Jediné neznámé hodnoty, které je tedy třeba zjistit učebním procesem, jsou tedy výstupní váhy skryté vrstvy, spojující tuto vrstvu s výstupními neurony. Výstupní váhy je tedy možné dopočítat a není tudíž potřeba časově a výpočetně náročného iterativního procesu.

Ani extrémní učící se stroje ovšem nepřinesli jen klady a úplné vynechání všech iterativních postupů. Je zde totiž komplikace s určením vhodné velikosti skryté vrstvy neuronů [8][16][6]. Nevhodná volba velikosti je srovnatelná se snahou napodobit složitý průběh křivky přímkou, nebo naopak napodobit přímku složitou křivkou. Tuto problematiku snaží řešit dále vyvíjené typy extrémních učících se strojů, které ovšem k určení správného počtu neuronů obvykle využívají časově náročných iterativních postupů. Dalším nedostatkem je poměrně nízká přesnost, dosažená programem vytvořeným v rámci práce.

Jedním z možných důvodů nedostatečné přesnosti ELM může být neschopnost dopředné jednovrstvé ANN pochytit dynamiku vývoje časové řady. Řešením tohoto problému může být další krok ve vývoji ELM, kterým je takzvaná síť s ozvěnou stavu (Echo State Network – ESN) [9][5], nahrazující skrytou vrstvu rekurentní neuronovou sítí.

Velkým zlomem ve využívání neuronových sítí byl nedávný prudký rozvoj a navýšení výkonu grafických karet. Složité výpočty prováděné obvykle nad obrovským množstvím dat byly donedávna příliš náročné na to, aby byly uskutečněny běžnými procesory v rozumném čase. Pro tyto účely bylo obvykle zapotřebí paralelní zpracování na mnoha procesorech. Situace se ovšem v posledních letech mění a k výpočtům je stále častěji a častěji využíváno grafických procesorů [10], které umožňují obrovský průtok zpracovávaných dat a jejich rychlé zpracování. Jednou z dominantních technologií v tomto oboru je CUDA.

Hlavním přínosem práce je detailní popis procesu návrhu a tvorby programů, a to určených jak pro zpracování procesorem, tak i pro graficky akcelerované zpracování. Práce je následně zaměřena na srovnání těchto verzí, a to hlavně z pohledu časové náročnosti. Výsledkem je schopnost rozlišit situace, kdy je grafická akcelerace vhodným nástrojem k urychlení průběhu programu a kdy je rychlejší ponechat zpracování na procesoru.

Tato práce je zaměřena na tvorbu dvou programů, modelujících neuronové sítě využívající algoritmu extrémních učících se strojů a sítě s ozvěnou stavu za účelem předpovědi časových řad, a to jak pro využití klasických, tak i grafických procesorů a jejich následné srovnání. V druhé části práce budou uvedeny informace o obou neuronových sítích a s nimi spojenou teorií. Třetí část je věnována popisu prostředků k tvorbě a samotné tvorbě programu. Nakonec ve čtvrté části jsou vyhodnoceny výsledky a je uvedeno srovnání verzí využívající procesor s verzemi využívající grafické karty.

2 VÝVOJ A MOŽNOSTI EXTRÉMNÍCH UČÍCÍCH SE STROJŮ

Vývoj extrémních učících se strojů byl započat ze dvou různých směrů, a to bez jejich prvotní vzájemné spolupráce. Dané směry byly oddělené a vlastně o sobě vůbec nevěděli. Prvním z těchto směrů bylo odvětví neurologie, snažící se o pochopení fungování biologického nervového systému a schopnost jeho napodobení pomocí uměle vytvořených modelů. Druhým směrem byl vývoj klasických umělých neuronových sítí, který se snažil o eliminaci mnohých nedostatků klasických metod učení neuronových sítí. Oba směry nakonec došly k podobným závěrům a možnosti částečného propojení obou odvětví.

2.1 Klasické extrémní učící se stroje

Prvotní vývoj klasických extrémních učících se strojů [4][11] mířil hlavně na eliminaci nedostatků starších metod učení dopředných neuronových sítí. Tyto sítě mají možnost využití v celé řadě odvětví od třídění dat, přes rozpoznávání tváří v obrázku, až po předpověď časových řad. Běžné metody učení byly obvykle založeny na zpětném síření chybové funkce (Back-Propagation – BP) [4], která byla iterativně minimalizována gradientním sestupem (Gradient descent). Tyto metody měly mnohé nedostatky, mezi které například patřilo sklouznutí do lokálního minima chybové funkce, vysoká složitost procesu a s tím související velká časová náročnost.

Prvotní extrémní učící se stroje využívali topologie plně propojených dopředných neuronových sítí [7][12][7][13][14], obsahujících většinou jedinou vrstvu skrytých neuronů. Některé vnitřní parametry, jako například vstupní váhy a v případě vícevrstvé topologie i váhy mezi neurony skrytých vrstev, byly iniciálně voleny náhodně, což nebylo až zas tak neobvyklé. V případě ELM ovšem takto zvolené váhy byly oproti jiným systémům, kde byly iterativně měněny podle chybové funkce, zafixovány a v dalším běhu sítě již nebyly měněny [9]. Tato změna otevřela cestu považování výstupu takovéto neuronové sítě za lineární systém, což otevřelo cestu využití lineární regrese k jednoduchému dopočtení výstupních vah. Výpočet výstupních hodnot lze pak provést pomocí vzorce:

$$y_i = \sum_{j=1}^{M} w_{out_j} f(w_{in_j} x_i + b_j), i \in [1, N], \qquad (1)$$

kde y_i je neuronovou sítí odhadnutým pokračováním i-té vstupní řady, f je aktivační funkce, W_{in} je vektor vstupních vah a b_j je předpojatá hodnota daného neuronu skryté vrstvy.

Velkou výhodou klasického ELM jsou minimální výpočetní nároky, tím pádem velice krátký čas strávený učením neuronové sítě a minimální potřebná intervence uživatele pro získání velice dobrého výsledku. Jako aktivační funkce je obvykle využíváno funkcí sigmoid nebo tanh. Jediným nastavitelným parametrem, který je individuální podle vstupních dat a může znamenat rozdíl mezi správným a nesprávným výsledkem je velikost skryté vrstvy. Pokud je počet neuronů skryté vrstvy příliš nízký tak neuronová síť nebude schopna pojmout prudší změny což může znamenat nesprávný výsledek. Naopak při volbě příliš vysokého počtu neuronů se aproximace stane zbytečně složitou, což má za následek zvýšení náročnosti výpočtu a také možnost nesprávného výsledku způsobeného nepotřebnou výchylkou aproximace.

Situaci si snaží řešit řada modifikací, která do procesu vkládá algoritmus buď pro postupné zvyšování nebo snižování velikosti skryté vrstvy [8][15]. Další varianty řeší problematiku například pomocí kernel funkcí [16].

2.2 Síť s ozvěnou stavu

Extrémní učící se stroje, využívající dopředné topologie mají obrovskou výhodu ve velice krátkém čase, potřebném k dokončení učebního procesu, který v případě jednovrstvé skryté vrstvy trvá řádově vteřiny. Jejich výhody ovšem končí se zvyšující se dynamikou vstupní časové řady, kde hodnoty mohou být spíše více závislé na předcházejících hodnotách než na nějaké stálé vývojové křivce.

Síť s ozvěnou stavu neboli Echo state network (ESN)[17], využívá možností a výhod zpětnovazebních neuronových sítí, nazývaných rekurentní neuronové sítě. Jak již název napovídá, tyto neuronové sítě obsahují ve své topologii cykly, umožňující v případě přítomnosti vstupních dat provedení jejich nelineární transformace v závislosti na předcházejících hodnotách, což může být chápáno jako forma dynamické paměti [5]. Každou z N hodnot v l-té vrstvě je možné tedy vyjádřit pomocí následujícího vzorce:

$$x_{l,n} = f(\sum_{j=1}^{N}(w_{l,j}\, x_{l-1,j} + w_{r,l+1,j} x_{l+1,j} + b_j)), n \in [1, N], \qquad (2)$$

kde $w_{l,j}$ je hodnotou váhy, spojující j-tou hodnotu předchozí vrstvy s j n-tou hodnotou aktuální vrstvy, $x_{l-1,j}$ je j-tou hodnotou předchozí vrstvy (l-1), $w_{r,l+1,j}$ je hodnotou rekurentní váhy, spojující současnou vrstvu s další vrstvou a $x_{l+1,j}$ je prvek j-tým prvkem následující vrstvy.

Učení rekurentních neuronových sítí bylo dlouhou dobu problematické, a to hlavně z důvodu velké složitosti při použití metod spojených s gradientem chybové

funkce [18]. Takovéto metody pro rekurentní sítě samozřejmě existují ovšem mají velké nedostatky jako například obrovskou složitost výpočtu velikého množství parametrů mnoha cykly, riziko degenerace gradientu a vysoké množství manuálně nastavovaných parametrů. S příchodem ESN se ovšem objevil nový přístup oddělující skrytou vrstvu od zbytku neuronové sítě, nazývaný Reservoir computing [17]. V případě ESN je pak část, obsahující vícevrstvou rekurentní neuronovou síť nazývána rezervoárem, její parametry, jako jsou váhy a předpoklady neuronů, jsou při tvorbě sítě náhodně zvoleny a dále již nejsou měněny. Tyto neurony tvoří nelineární transformaci vstupu a jeho historie. Výstup je pak tvořen pomocí lineární regrese.

Hledání pouze lineárního vztahu mezi vstupem a výstupem by ve většině případů nebylo dostačující, což by zapříčinilo velkou chybu odhadu. Je zde tedy obvykle využito nelineární rozprostření vstupu do velkého vektoru, zprostředkované vícevrstvou rekurentní neuronovou sítí skrytou v rezervoáru.

V rámci neuronových sítí, využívajících rezervoárových metod je síť s ozvěnou stavu jednou z metod sestavování rekurentního rezervoáru a řešení vstupů a výstupů. V rámci této metody je důležitá snaha o vytvoření dostatečně bohatého prostoru pro rozvoj vstupní posloupnosti, což znamená volbu dostatečně velkého počtu neuronů skryté vrstvy, a to podle charakteru vstupních dat v rámci od desítek až do tisíců neuronů. Dalším důležitým faktorem je propojení těchto neuronů, které by mělo být dostatečně náhodné, a hlavně ne moc časté. Hustota propojení vrstev rezervoáru má velký vliv na přítomnost výsledného stavu ozvěny [17]. Literatura obvykle doporučuje maximálně dvacetiprocentní propojení ze všech možných spojů.

Přítomnost zmíněného stavu ozvěny je jednou z nejdůležitějších součástí sestavování vhodného rezervoáru pro ESN. Jedná se o stav, ve kterém na výslednou hodnotu má vliv jak předchozí stav neuronu, tak i přecházející vstupní hodnota. Tento vliv by se současně měl pomalu vytrácet a neměl by postupně nabývat na svém účinku. Správné nastavení sítě vedoucí k těmto účinkům již ovšem není tak jednoduché, nelze to zaručit přesně předepsaným postupem či dodržením podmínek a z literatury vyplívá, že nastavení bude rozdílné pro různá data. Existuje ovšem několik podmínek, které sice nezaručují přítomnost ozvěny stavu, ale s jejich dodržením existuje vyšší pravděpodobnost přítomnosti zmíněného stavu. Jednou z takovýchto podmínek je udržet spektrální poloměr matic vah skryté vrstvy, tedy rezervoáru, menší než jedna. Hodnotu spektrálního poloměru lze jednoduše získat z hodnot vlastních čísel matice, a to vyhledáním nejvyšší hodnoty z absolutních hodnot vlastních čísel. Výsledný spektrální poloměr by být dán v závislosti na potřebné nelinearitě a době, po kterou starší data a stavy neuronů ovlivňují aktuální stav. Vzhledem k téměř lineárnímu průběhu funkce

tanh v okolí nuly je pro více nelinearity potřeba volit spektrální poloměr spíše bližší k jedničce, což současně způsobuje delší paměť, hodnoty blíže k nule jsou spíše vhodné pro případy, kde dlouhodobá paměť může uškodit. Je třeba ovšem brát v potaz fakt, že velikost spektrálního poloměru není dostatečnou podmínkou pro to, aby nastal stav ozvěny. Literatura ovšem hovoří i podmínce zaručující přítomnost stavu ozvěny, která je ovšem poněkud omezující. Ta říká, že pokud je využito nelinearity funkce tanh a velikost nejvyšší singulární hodnoty je menší než jedna, a to pro libovolný vstup včetně nuly. V praxi je tato podmínka ovšem málokdy dodržena. Zbylé parametry, jako jsou vstupní váhy a možnost vah zpětné vazby z výstupu, jsou obvykle voleny jako plné, náhodně generované matice.

2.3 Určení výstupních vah

Jak již bylo zmíněno po náhodném zvolení vstupních vah a parametrů neuronů je možné považovat učební proces za problematiku lineární algebry, pokud zvolené parametry již během procesu zůstanou nezměněny. Dopočet výstupních vah [19] je pak převeden na rovnici

$$W_{out} = H^{\dagger}T, \qquad (3)$$

kde $T_l = [t_1, \dots, t_N]^T$ je matice očekávaných výsledků pro N vstupních řad, tedy již dopředu známých hodnot určených k učení. $W_{out} = [w_{out1}, \dots, w_{outM}]$ je vypočítávaná matice výstupních vah pro M neuronů skryté vrstvy, spojených s vrstvou výstupní. Nakonec $H^{\dagger}$ je takzvaná zobecněná inverzní matice k matici výstupních hodnot skryté vrstvy H, která bývá také nazývána jako pseudoinverzní matice.

Tato matice je zapotřebí proto, že ve velkém množství případů nemá výstupní matice skryté vrstvy neuronů čtvercový tvar a z takové matice není možné běžným postupem získat matici inverzní, která je zapotřebí pro samotný výpočet výstupních vah. Nejčastěji je používána Moore-Penroseova pseudoinverze [3][14], jejíž výpočet je odvozen ze čtyř podmínek.

$$AA^{\dagger}A = A, \qquad (4)$$

$$A^{\dagger}AA^{\dagger} = A^{\dagger}, \qquad (5)$$

$$(AA^{\dagger})^{\mathrm{T}} = AA^{\dagger}, \qquad (6)$$

$$(A^{\dagger}A)^{\mathrm{T}} = A^{\dagger}A, \qquad (7)$$

kde A je původní matice, ze které je tvořena zobecněná inverzní matice, $A^{\dagger}$ je zobecněná inverzní matice z původní matice a index T značí transpozici matice.

Jednou z možností výpočtu zobecněné inverzní matice je výpočet za pomoci singulárního rozkladu hodnot (zkráceně SVD – Singular Value Decomposition) [20][21]. Za pomoci singulárního rozkladu lze získat matice U a V, a vektor vlastních čísel matice, ze kterého je následně vytvořena matice D diagonálním

vložením vektoru vlastních čísel. Z těchto tří matic je možné následně získat zobecněnou inverzní matici $A^{\dagger}$ [3][21], a to podle vzorce

$$A^{\dagger} = VD^{-1}U^{T}, \qquad (8)$$

Kde D^{-1} je inverzní matice k diagonální matici vlastních čísel. Nyní lze ověřit, že vypočtená matice splňuje podmínky 4–7, a že se tudíž opravdu jedná o zobecněnou inverzní matici.

2.4 Zvýšení přesnosti odhadu – sestava několika ESN

V rámci využití ESN pro předpovídání časových řad může vzhledem k náhodnosti mnohých parametrů neuronové sítě dojít k omezené nestabilitě, způsobující sice minimálně, ovšem přece jen rozdílné výsledky při vícenásobném zpracování stejných dat. U aplikací, které vyžadují maximální možnou přesnost může být tento fakt velice nežádoucí. Řešením této situace bylo vytvoření sestavy několika ESN, paralelně zpracovávajících stejná data, která jsou následně kombinována za účelem tvorby přesnějšího odhadu [22].

V rámci množiny několika ESN je důležité, aby všechny měly vhodně vybrané parametry sítě, na druhou stranu je ovšem důležitá rozmanitost těchto parametrů. Pokud budou parametry nevhodné, nemá smysl výstupy skládat dohromady protože výsledky obsahují vysokou chybu. Pokud budou parametry téměř shodné, nebude mezi jednotlivými výsledky žádný rozdíl.

Obvyklým přístupem v rámci sestav je paralelně vypočítané výsledky spojit jejich zprůměrováním, což je sice jednoduchý přístup, který ale není pro všechny případy vhodný. V tomto případě byla ke spojení výsledků sestavy využita dopředná neuronová síť, učená pomocí algoritmu extrémních učících se strojů.

Numerické testy ve výsledku potvrdili schopnost sestavy několika ESN, spojované dohromady pomocí ELM eliminovat nestabilitu způsobeno náhodně volenými parametry ESN, a dokonce i zvýšit přesnost celého systému [22].

2.5 Knihovny pro programování neuronových sítí

V dnešní době prudkého rozvoje neuronových sítí existuje široká škála nástrojů, zaměřených na zjednodušení, urychlení a optimalizaci jejich programování. Obvykle se jedná o knihovny s otevřeným kódem, implementující operace specifické pro neuronové sítě, mezi které patří například práce a matematické operace s velkými maticemi, prvkové transformace pomocí definovaných funkcí a podobně.

Mezi tyto knihovny patří například Nd4j, což je nástroj pro práci s velkými a více dimenzionálními poli hodnot pro programovací jazyk Java. Mezi jeho přednosti patří možnost využití mnohých matematických operací mezi tensory,

aplikovatelných prvkově na jeden tensor, například ve formě transformací, nebo mezi tensory, například násobení matic. Další výhodou je možnost využití jak CPU, tak i GPU a výběru mezi nimi. Jedná se o nástroj, umožňující větší svobodu v rámci tvorby neuronových sítí, jeho využití ovšem vyžaduje znalost problematiky.

Dalším nástrojem, umožňujícím podobnou flexibilitu a svobodu v programování neuronových sítí je TensorFlow. Jedná se opět o knihovnu s otevřeným kódem, vytvořenou Google Brain týmem původně pro vnitřní využití společností Google. Jedná se soubor matematických operací určených hlavně pro využití ve spojení s programovacím jazykem Python. Umožňuje podporu CPU i GPU, navíc definuje novou možnost Tensorového procesoru (Tensor processing unit – TPU), speciálně navrženého pro dosažení vysoké propustnosti u aritmetických operací s nízkou přesností směřovanou na použití modelů neuronových sítí.

Další možností, využívající tentokrát možnosti rychle a jednodušeji tvořit neuronové sítě bez zásahů do vnitřních proměnných a funkcí je Keras. Jedná se o nadstavbu TensorFlow zaměřenou na modulární a uživatelsky přátelský přístup. Knihovna je samozřejmě s otevřeným kódem a její hlavní předností je možnost rychlé tvorby sítě definovanými metodami pro přidávání vrstev a modifikaci parametrů. Přes široké možnost modifikace a přizpůsobení většiny struktury nenabízí ovšem tak velkou flexibilitu jako TensorFlow.

3 NÁVRH A TVORBA PROGRAMU AKCELEROVANÉHO POMOCÍ CUDA

Tato kapitola je věnována popisu prostředků, využitých jak k tvorbě dopředné neuronové sítě, využívající k procesu učení algoritmus ELM, tak i pro tvorbu ESN, obsahující rekurentní skrytou vrstvu. Vybraným programovacím jazykem je JAVA, a to hlavně z důvodu přenositelnosti mezi různými operačními systémy a podpory vybraných knihoven. Program je tvořen ve dvou verzích, kde jedna provádí výpočty za pomocí klasického procesoru (CPU) a druhá ke stejným výpočtům využívá grafický procesor (GPU), specificky platformu CUDA.

První část kapitoly je věnována prostředkům použitým k tvorbě obou programů a obsahuje stručný popis každého z nich. Další dvě části jsou již věnovány detailnímu popisu tvorby programu. Druhá část se tedy specificky věnuje tvorbě dopředné jednovrstvé neuronové sítě, využívající k učení algoritmu extrémních učících se strojů. Nakonec ve třetí části je podobně popsána tvorba druhého programu, kterým je neuronová síť s rekurentním vícevrstvým rezervoárem nazývané síť s ozvěnou stavu. Tato síť k učení využívá podobných principů jako ELM, které jsou modifikovány pro využití v rekurentních neuronových sítích.

3.1 Využité prostředky

Jednou ze základních využívaných knihoven je OpenBLAS, což je soubor základních metod lineární algebry. Jde o knihovnu s otevřeným kódem, která je rozšířením knihovny GotoBLAS, která již dále nebyla vyvíjena. Důležitou funkcí je zde zprostředkování rozhraní operacím lineární algebry pro operace mezi vektory, maticemi a vzájemně mezi vektory a maticemi. OpenBLAS dále také řeší funkci více vláknového zpracování matematických operací, která je v dnešní době velmi důležitou součástí vzhledem k využití více jádrových procesorů.

Další součástí je knihovna JavaCPP, poskytující rozhraní pro C++ v Javě. Tato knihovna efektivně poskytuje možnosti jazyka C++, a to i ty, které jsou považované za problematické. Je zde využíváno sémantické podobnosti mezi oběma jazyky. Využití C++ knihoven je pro projekt velice užitečné hlavně k využití CUDA, jejíž knihovny jsou pro jazyk C a C++. Samozřejmě že není podporován celý programovací jazyk, ale jsou zde Javě dodány důležité možnosti a funkce.

Velice důležitou součástí projektu je CUDA Toolkit, zprostředkovávající rozhraní a knihovny využívané pro akceleraci pomocí grafických procesorů. Jedná se o produkt firmy NVIDIA, určený pro jejich grafické karty, které podporují CUDA

technologii. Hlavním záměrem je poskytnout prostředky k dalšímu vývoji využití grafických procesorů například ke zpracování obrázků a videa, provádění výpočtů lineární algebry a mnoha dalším účelům. Je zde poskytnuta sada nástrojů, mezi které patří vývojové nástroje, vzorové kódy, nástroje sledování výkonu, a hlavně skupina knihoven, akcelerovaných grafickým procesorem. První z takových knihoven je CUDA Math Library, poskytující velice přesné a ověřené klasické matematické funkce. Je zde například podpora datových typů float a double, rozšířených trigonometrických a exponenciálních funkcí, chybové funkce, operace s plovoucí desetinou čárkou a další. Další důležitou součástí balíku akcelerovaných knihoven je cuRAND, zprostředkovávající tvorbu náhodných hodnot, a to s využitím vysokého počtu jader a mnohem rychleji. Je zde dále možnost využití různých kernel funkcí a výběr rozdělení pravděpodobnosti náhodného generátoru. V neposlední ředě je důležitou součástí akcelerovaných knihoven cuBLAS, umožňující provádění algebraických operací za pomocí grafického procesoru. Vzhledem k zaměření lineární algebry na vektory, vektorové prostory, matice a podobně, je zde velká pravděpodobnost na práci s velkým množstvím dat, se kterými je potřeba provést mnoho operací. Je zde tedy efektivně implementováno jak zpracování na mnoha procesorech jedné grafické karty, tak i možnosti distribuce výpočtů mezi několik grafických karet. Takovéto možnosti jsou velice užitečné například pro zpracování obrázků, které jsou obvykle reprezentovány buď jako vektory, nebo jako velká matice hodnot. Další využití těchto vlastností je v neuronových sítích, kde často samotný průchod dat sítí bývá realizován pomocí řady maticových operací, jako je násobení. Nakonec je zde knihovna s názvem cuSOLVER, obsahující skupinu metod, které efektivně řeší různé rozklady a algoritmy. Je zde například implementován i singulární rozklad hodnot, důležitý pro výpočet zobecněné inverzní matice.

Nakonec je v projektu využito knihovny ND4J, spojující všechny předchozí zmíněné části a poskytující programovacímu jazyku JAVA datový typ NDArray a mnohé matematické operace s ním. Jedná se o možnost využití vícedimenzionálního pole prvků, jako je vektor, matice či tensor. Tyto pole a operace s nimi jsou velice důležitou součástí programu, jejich struktura a operace mezi nimi reprezentují topologii neuronové sítě a její vnitřní průběhy.

Rozhraní programovacího jazyka je zde obálkou pro verze BLASu, pomocí kterého jsou následně realizovány různé operace nad instancemi datového typu NDArray. V rámci tohoto projektu je využíváno dvou různých verzí BLASu, které se liší využitím odlišného zakončení neboli backendu. První je JBLAS, využívající ke zpracování příkazů klasický procesor a zakončení nazývané nd4j-native. Druhou verzí pak je JCUBLAS, zpracovávající příkazy pomocí grafického procesoru ve spojení s technologií CUDA a využívající zakončení nazývané nd4j-cuda-8.0. Toto

zakončení bylo zdrojem prvních komplikací při tvorbě programu. Je totiž naprogramované specificky pro spolupráci s CUDA Toolkit 8. Aktuální verzí CUDA Toolkit ovšem je verze devět. Ve snaze o co nejefektivnější provedení byl tento fakt zanedbán, což způsobilo celou řadu problémů. BLAS totiž nebyl schopný pracovat s novějšími knihovnami a vzniklý výpis chyb informoval o chybějících knihovnách, které ovšem existovali. Problém byl nakonec vyřešen instalací verze osm toolkitu z archivu. Pomocí výběru vhodného zakončení lze tedy ovlivnit, zda budou matematické operace provedeny pouze za pomoci procesoru, nebo zda budou akcelerovány grafickou kartou.

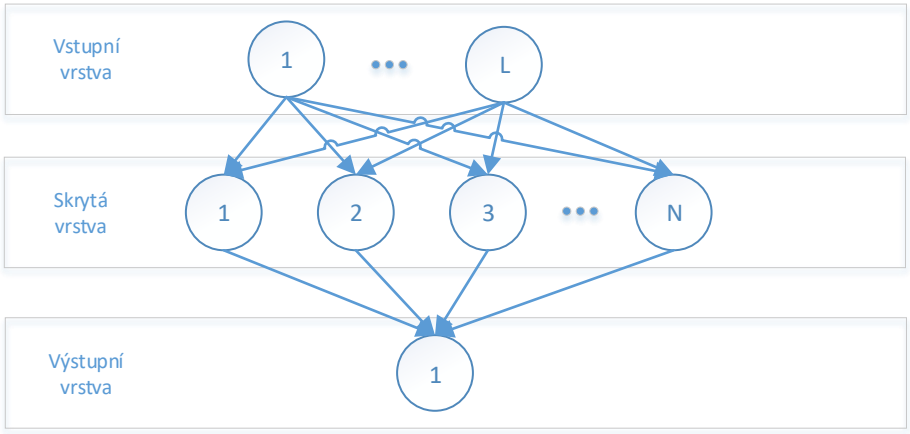
Knihovna ND4J poskytuje rozhraní programovacího jazyka širokou škálu základních a rozšířených operací. Základní operace lze rozdělit podle toho, jestli probíhají na úrovni prvků nebo polí. Na úrovni prvků lze provádět operace třemi způsoby, a to se skaláry, vektory nebo maticemi. Příkladem operace se skalárem může být přičtení či vynásobení matice skalárem, tedy jedním číslem. Toto číslo se pak přičte ke všem prvkům matice, popřípadě se s ním každý prvek matice samostatně vynásobí. Prvkové operace s vektory závisí na tvaru vektoru, tedy zda jde o vektor řádkový či sloupcový. Pokud je vektor řádkový tak například u prvkového součtu vektoru s maticí dojde k individuálnímu přičtení daného vektoru ke každému řádku matice. Pokud je pak vektor sloupcový tak analogicky dojde k přičtení ke každému sloupci matice. Poslední možností operací s prvky je mezi maticemi. Tato možnost umožňuje například vynásobit mezi sebou dvě matice, aniž by proběhlo klasické násobení matic. Jednoduše proběhne vynásobení jednotlivých odpovídajících prvků matic, tedy první prvek prvního řádku jedné matice pouze s prvním prvkem prvního řádku matice druhé a tak dále pro ostatní prvky. Vedle operací na úrovni prvků lze také využít operací na úrovni polí, jejichž hlavním zástupcem je klasické násobení dvou matic, u kterého je jediná podmínka, a to že počet řádků matice, kterou násobíme, musí být roven počtu sloupců násobené matice. Další základní možností využití knihovny pak spočívají v manipulaci s maticemi. Do této kategorie spadá možnost transpozice matice, možnost změny rozměrů matice, pokud zůstane zachována její celková délka, a vertikální a horizontální skládání matic. Další možnosti, již patřící mezi rozšířené, jsou například transformace. Ty umožňují jednoduše transformovat celou matici hodnot pomocí mnoha předem definovaných funkcí, příkladem může být funkce sigmoid, tanh, sin a mnohé další. Mezi rozšířené operace patří také různé ztrátové funkce, Fourierova transformace a konvoluce.

Pro program jsou také velice důležité výhodné možnosti tvorby polí typu NDArray a systém, pomocí kterého jsou ukládány do paměti. Tato pole totiž nevyužívají pro ukládání dat klasické haldy (HEAP) jazyku JAVA. Místo toho jsou v paměti uloženy jako jednolitý blok po sobě jdoucích hodnot. Tento systém jim

přináší mnohé výhody jako podstatné zlepšení výkonu a lepší schopnost spojení a spolupráce s rychlými a optimalizovanými knihovnami používaného BLASu. Výhody těchto polí se tedy projeví hlavně při manipulaci s velkým množstvím dat. Dalším faktorem, zlepšujícím výkonnost knihovny při využití velkých polí je, že dvě a více polí může odkazovat na stejná data v paměti, čehož je hlavně využíváno v případech, kdy je třeba provést například transpozici velké matice bez nutnosti kopírovat data. Z hlediska tvorby polí je zde také implementováno velké množství výhodných možností. První z nich, která je velmi důležitá vzhledem k zaměření projektu na ELM, je možnost náhodné generace matice, a to i s možností ovlivnění generátoru náhodných hodnot. Lze zde změnit jak pravděpodobnostní rozdělení náhodných hodnot, tak i takzvané semeno, podle kterého jsou hodnoty generovány. Další možnosti tvorby polí jsou tvorba polí nul, vhodné pro inicializaci polí, tvorba pole jedniček, tvorba pole z klasického pole jazyka JAVA anebo tvorba z ostatních NDArray polí. Je samozřejmě možnost i vytvořit pole diagonálním vložením vektoru, vhodné například pro tvorbu jednotkové matice.

3.2 Popis tvorby programu pro ELM akcelerovaného pomocí CUDA

V projektu je využito topologie jednovrstvé dopředné neuronové sítě s plně propojenými vrstvami, viz Obrázek 1, a to z důvodu výborných vlastností a schopností v oblasti předpovědi časových řad. Topologie tedy obsahuje tři vrstvy, jejichž velikost se liší na základě několika parametrů. První vrstva, reprezentující vstup neuronové sítě, má velikost odpovídající délce vstupních posloupností, načtených na počátku běhu programu. Druhá vrstva, plnicí funkci skryté vrstvy neuronové sítě má nastavitelnou velikost. Tuto velikost lze ovšem v současné verzi měnit pouze ve zdrojovém kódu programu. Poslední vrstvou je výstup neuronové sítě, obsahující předpovězené hodnoty. Její velikost je v současné verzi jedna



Obrázek 1 - Příklad topologie ELM

vzhledem k předpovídání jedné v čase následující hodnoty. V případě potřeby předpovědi více hodnot je možné počet výstupních neuronů zvýšit, jako v předchozím případě by ovšem byl potřeba v současné verzi zásah do zdrojového kódu.

Jako první bylo potřeba vyřešit, jakým způsobem budou do programu vkládána data. Vstupními daty je zde několik řad číselných hodnot stejné délky. Vzhledem k potřebě vložení stovek nebo tisíců hodnot do programu nepřipadá manuální vkládání jednotlivých hodnot v úvahu, je zde tedy využito načítání hodnot z textového souboru specifického formátu. Je zde potřeba, aby všechny hodnoty každé posloupnosti byly zapsány v řádku a odděleny středníkem. Není potřeba uvádět počet posloupností nebo hodnot v posloupnosti. Při zavolání metody loadData() program nejprve projde načítané soubory a zjistí, kolik je v souboru uloženo posloupností a kolik hodnot každá posloupnost obsahuje. Po projití celého souboru je cyklus ukončen a zahájen další cyklus, který již načítá uložená data a ukládá je připravených polí hodnot, jejichž velikost je již známa z předchozího cyklu. Data jsou matice hodnot, určené pro učení a testování. Již v této metodě jsou od maticí hodnot odděleny poslední hodnoty každé posloupnosti, reprezentující cílové hodnoty pro proces učení a ověřovací hodnoty, využívané pro výpočet chyby testovacího procesu. Tyto matice, počátečně uložené jako klasické pole hodnot programovacího jazyku JAVA, jsou následně převedeny na datový typ INDArray, využívaný knihovnou ND4J a ostatními metodami programu.

Jakmile je známa velikost vstupních matic, velikost skryté vrstvy a počet výstupů je možno začít s tvorbou struktury neuronové sítě. V dalším kroku jsou tedy vytvořeny a inicializovány matice vstupních vah a předpokladů jednotlivých neuronů. Tyto hodnoty jsou náhodně generovány pomocí metody Nd4j.rand(řádky, sloupce), poskytované knihovnou ND4J. U matic je potřeba zvolit vhodnou velikost a vhodně zvolit jejich tvar. Matice vstupních vah je počet řádků roven počtu vzorků vstupních posloupností a počet řádků je dán počtem neuronů skryté vrstvy. Tvar je dán umožněním maticového násobení bez potřeby provádění dalších operací, například transpozice. U matice předpokladů neuronů skryté vrstvy je potřeba, aby hodnoty předpokladů byly stejné pro všechny vstupní řady. Toho je docíleno náhodným vygenerováním hodnot prvního řádku, což odpovídá hodnotám pro první posloupnost, a následným vkládáním kopií prvního řádku do dalších řádků, a to, dokud není dosaženo shody počtu řádků a počtu vstupních posloupností.

Nyní následuje další krok, a to samotný proces učení neuronové sítě. Ten je realizován metodou train() a spočívá v provedení tří hlavních kroků, a to výpočtem výstupních hodnot jednotlivých neuronů skryté vrstvy, vypočtením zobecněné

inverzní matice k matici výstupních hodnot skryté vrstvy a nakonec v posledním kroku výpočet výstupních vah skryté vrstvy. V prvním kroku je nejprve potřeba vypočítat vstupní hodnoty jednotlivých neuronů. Tyto hodnoty jsou vzhledem k plně propojené topologii dány součtem všech vstupních hodnot v posloupnosti, vynásobených vstupními vahami. Postup lze popsat vzorcem

$$s_{in_1} = \sum_{i=1}^{L} x_i w_{in_i},$$

kde, s_{in} je vektor vstupních hodnot skryté vrstvy, $X = [x_1, \dots, x_L]$ je vektor vstupních hodnot jedné vstupní řady neuronové sítě a W_{in} je vektor vstupních vah neuronové sítě. Ze vzorce lze jasně vidět, že výpočet všech vstupních hodnot skryté vrstvy pro všechny vstupní řady lze jednoduše programové realizovat jedním krokem pomocí maticového násobení mezi maticí vstupních hodnot a vstupních vah. Vzhledem k vhodné volbě tvaru matice vstupních vah lze navíc provést násobení okamžitě bez dalších úprav matic. Druhým krokem ve výpočtu výstupu skryté vrstvy je přidání předpokladu neuronů. Předpoklad neuronů je z pohledu neuronové sítě považován za další vstup, individuální pro každý neuron skryté vrstvy, se vstupní vahou rovnou jedné. Postačuje tedy přičtení k již vypočítané vstupní hodnotě. Operaci lze provést pomocí prvkového součtu dvou matic, a to matice předpokladů a vypočtené matice vstupů neuronů. Následně je v dalším kroku provedena transformace pomocí aktivační funkce. Aktivační funkce by podle teorie měla být spojitá a nekonečně diferencovatelná. V programu je využito funkce sigmoid, která je vhodným příkladem aktivační funkce a je již navíc implementována mezi funkcemi v knihovně ND4J. Proces lze provést pomocí metody Transforms.sigmoid(INDArray), která provede transformaci prvek po prvku a jako návratovou hodnotu dává pole s vypočtenými transformacemi. Výstupem transformace je již matice výstupních hodnot skryté vrstvy neuronů.

Dalším krokem v procesu učení je výpočet zobecněné inverzní k matici právě vypočítaných výstupních hodnot skryté vrstvy. Zobecněná inverzní matice je zapotřebí k výpočtu výstupních vah a její výpočet je již komplikovanější. Jednou z možností jejího výpočtu je za použití SVD, neboli singulárního rozkladu hodnot, a to pomocí metody gesvd(). V této části již začínají další komplikace a rozdíly mezi CPU a CUDA verzí programu. Singulární rozklad v CPU verzi vyžaduje jako vstup pole hodnot typu INDArray a jako výstup vrací tři pole stejného typu, kde jedno je vektor a dvě další jsou matice. Při použití CUDA verze rozkladu je vstupem také pole, je zde ovšem omezení. Vstupní pole musí být buď čtvercové anebo musí mít vyšší počet řádků než sloupců. Navíc se oproti CPU verzi liší i výstup, a to tak, že místo vektoru a dvou matic jsou zde výstupem tři vektory, z nichž jeden je navíc uložen s jiným řazením. Při nedodržení vhodného formátu vstupu nebo volbě

špatných rozměrů při inicializaci výstupních proměnných je navíc pouze vygenerována výjimka, informující o selhání metody, která počítá hodnoty rozkladu. Po dlouhém bádání byla nalezena informace o požadavcích na velikost vstupního pole prvků a po mnoha pokusech s volbou mnohonásobně větších výstupních polí byl odhalen formát výstupu. Naštěstí po drobných změnách v následujícím výpočtu zobecněné matice neovlivňuje transpozice vstupu výsledek. Komplikace s nepodporovanými rozměry vstupní matice tedy byla nakonec vyřešena transpozicí vstupu a dva výstupní vektory, které byly u CPU verze matice musely být přeformátovány na matice. Knihovna Nd4j naštěstí pro tyto účely obsahuje nástroje. Tímto nástrojem je metoda reshape(), která umožňuje změnu rozměrů pole, pokud jeho délka zůstane zachována. Jako parametry má nejen počet řádků a sloupců nového pole, ale také možnost volby jedno ze dvou systémů řazení. Tyto systémy jsou C řazení, převzaté z programovacího jazyka C, který ukládá jednotlivé prvky pole v paměti po řádcích za sebou, a F řazení, pocházející z jazyka Fortran, který naopak ukládá prvky v paměti po sloupcích. Volbou řazení z jazyka Fortran byl vyřešen problém v odlišném řazení jednoho z výstupních vektorů. Po převedení výstupu singulárního rozkladu zbývá dopočet zobecněné inverze, a to podle vzorce 6. Matice V a U jsou již připraveny a zbývá pouze matice D, která je vytvořena diagonálním vložením výstupního vektoru ze SVD do matice nul. Z důvodu potřeby pro výpočet je pak takto vytvořená diagonální matice invertována pomocí metody InvertMatrix.invert() a druhá matice je transponována. V následujícím kroku je již prováděno maticového násobení za účelem získání zobecněné inverzní matice.

Po výpočtu zobecněné inverzní matice již jsou k dispozici všechny potřebné hodnoty k dopočtení výstupních vah podle vzorce (1). Je tedy provedena operace maticového násobení, jejímž výsledkem je cílová hodnota učebního procesu. Z důvodu ověření úspěšnosti učebního procesu je v následujícím kódu vypočítán odhad neuronové sítě na testovací data. Z tohoto odhadu je spočítána chybová funkce, která by měla být minimální, pokud ne nulová. Na začátku celého procesu učení byl za účelem změření doby potřebné pro učební proces do proměnné uložen systémový čas v milisekundách. Po ukončení výpočtu chybové funkce je systémový čas odečten znovu a je proveden rozdíl dvou odečtených hodnot, který je následně převeden na sekundy. Výpisem očekávaného výstupu, předpovězených hodnot, chyby a doby zpracování je ukončena metoda učení neuronové sítě.

Následuje proces testování, mající za úkol ověření správného průběhu učebního procesu a správné funkčnosti neuronové sítě, tentokrát ovšem ne na datech použitých k učení ale na rozdílné množině dat. Postup je velice podobný jako v metodě učení až na několik rozdílů. Nejprve jsou opět vypočítány vstupní hodnoty jednotlivých neuronů skryté vrstvy, následuje přičtení předpokladů

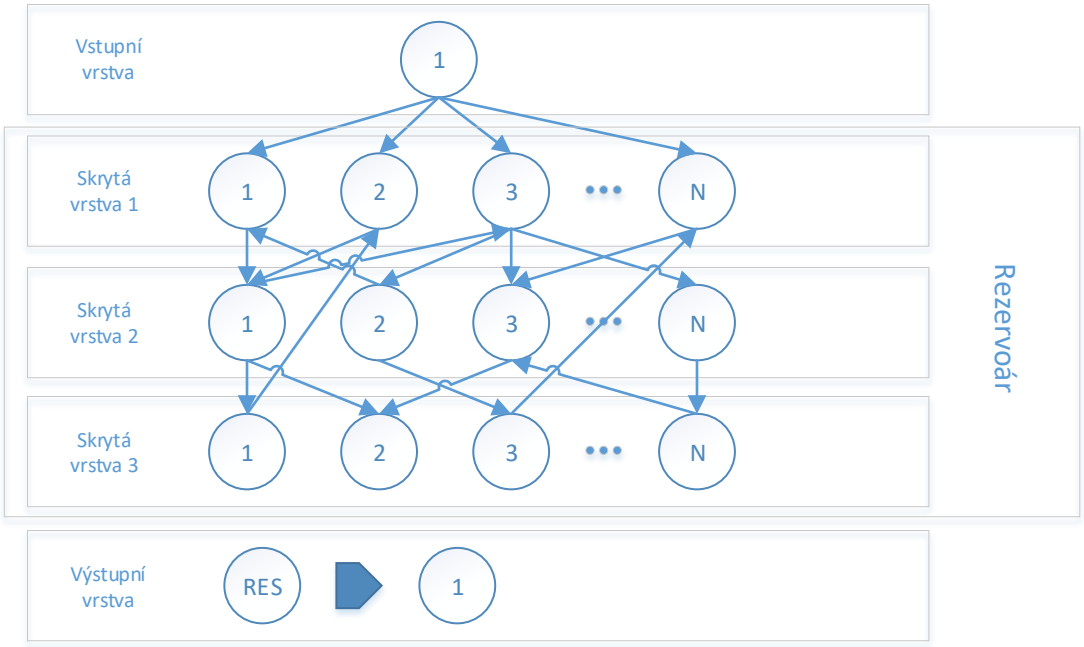
neuronů a výpočet transformace aktivační funkcí. V následujícím kroku již neprobíhá opětovný výpočet výstupních vah, ale jsou použity výstupní váhy vypočítané procesem učení. Následuje tedy získání předpovězených hodnot a výpočet chybové funkce.

3.3 Popis tvorby programu pro ESN akcelerovaného pomocí CUDA

V případě druhého programu již topologie není tak jednoduchá jako v první případě. Je opět složena ze tří druhů neuronů, a to vstupní neuron, výstupní neuron a neurony skryté vrstvy. Zásadní rozdíl nastává u skryté vrstvy, která je u ESN považována za rezervoár, obsahující vícevrstvou rekurentní neuronovou síť, a to navíc s řídce propojenými vrstvami. Vzorová topologie ESN je zobrazena v Obrázek 2.

Vzhledem k potřebě vzít v potaz časovou posloupnost dat bylo potřeba zvolit jiný přístup než v minulém případě. Vstupních neuronů již není podle počtu hodnot vstupní posloupnosti, ale vzhledem k uvažování jedné vstupní posloupnosti je pouze jeden. Do tohoto vstupního neuronu je v každém cyklu načtena jedna hodnota ze vstupní posloupnosti, přičemž počet cyklů je roven počtu hodnot ve vstupní posloupnosti.

Počet výstupních neuronů byl v případě tohoto programu zvolen také jako jeden, hodnotu je možné ovšem změnit, což by mělo za následek možnost předpovídání více než jedné hodnoty. Změna by samozřejmě musela proběhnou před procesem učení a v rámci učebního procesu by bylo zapotřebí vypočítat výstupní váhy pro další neurony. Správná funkce skryté vrstvy závisí na správném nastavení několika parametrů, které se pro každá data liší a dle literatury je nelze jednoznačně určit. Z tohoto důvodu bylo zapotřebí ponechat počet neuronů na vrstvu, počet vrstev a úroveň propojení jednotlivých vrstev jako nastavitelné, aniž by bylo potřeba zásahu do kódu programu přidáním dalších proměnných. V případě samotných dat neuronů skryté vrstvy to nebyl problém. Data neuronů každé vrstvy lze považovat za dlouhý vektor. Tyto vektory pak lze jednoduše poskládat do celkové matice dat, jejíž počet řádků je roven počtu vrstev a počet sloupců se rovná počtu neuronů na vrstvu. Matice předpokladů jednotlivých neuronů má naprosto stejný tvar, jen je při tvorbě topologie náhodně vygenerována a dále se nemění. Vzhledem k jedinému vstupnímu neuronu je matice vstupních vah také jednoduchá. Jejím úkolem je rozprostřít vstupní hodnotu do všech neuronů vstupní vrstvy. Jedná se tedy o matici náhodných vah, která plně propojuje vstupní vrstvu s první vrstvou rezervoáru. Vzhledem k volitelnému počtu skrytých vrstev již k vyjádření



Obrázek 2 - Vzorová topologie ESN

a uchování vah, propojujících dopředně tyto vrstvy, nejsou dvě dimenze matice postačující. Je zde tedy využito tensoru, obsahujícího tři dimenze, přičemž dimenze navíc slouží k oddělení matic pro jednotlivé vrstvy. Rozměry tohoto tensoru jsou tedy počet vrstev, počet neuronů zdrojové vrstvy dat a počet neuronů cílové vrstvy dat. Podobně bylo zapotřebí vytvořit i tensor rekurentních vah, obsahující váhy pro propojení každé vrstvy se všemi vyššími vrstvami. Tento tensor vyjadřuje zpětnovazební spoje a oproti předchozímu se liší pouze v počtu matic, vložených do tensoru. Vzhledem k nastavitelnému počtu vrstev a k faktu, že každá vyšší vrstva přijímá rekurentní spoje z o jednu vyššího počtu vrstev je potřeba před vytvořením tensoru zjistit skutečný počet rekurentních matic.

První komplikace nastali při snaze vytvořit tří dimenzionální tensor náhodných hodnot. Přesto že knihovna Nd4j obsahuje poměrně široké možnosti tvorby matic a vektorů, obsahujících náhodné hodnoty generované podle různých rozdělení a náhodných generátorů, nelze zde generovat tensor náhodných hodnot. Při zapsání třetího parametru (Nd4j.rand(int, int, int)) není třetí hodnota považována za rozměr třetí dimenze, ale za hodnotu semene pro generátor náhodných hodnot. Toto bylo překvapující vzhledem k funkčnosti metod pro generování tensoru nul či jedniček, využívající velice podobné syntaxe. Bylo tedy třeba změnit přístup a pro vygenerování tensoru využít jiných nástrojů. Z tohoto důvodu byla vytvořena metoda CreateRandomTensor(int, int, int), jejíž parametry nejsou nic jiného než rozměry každé ze tří dimenzí, a jejíž účel je tvorba tří dimenzionálního tensoru. V metodě je nejprve je vytvořen nulový tensor o daných rozměrech, přičemž první hodnota určuje počet vložených matic a další dvě

hodnoty definují velikost matic. Následuje cyklus, ve kterém je vytvořena matice náhodných hodnot, která je následně vložena na určené místo pomocí metody tensor.putSlice(int, INDArray). Iterační proměnná cyklu je velice užitečná pro definování polohy matice v tensoru. Po doběhnutí cyklu je vrácen takto náhodně vytvořený tensor.

Pro účely ESN z literatury vyplynula potřeba vytvořit na rozdíl od ELM, které využívá plně propojené vrstvy, a tudíž husté matice, neúplné propojení vrstev, realizované pomocí řídkých matic vah. Po bližším prozkoumání možností knihovny Nd4j vyšlo ovšem najevo, že tvorbu řídkých matic nepodporuje a že je to jedno z budoucích rozšíření knihovny. Bylo tedy potřeba vymyslet vlastní postup tvorby těchto matic. Vhodným postupem by mohlo být odděleně vytvořit dvě matice, a to plnou matici náhodných vah a matici jedniček a nul s přesným poměrem, daným například procentuálním poměrem jedné z hodnot, a tyto matice prvkově vynásobit. Knihovna nabízí možnosti úpravy dostupných náhodných generátorů včetně změny rozdělení pravděpodobnosti, semene, podle kterého se posloupnosti generují, dokonce je i možno změnit rozmezí ze kterého jsou hodnoty generovány, možnost generovat matici ze dvou hodnot s pravděpodobností výskytu hodnot ovšem nalezena nebyla. Matici jedniček a nul je ovšem možno generovat i za pomocí knihovnou poskytovaných prvkových transformací, jednou z nichž je zaokrouhlení. Je tedy vygenerována matice náhodných hodnot v rozmezí od nuly do jedné, jejíž hodnoty jsou následně zaokrouhleny na celá čísla a je vytvořena požadovaná matice. Tímto způsobem je dosaženo průměrné propojení vrstev, neboli zastoupení jedniček v matici, kolem padesáti pěti procent. Úroveň propojení vrstev je ovšem jednou z hodnot mající vliv na správnost výsledné předpovědi, kterou nelze jednoznačně určit a pro každá data může být jiná. Navíc dle literatury by se propojení vrstev mělo pohybovat spíše kolem dvaceti procent než kolem šedesáti. Je tedy vytvořen cyklus, který v každém běhu je vytvořena další matice náhodných hodnot od nuly do jedné, která je následně zaokrouhlena a prvkově násobena s prvotní maticí jedniček a nul. Tímto postupem je dosaženo postupné snižování úrovně propojení vrstev nastavením pouze jednoho parametru. Bohužel ovšem není možno zadat přesně výsledné propojení vrstev, jedná se o náhodně generované hodnoty, a tudíž lze pouze určit průměrnou úroveň propojení. Pokud je tedy parametr nastaven na nulu, tak je propojení vrstev plné, při nastaví hodnoty na číslo jedna je propojení vrstev průměrně kolem padesáti pěti procent, je-li využito cyklu jednou, parametr je tedy dvě, je propojení vrstev kolem dvaceti pěti procent, využitím cyklu třikrát je možné se dostat na průměrně patnáct procent. Samozřejmě že se zvyšujícím se počtem provedených cyklů narůstá i rozptyl výsledného propojení.

Dalším důležitým krokem bylo správné pochopení učebního a testovacího procesu ESN. První myšlenka byla zaměřena na vstup. Pokud by byl zvolen přístup jako v případě ELM a počet vstupních neuronů byl zvolen roven počtu prvků ve vstupní posloupnosti tak by všechna data prošla sítí najednou a rekurentní vazby by neměly žádný význam. Byla tedy zvolena cesta jednoho vstupního neuronu v cyklu o délce vstupní posloupnosti, přičemž na začátku každého cyklu je načtena další hodnota.

Následující řešenou částí byl průběh dat skrytou vrstvou sítě. Vzhledem k rekurentní povaze skryté vrstvy je velký důraz kladen na časovou posloupnost vstupujících dat. Nejlepším řešením se jeví pomocí cyklu o délce počtu vrstev procházet postupně vrstvami a dopočítávat výstupní hodnoty neuronů. Pokud ovšem bude proces zahájen od první vrstvy a bude pokračovat dále do nižších vrstev tak data projdou celou sítí během jednoho cyklu a v dalším cyklu budou mít rekurentní spoje ze všech vrstev zdrojové hodnoty vypočítané ze stejné vstupní hodnoty. Lepší možností je zde procházet síť od poslední vrstvy k první, což ve výsledku způsobí časový posuv mezi jednotlivými vrstvami rezervoáru za cenu přítomnosti několika operací, které by měly proběhnout s nulovou maticí dat. V každém cyklu jsou tedy načtena data dané vrstvy do proměnné, dále je podmínkou zkontrolováno, jestli datová matice neobsahuje pouze nuly, a pokud ne tak proběhne proces výpočtu nových výstupních hodnot dané vrstvy. Pokud je datová matice nulová, není třeba s ní provádět jakékoliv operace vzhledem k nulovému výsledku násobení, jedná se o omezený počet operací na začátku učebního procesu, kdy první vstupní data ještě neprošla celou sítí.

Při výpočtu nových hodnot je po načtení dat potřeba načíst z tensoru matici vah pro danou vrstvu. To je realizováno pomocí funkce tensorAlongDimension(), která je schopná z tensoru o určitém počtu dimenzí vytáhnout tensor o nižším počtu dimenzí. V tomto případě je potřeba z tensoru o třech dimenzích vytáhnout tensor o dvou dimenzích neboli matici. Následně proběhne maticové násobení dat vrstvy s prořídlou maticí vah, získanou z tensoru, a výsledná data jsou uložena dočasně do datové matice. Dále následuje přidání rekurentních vazeb z nižších vrstev. Prvotní řešení této úlohy spočívalo ve snaze o vložení a přičtení ukazatele na prvotně náhodně určený prvek z datové matice. Tato snaha ovšem nebyla úspěšná, a nakonec byla zvolena varianta přičtení hodnot vynásobených maticí vah. Vzhledem k volitelnému počtu vrstev skryté vrstvy a snaze o možnost zpětných vazeb ze všech nižších vrstev to ovšem znamená přičtení proměnlivého množství matic podle počtu nižších vrstev. Řešení spočívalo v cyklu, jehož délka je proměnlivá v závislosti na rozdílu indexu zpracovávané vrstvy a iterační proměnné vrstev, index matice v tensoru rekurentních vah je určován pomocnou hodnotou. Po přičtení všech rekurentních hodnot nebo v případě poslední vrstvy,

která nepřijímá rekurentní vazby, jsou k datům vrstvy přičteny náhodné předpoklady neuronů a data jsou převedena pomocí aktivační funkce. V případě prvotních dat, používaných během tvorby programu byla zvolena jako aktivační funkce tanh a proces byl proveden pomocí metody prvkových transformací, dostupné z knihovny Nd4j. Výsledná data zpracovávané vrstvy jsou následně uložena do datové matice skryté vrstvy a celý proces pokračuje až dokud nejsou zpracována všechna vstupní data. Další výjimka je u první vrstvy, která je vypočítávána mimo hlavní cyklus vrstev. Její vstupní data nejsou brána z matice dat skryté vrstvy ale ze vstupního neuronu a bez ohledu na počet uběhlých cyklů je potřeba ji počítat vždy. Proces výpočtu dat této vrstvy je stejný až na použití plné matice vah, což znamená plné propojení vstupní vrstvy s první skrytou vrstvou neuronové sítě.

Po ukončení všech cyklů, což znamená po načtení všech dat do skryté vrstvy neuronové sítě, je v učebním procesu na řadě získání výstupních vah. Podobně jako u ELM pro tento účel byla zvolena metoda lineární regrese neboli dopočtení výstupních vah pomocí Moore-Penrose pseudoinverzní matice, počítané ze všech neuronů skryté vrstvy. Bylo využito funkce hstack(dstArray, srcArray), umožňující vkládat data z jednotlivých vrstev za sebe do jedné proměnné. Samotný proces dopočtení pseudoinverzní matice je téměř shodný s výpočtem u předchozího programu. Opět je zde využito singulárního rozkladu hodnot a následného dopočtení matice z vypočtených hodnot. V procesu se ovšem oproti minule objevila chyba. Vzhledem ke tvaru vstupní matice, která je dlouhá jednořádková sekvence hodnot. Jedním z výsledků SVD je tedy pouze jedno vlastní číslo, které po diagonálním vložení vektoru do matice vytvoří pouze matici o jednom řádku a jednom sloupci. Následná snaha o inverzi této matice samozřejmě selže. Po dalším studiu problematiky vyšlo ovšem najevo, že co v některé literatuře bylo označeno jako výpočet inverze byla ve skutečnosti jiná forma pseudoinverze, znamenající pouze převrácení hodnot na hlavní diagonále. Vzhledem k faktu, že všude mimo hlavní diagonálu jsou nuly byla tedy matice z leva vydělena maticí jedniček, což způsobilo převrácení hodnot a vyřešení problému. Po dopočtení je Moore-Penrose pseudoinverzní matice vynásobena výstupními hodnotami, čímž jsou dopočteny výstupní váhy.

Následuje již pouze dopočtení výstupu pro ověření správnosti učebního procesu dopočtením odhadnuté hodnoty, následované vypočtením chyby procesu, počítané z poměru rozdílu odhadnuté a cílové hodnoty vůči rozsahu vstupních hodnot. Po celou dobu učení je samozřejmě počítána doba trvání procesu, založená na rozdílu systémového času na začátku a na konci procesu.

Nakonec je zde i metoda testování, která je shodná s metodou učení až na chybějící dopočet výstupních vah a pseudoinverzi. Jde o ohodnocení schopnosti

neuronové sítě odhadnout správnou hodnotu z dat, která nebyla přítomna v učebním procesu a výstupní váhy jim tedy nejsou přesně přizpůsobeny.

4 VYHODNOCENÍ VÝSTUPŮ PROGRAMŮ

V této kapitole je provedeno vyhodnocení výsledků vytvořených programů, a to hlavně z hlediska funkčnosti programu, přesnosti a doby zpracování. Hlavní výhodou ELM a ESN by měla být podstatně nižší doba učení než u ostatních metod učení neuronových sítí, využívajících iterativní postupy. Době učení je zde tedy dáván největší důraz.

4.1 Testování doby učení ELM

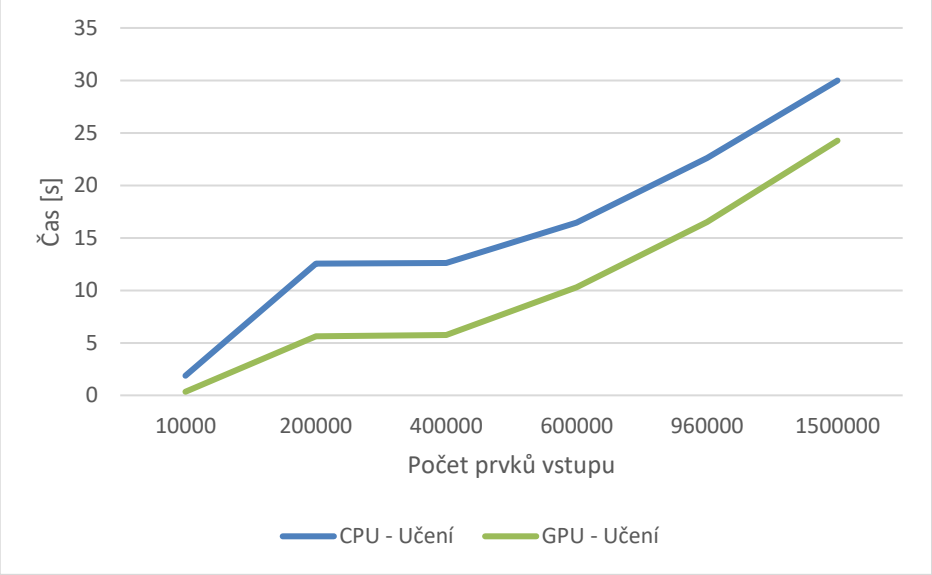
Doba učení a testování je v programu vyčítána pomocí rozdílu systémového času na začátku a na konci obou metod. Systémový čas je počítán v milisekundách a následně převáděn na sekundy. Důležitým faktorem z pohledu doby zpracování je také možnost akcelerace pomocí grafického procesoru, využívajícího technologii CUDA. Testy byly prováděny na programu vytvořeném v rámci této práce, přičemž počet neuronů skryté vrstvy byl zvolen na deset tisíc a velikost vstupních matic dat byla postupně zvyšována. První test proběhl na poskytnuté matici dat, která obsahovala dvě posloupnosti po pěti tisících prvcích. Další testy proběhli pouze za účelem zjištění doby zpracování na náhodně generovaných vstupních hodnotách. V následující tabulce jsou uvedeny naměřené hodnoty doby zpracování v závislosti na velikosti vstupní matice dat. Tyto hodnoty byly následně vloženy do grafů.

V tabulce lze vidět očekávanou rostoucí časovou náročnost výpočtu s rostoucí velikostí vstupní matice. Důležitý je zde i podstatný rozdíl mezi procesem učení a testování, který je daný jediným rozdílem, a to výpočtem výstupních vah. Vyplývá z toho tedy vysoká výpočetní náročnost procesu výpočtu zobecněné inverzní matice. Dále si lze všimnou podstatného rozdílu doby zpracování mezi prvním a druhým výpočtem, na který může mít vliv náhodné generování vstupní matice hodnot.

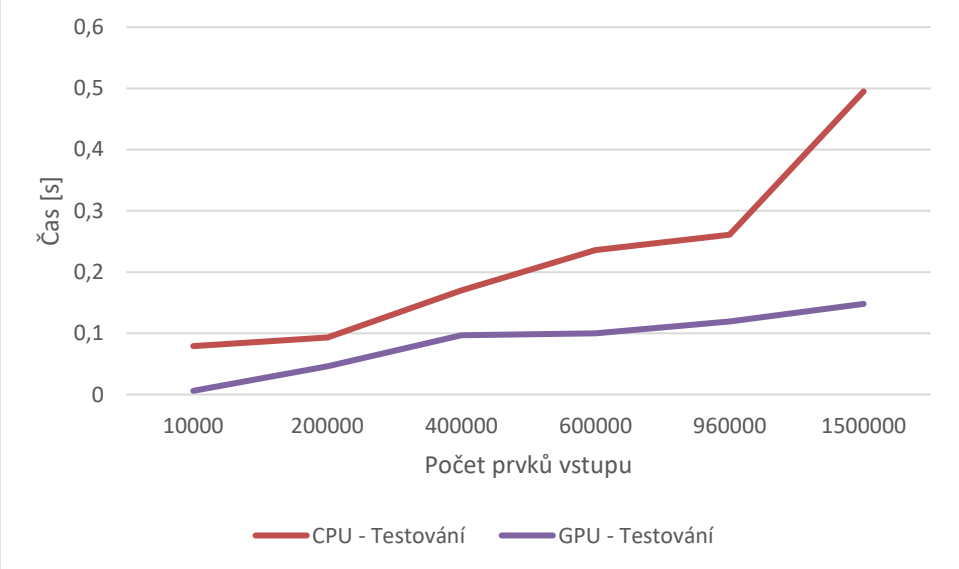
Důležitou věci je zde ovšem srovnání zpracování pomocí procesoru oproti verzi akcelerované pomocí CUDA. Na počátku je rozdíl u procesu učení obrovský, činí více než pětinásobek doby zpracování. Srovnání lze vidět v tabulce Tabulka 1. Se zvyšujícím se počtem prvků ovšem rozdíl klesá až se ustálí zhruba jeden a čtvrt násobku. Grafické srovnání časové náročnosti procesu učení lze vidět v grafech - Graf 1 a Graf 2.

Tabulka 1 - Doba zpracování pro různý počet vstupních prvků.

Velikost matice	CPU			GPU – CUDA		Počet prvků
	Učení [s]	Testování [s]		Učení [s]	Testování [s]	
2x5000	1,87	0,079		0,343	0,006	10000
400x500	12,546	0,093		5,626	0,046	200000
400x1000	12,607	0,17		5,734	0,097	400000
600x1000	16,439	0,236		10,289	0,1	600000
800x1200	22,611	0,261		16,504	0,119	960000
1000x1500	29,991	0,495		24,265	0,148	1500000



Graf 1 - Závislost doby učení na počtu vstupních vzorků.



Graf 2 - Závislost doby testování na počtu vstupních vzorků.

V grafu lze také vidět grafické srovnání průběhu procesu testování. I zde je ve vidět z počátku obrovský rozdíl, kde zpracování pomocí CUDA je více než třináctkrát rychlejší, následné se rozdíl ustálí zhruba na dvojnásobku. Nakonec ovšem při velkých rozměrech vstupní matice začne rozdíl opět rapidně narůstat, a to na vice než trojnásobek.

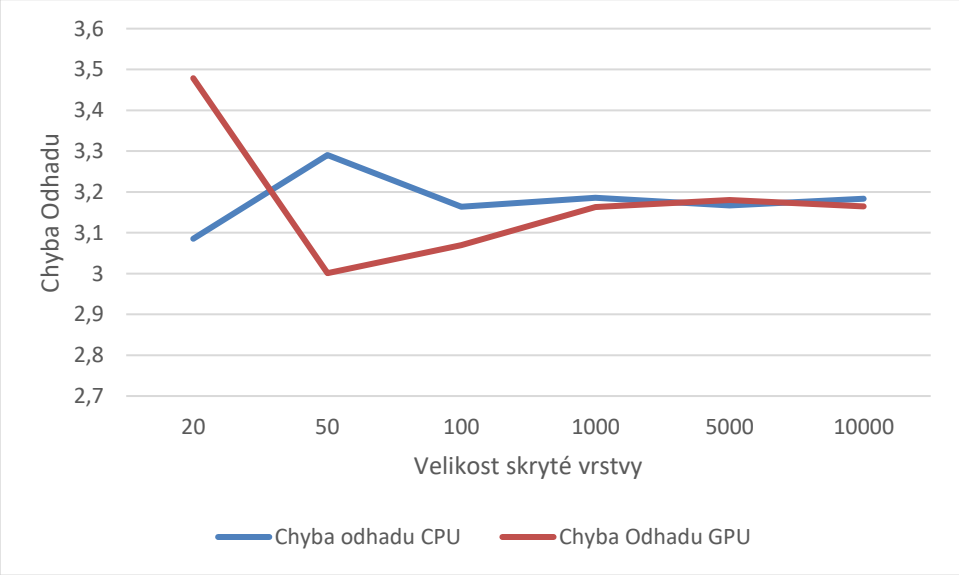
4.2 Testování přesnosti odhadu ELM

Komplikace nastaly při vyhodnocování přesnosti předpovědí neuronové sítě. Proces učení proběhne správně a chyba učení je minimální, a to řádově 10^{-7}. Chyba testování na druhou stranu je poměrně vysoká. Dle prostudované literatury jsou dvě možnosti vzniku chyby odhadované hodnoty. První možností je nedostatečný počet vstupních dat, a to ať už v ohledu počtu vzorků na posloupnost, nebo v počtu posloupností. Vzhledem k poměrně vysokému počtu vzorků v každé z posloupností je zde pravděpodobnější možností nedostatečný počet vstupních posloupností. To může mít za následek přílišné přizpůsobení neuronové sítě vstupním datům a následný vznik chyby při testování na odlišné posloupnosti. Druhou možností je potom v literatuře často uvažovaný problém správného odhadnutí velikosti skryté vrstvy. Příliš malý počet neuronů skryté vrstvy by mohl zapříčinit nedostatečné přizpůsobení vstupním datům, zatímco příliš vysoký počet neuronů skryté vrstvy by mohl naopak způsobit přílišné přizpůsobení vstupním datům.

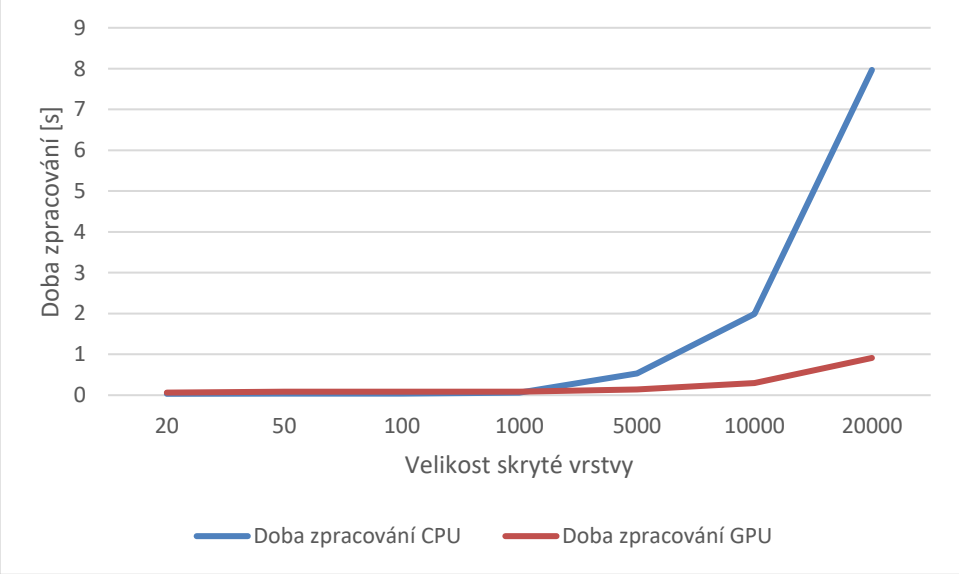
Přesnost byla testována na druhé možnosti, která je častěji diskutována, a to pomocí vstupní matice, obsahující dvě posloupnosti po pěti tisících prvcích. Pro test bylo dále zvoleno několik možností velikosti skryté vrstvy. Jeho výsledky jsou ukázány v tabulce. Vzhledem k zaměření práce na akceleraci procesu pomocí grafických procesorů a technologie CUDA je v tabulce Tabulka 2 uvedeno i srovnání rozdílu přesnosti mezi CPU a CUDA variantou a dále poměrně zajímavé srovnání doby zpracování pro rozdílné velikosti skryté vrstvy. Hodnoty přesnosti jsou aritmetickým průměrem deseti po sobě jdoucích pokusů. Pro lepší názornost jsou hodnoty vyneseny do grafů - Graf 3 a Graf 4.

Tabulka 2 - Srovnání chyby odhadu a doby zpracování pro různé velikosti skryté vrstvy.

Velikost skryté vrstvy	Chyba odhadu		Doba zpracování [s]	
	CPU	GPU	CPU	GPU
20	21,0386	12,3244	0,034	0,063
50	3,0853	3,4786	0,038	0,083
100	3,2904	3,001	0,042	0,083
1000	3,164	3,0698	0,064	0,088
5000	3,186	3,1627	0,53	0,14
10000	3,167	3,18	1,991	0,299
20000	3,1834	3,1644	7,967	0,912



Graf 3 - Graf závislosti chyby odhadu na počtu neuronů skryté vrstvy.



Graf 4 - Graf závislosti doby zpracování na velikosti skryté vrstvy.

Z tabulky Tabulka 2 lze okamžitě vidět, že první hodnota velikosti skryté vrstvy je nedostatečná a dochází zde k nedostatečnému přizpůsobení neuronové sítě. U ostatních velikostí skryté vrstvy jsou již hodnoty velice podobné. Během testování se ovšem do hodnoty pěti tisíc neuronů skryté vrstvy objevovali poměrně velké skoky v přesnosti, kde se chyba pohybovala od dvou až téměř do čtyř. Od pěti tisíc neuronů výše se již neuronová síť jevila podstatně stabilnější a hodnoty se lišili pouze v rámci zhruba dvou desetin. Rozdíl v přesnosti mezi zpracováním pomocí procesoru a CUDA je minimální, lze ovšem pozorovat drobné zvýšení přesnosti u verze programu, využívající CUDA.

Uvedené srovnání doby zpracování je ovšem velice zajímavé. U nízké velikosti skryté vrstvy je na rozdíl od předchozích testů rychlejší CPU varianta. Rozdíl se pomalu srovnává až se někde mezi tisícem a pěti tisíci neurony vyrovná. Následuje ovšem rapidní nárůst rozdílu ve prospěch CUDA verze, a to až na téměř devítinásobek u dvaceti tisíc skrytých neuronů. Vyplívá z toho tedy že s rostoucí velikostí matic je grafickým procesorem akcelerovaná verze výhodnější a výhodnější.

4.3 Testování přesnosti odhadu ESN

V ohledu ESN bylo třeba se nejdříve věnovat přesnosti odhadu. Program byl prvotně testován na shodných datech jako ELM. Učení sítě vždy vzhledem k použití lineární regrese proběhlo s minimální chybou, a to řádově 10^{-6} procenta rozsahu vstupních hodnot. Proces testování oproti tomu průměrně vyšel s chybou kolem dvou procent. Vše se tedy jevilo být v rámci přijatelných tolerancí.

Rozšíření zadání ovšem požadovalo test na reálných datech, a to specificky na finančním indexu. Po získání těchto dat a provedení prvotních testů již ovšem přesnost nebyla až tak dobrá, chyba se totiž pohybovala kolem sedmdesáti sedmi procent rozsahu vstupních hodnot. Tento výsledek je nejspíše následkem charakteru vstupních dat. Aby ovšem nedošlo k omylu tak bylo vyzkoušeno několik nalezených možností zlepšení výsledku a jejich testování.

Prvotní snaha najít řešení problému se ohlížela zpět na tvorbu rezervoáru a na otázku, jak jej vytvořit optimálně pro vstupní data. Byly tedy zkoušeny různé možnosti nastavení parametrů postupnými změnami jedno z nich a zachováním ostatních. Změny těchto parametrů ovšem na výstupní hodnotu měly minimální, pokud ne žádný, vliv. Následovala tedy úprava tvorby rezervoáru, a to podle výše zmíněných pravidel. Specificky byla snaha o dodržení podmínky zaručující dosažení stavu ozvěny, tedy že největší singulární hodnota váhových matic rezervoáru má být nižší než jedna. Za tímto účelem byl do tvorby řídkých matic vah přidán výpočet singulárního rozkladu matic, přesněji metoda gesvd(), poskytovaná knihovnou Nd4j. Z rozkladu byla vybrána pouze proměnná,

obsahující vektor singulárních hodnot. Z tohoto vektoru byla následně vybrána maximální hodnota. Po prvotní vyzkoušení se ukázalo, že singulární hodnota matic byla podle tohoto pravidle příliš velká a u větších matic dosahovala i hodnot přes dvanáct. Singulární hodnoty se zdály být nejvíce závislé na hodnotách jednotlivých prvků matice, proto snaha o jejich zmenšení spočívala v dělení jednotlivých prvků matic vah stejným číslem. Po dosažení maximální singulární hodnoty, která nebyla příliš malá, ale na druhou stranu nepřesahovala jedničku byl znovu proveden test programu se zaměřením na přesnost odhadu. Opět ovšem bylo dosaženo stejného výsledku s poměrně vysokou chybou.

Během dalších testů s jinými daty ovšem bylo zaznamenáno, že výstupní hodnota procesu testování je nápadně podobná výstupu, dosaženém při ověření chyby učebního procesu. Síť je tedy velice dobře přizpůsobena učebním datům, při přivedení neznámých dat na vstup ovšem není schopná zobecnění a výsledek odhadu je poměrně nepřesný. Po dohledání informací o podobných realizacích ESN se stejnými nedostatky vyšlo najevo, že se nejspíše jedná o problém přílišného přizpůsobení neuronové sítě učebním datům.

Problematika přeučení sítě je řešitelná hned několika postupy. Prvním nalezeným postupem bylo zmenšení počtu skrytých neuronů v rezervoáru, které již ovšem bylo bez úspěchu testováno dříve a nalezené diskuze o problematice hovoří o riziku volby příliš nízkého počtu neuronů, což by mělo negativní vliv na schopnost sítě najít řešení. Druhou možností řešení je rozdělení dat do tří částí, a to učební, validační a testovací. Potom by bylo možné průběžně během učení ověřovat přesnost pomocí validační části dat a v okamžiku, kdy se přesnost přestane zlepšovat ukončit učební proces předčasně. Tato metoda je ovšem použitelná pouze u online sekvenčních metod učení, tudíž vzhledem k použití obyčejné lineární regrese v programu nepoužitelné. Další možností bylo změny poměru dat, využívaných k procesu učení a testování, a to do podoby, kdy je většina z celkové množiny vstupních hodnot určena k procesu učení, přesněji poměr je mezi deseti až dvaceti procenty dat pro testování a osmdesáti až devadesáti procenty dat pro učení. Z původního poměru padesáti procent dat na učení byly tedy učebnímu procesu postupně přidávány data, dokud pro učební proces nebylo určeno devadesát procent dat. Žádoucí změny výsledku odhadu ovšem dosaženo nebylo. Poslední nalezenou možností pro dosažení lepších výsledků je použití jiné metody pro výpočet výstupních vah. Nalezené možnosti často uvažovali použití Tychonovovy regularizace (známé jako Ridge regression[23]) a podobných metod, které obsahují volitelné parametry, vyvažující složitost modelu oproti charakteru vstupních dat. Toto řešení by ovšem již byl příliš velký zásah do programu a nebylo dále ověřováno.

4.4 Testování doby zpracování ESN

Testování doby zpracování metod ESN probíhalo identicky jako u ELM, tedy odečtením systémového času na začátku a na konci průběhu metod učení a testování. Následně je proveden rozdíl odečtených časů a výsledek je převeden na sekundy. Hlavní důraz v této práci je kladen na rozdíly verze zpracovávající data pouze pomocí CPU a verze využívající grafickou akceleraci. Vzhledem k optimální kombinaci vysoké propustnosti vnitřní sběrnice a schopnosti provést enormní počet operací s plovoucí desetinou čárkou za sekundu by v rámci maticových výpočtů měla grafická akcelerace čas zpracování zkrátit. Testy doby zpracování probíhaly pro různé možnosti nastavení parametrů sítě, tedy pro různý počet neuronů na vrstvu, různý počet vrstev a pro různou úroveň propojení vrstev.

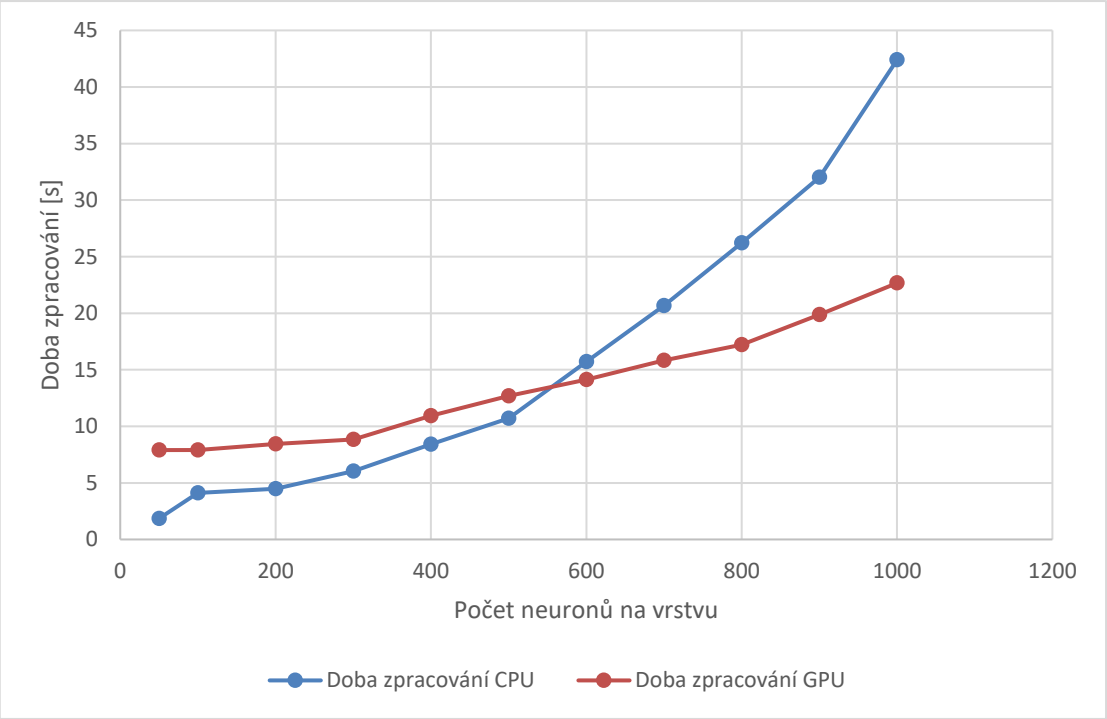
Jako první byl zvolen test závislosti doby zpracování na počtu neuronů skryté vrstvy. Jedná se poměrně důležitý parametr, který částečně definuje, jak složitou vstupní posloupnost bude neuronová síť schopna zvládnout, protože definuje do jak bohatého prostoru bude rozšířena vstupní posloupnost.

Testování proběhlo tedy pro různé velikosti skryté vrstvy s tím, že ostatní parametry nebyly měněny. Počet vrstev byl po celou dobu nastaven na pět a úroveň propojení vrstev jak pro dopředné, tak i rekurentní vazby byla kolem dvaceti pěti procent, parametr byl tedy pro oboje váhy nastaven na číslo dva. V průběhu testů nebyla brána v potaz přesnost odhadu. Počet neuronů na vrstvu byl prvotně nastaven na padesát, tedy dohromady dvě stě padesát neuronů celkem ve všech vrstvách, a byl postupně zvyšován až na tisíc na vrstvu. Každá z hodnot v následující tabulce Tabulka 1 je vypočítána jako průměrná hodnota z celkem deseti průběhů, hodnoty byly pro lepší názornost vyneseny i v grafu Graf 5.

Na počátku měření vše naznačovalo, že výsledek bude jiný, než bylo očekáváno, a to že varianta zpracovávaná pouze procesorem bude z nějakého důvodu rychlejší než graficky akcelerovaná verze, a to i přes poměrně vysokou složitost výpočtů. U malých rozměrů skryté vrstvy, například změřených padesáti neuronů na vrstvu je procesor dokonce více než čtyřikrát rychlejší. Při postupné navyšování počtu neuronů na vrstvu se ovšem GPU akcelerovaná verze pomalu začala blížit CPU verzi, až nakonec při zhruba pěti stech padesáti neuronech na vrstvu se doby zpracování vyrovnají. Z grafu lze vidět, že křivka doby zpracování pomocí CPU v závislosti na počtu neuronů na vrstvu má spíše exponenciální charakter. Oproti tomu stejná charakteristika pro GPU má spíše lineární povahu. Od určitého počtu prvků na vrstvu tedy začne být akcelerovaná verze opravdu výhodnější.

Tabulka 3 - Doba zpracování ESN v závislosti na velikosti skryté vrstvy

Neuronů ve vrstvě	Doba zpracování	
	CPU [s]	GPU [s]
50	1,8543	7,9014
100	4,1247	7,9119
200	4,4753	8,4460
300	6,0261	8,8436
400	8,4147	10,9383
500	10,7094	12,6987
600	15,7229	14,1273
700	20,6742	15,8189
800	26,2126	17,2019
900	32,0247	19,8854
1000	42,3901	22,6644



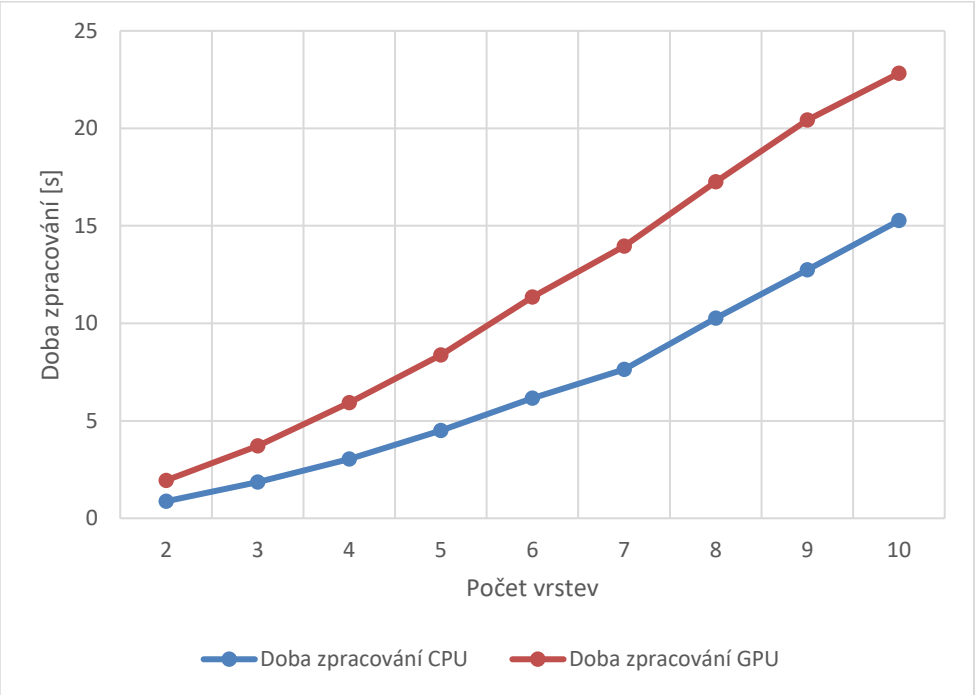
Graf 5 - Doba zpracování ESN v závislosti na velikosti skryté vrstvy

Dalším testem v pořadí byl rozdíl doby zpracování CPU a GPU verze v závislosti na počtu vrstev rezervoáru. Vzhledem ke způsobu, jakým je program napsán a k výsledku předchozího testu by změna počtu vrstev neměla i přes přidání složitosti mít zásadní vliv při srovnávání CPU s GPU. Přidáním dalších vrstev jsou sice přidávány další neurony, počet neuronů na vrstvu je ovšem stále stejný. Počet neuronů na vrstvu pro tento test byl zvolen sto, úroveň propojení vrstev zůstává stejná jako v předchozím případě, tedy kolem dvaceti pěti procent a je měněn pouze parametr počtu vrstev. Výsledky jsou jako obvykle průměrem z deseti pokusů.

Z naměřených hodnot lze jasně vidět (viz Tabulka 4, Graf 6), že navýšení počtu vrstev sice má na dobu zpracování vliv, ovšem doba zpracování téměř lineárně narůstá s počtem vrstev. Graficky akcelerovaná verze je zde tedy pomalejší pouze z důvodu volby malého počtu neuronů na vrstvu. Změna počtu vrstev tedy nemá vliv na výhodnost či nevýhodnost grafické akcelerace.

Tabulka 4 - Doba zpracování ESN v závislosti na počtu vrstev

Počet vrstev	2	3	4	5	6	7	8	9	10
Doba zpracování CPU [s]	0,8734	1,8583	3,0377	4,4992	6,1503	7,6343	10,2615	12,7445	15,2669
Doba zpracování GPU [s]	1,9439	3,7167	5,9246	8,3684	11,3447	13,9543	17,2564	20,4297	22,8249



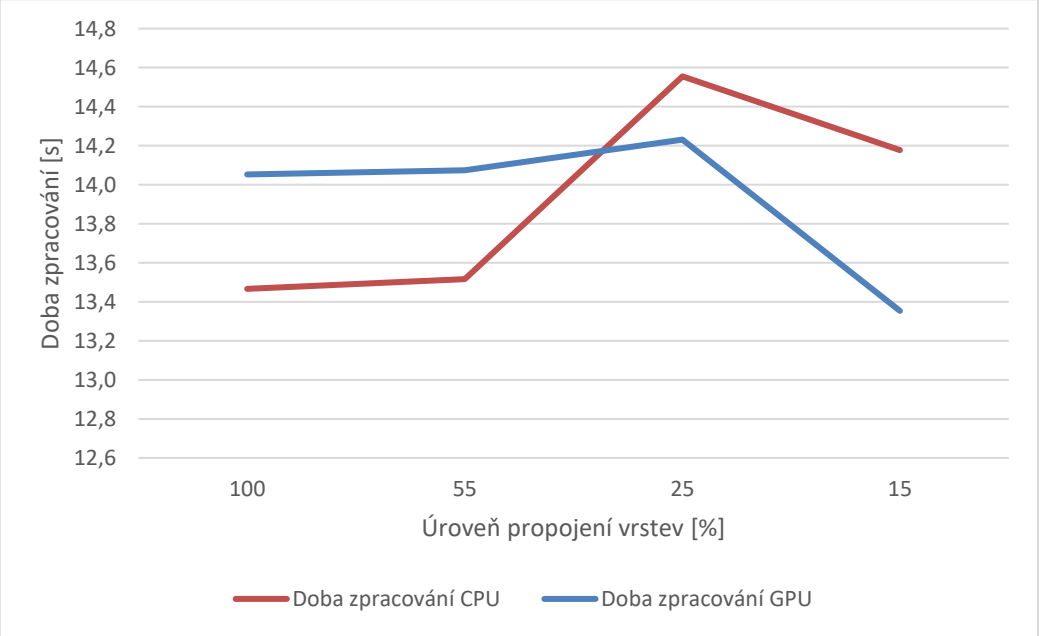
Graf 6 - Doba zpracování ESN v závislosti na počtu vrstev

Posledním z řady testů pro ESN je vliv úrovně propojení vrstev na dobu zpracování pro obě verze programu. Pro tento účel budou parametry sítě nastaveny na stálou hodnotu, až na úroveň propojení jednotlivých vrstev v rámci rezervoáru, což bude iniciálně nastaveno na plné propojení obou vrstev a následně bude postupně snižováno pomocí proměnných, zmíněných při popisu tvorby programu. Počet neuronů na vrstvu byl nastaven na pět set padesát a počet vrstev na pět. Naměřené hodnoty jsou opět průměrovány z deseti pokusů, přesné hodnoty lze najít v tabulce Tabulka 5 a pro názornost byly vyneseny do grafu Graf 7.

Na počátku měření se vzhledem k rostoucí tendenci hodnot zdálo, že procento nulových hodnot v maticích vah má na dobu zpracování malý vliv. Ke konci měření se ovšem projevila poměrně vysoká náhodnost těchto hodnot, kdy pro poslední měření v obou případech doba zpracování poměrně dost poklesla místo očekávaného nárůstu. Může to být výsledkem postupu využívaného programem pro generování nulových hodnot v maticích. Ten využívá kombinace zaokrouhlování náhodně generovaných hodnot od nuly do jedné a prvkového násobení matic. Čím vyšší je tedy počet operací násobení, tím větší náhodnost vzniká ve výsledném poměru nulových a nenulových hodnot v matici, což sice mohlo zapříčinit nepřesnosti, ale nemělo by to způsobit větší propojení vrstev po více násobeních. Výsledkem tedy je poměrně náhodná závislost a minimální nebo spíše nulový vliv propojení vrstev na rozdíl doby zpracování pomocí CPU nebo s grafickou akcelerací.

Tabulka 5 - Závislost doby zpracování na úrovni propojení vrstev

Propojení Vrstev	Doba zpracování	
	CPU	GPU
100	13,4665	14,0529
55	13,5171	14,0736
25	14,5554	14,2312
15	14,1782	13,3537



Graf 7 - Závislost doby zpracování na úrovni propojení vrstev

5 ZÁVĚR

Práce je věnována možnosti využití umělých neuronových sítí pro předpovídání časových řad. Přesněji jsou zde popsány sítě využívající náhodného generování parametrů skryté vrstvy a výpočtu výstupních vah pomocí lineární regrese. Je zde uveden detailní popis tvorby zaprvé jednovrstvé dopředné neuronové sítě, využívající algoritmu extrémních učících se strojů, a zadruhé neuronové sítě, využívající nahrazení skryté vrstvy rezervoárem obsahujícím rekurentní vícevrstvou neuronovou síť, což je řešení nazývané síť s ozvěnou stavu.

Práce je v rámci těchto dvou sítí zaměřena na možnosti využití grafické akcelerace pomoci technologie CUDA ve spojení s knihovnou Nd4j. Z provedených testů vyšlo jasně najevo, že některé parametry, jako je hustota propojení vrstev a počet vrstev rekurentního rezervoáru nemají na výhodnost využití grafické akcelerace téměř žádný vliv, a to vzhledem k minimálnímu ovlivňování složitosti jednotlivých výpočtů. Co na druhou stranu již velký vliv má je samotná velikost jednotlivých skrytých vrstev. Na základě tohoto parametru totiž bylo jasně z testů vidět, že pokud jsou jednotlivé vrstvy malé, je výhodnější ponechat výpočty na procesoru, od jisté velikosti ovšem začne doba zpracování procesorem prudce narůstat, zatímco při zpracování s grafickou akcelerací se závislost drží téměř lineárního charakteru.

Zjištěná závislost je připisována poměru doby stráveném přesunem dat mezi systémovou a grafickou pamětí a dobou samotného zpracování dat. Pokud je tedy síť malá, trvá déle přesunout data do grafické paměti než jejich zpracování a tím pádem je ve výsledku zpracování procesorem rychlejší vzhledem k vynechání přesunu dat. Grafická akcelerace má tedy na základě provedených testů velký potenciál pro využití v rozsáhlých sítích, které mají vysokou složitost jednotlivých výpočtů, například násobení rozsáhlých matic, a neprovádí neustále jen operace kopírování. To je názorně vidět u obou vytvořených programů při přidání počtu neuronů na vrstvu nad určitou hodnotu může být grafická akcelerace i mnohonásobně rychlejší.

Literatura

[1] ENE, Alexandru; STIRBU, Cosmin. A Java application for the failure rate prediction using feed forward neural networks. In: *Electronics, Computers and Artificial Intelligence (ECAI), 2016 8th International Conference on*. IEEE, 2016. p. 1-4.

[2] HUSSAIN, Tassadaq, et al. Experimental Study on Extreme Learning Machine Applications for Speech Enhancement. *IEEE Access*, 2017.

[3] HUANG, Guang-Bin; ZHU, Qin-Yu; SIEW, Chee-Kheong. Extreme learning machine: theory and applications. *Neurocomputing*, 2006, 70.1: 489-501.

[4] CAMBRIA, Erik, et al. Extreme learning machines [trends & controversies]. *IEEE Intelligent Systems*, 2013, 28.6: 30-59.

[5] GEORGE, Koshy, et al. Comparison of neural-network learning algorithms for time-series prediction. In: *Advances in Computing, Communications and Informatics (ICACCI), 2017 International Conference on*. IEEE, 2017. p. 7-13.

[6] KUMAR, Sachin, et al. ELM variants comparison on applications of time series data forecasting. In: *Advances in Computing, Communications and Informatics (ICACCI), 2016 International Conference on*. IEEE, 2016. p. 1404-1409.

[7] PLUMMER, E. Time series forecasting with feed-forward neural networks: guidelines and limitations. *Neural Networks*, 2000, 1: 1.

[8] HUANG, Guang-Bin; CHEN, Lei. Convex incremental extreme learning machine. *Neurocomputing*, 2007, 70.16: 3056-3062.

[9] CRONE, Sven F.; HIBON, Michele; NIKOLOPOULOS, Konstantinos. Advances in forecasting with neural networks? Empirical evidence from the NN3 competition on time series prediction. *International Journal of forecasting*, 2011, 27.3: 635-660.

[10] FATAHALIAN, Kayvon; SUGERMAN, Jeremy; HANRAHAN, Pat. Understanding the efficiency of GPU algorithms for matrix-matrix multiplication. In: *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*. ACM, 2004. p. 133-137.

[11] ODELOWO, Babafemi O. a David V ANDERSON. Speech enhancement using extreme learning machines. 2017 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA) [online]. IEEE, 2017, , 200-204 [cit. 2017-12-14]. DOI: 10.1109/WASPAA.2017.8170023. ISBN 978-1-5386-1632-1. Dostupné z: http://ieeexplore.ieee.org/document/8170023/

[12] RODAN, Ali; TINO, Peter. Minimum complexity echo state network. *IEEE transactions on neural networks*, 2011, 22.1: 131-144.

[13] VAN HEESWIJK, Mark, et al. Advances in extreme learning machines. 2015.

[14] ZHANG, Lei; ZHANG, David. Evolutionary cost-sensitive extreme learning machine. *IEEE transactions on neural networks and learning systems*, 2017.

[15] ZHANG, Ruiquan, et al. Multivariate chaotic time series prediction based on improved extreme learning machine. In: *Control Conference (CCC), 2017 36th Chinese*. IEEE, 2017. p. 4006-4011.

[16] HUANG, Guang-Bin. An insight into extreme learning machines: random neurons, random features and kernels. *Cognitive Computation*, 2014, 6.3: 376-390.

[17] KUMAR, Sachin, et al. ELM variants comparison on applications of time series data forecasting. In: *Advances in Computing, Communications and Informatics (ICACCI), 2016 International Conference on*. IEEE, 2016. p. 1404-1409.

[18] JAEGER, Herbert. *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*. Bonn: GMD-Forschungszentrum Informationstechnik, 2002.

[19] HUANG, Gao, et al. Trends in extreme learning machines: A review. *Neural Networks*, 2015, 61: 32-48.

[20] GOLUB, Gene H.; REINSCH, Christian. Singular value decomposition and least squares solutions. *Numerische mathematik*, 1970, 14.5: 403-420.

[21] MURAKAMI, Kenji; AIBARA, Tsunehiro. An improvement on the Moore-Penrose generalized inverse associative memory. *IEEE transactions on systems, man, and cybernetics*, 1987, 17.4: 699-707.

[22] YAO, Wei, et al. Ensembles of echo state networks for time series prediction. In: *Advanced Computational Intelligence (ICACI), 2013 Sixth International Conference on*. IEEE, 2013. p. 299-304.

[23] SHI, Zhi-wei; HAN, Min. Ridge regression learning in ESN for chaotic time series prediction. *Control and Decision*, 2007, 22.3: 258.

Seznam použitých zkratek

CUDA	-	Compute Unified Device Architecture
AAN	-	Artificial Neural Network
BP	-	Back-Propagation
ELM	-	Extreme Learning Machine
ESN	-	Echo State Network
SVD	-	Singular Value Decomposition
CPU	-	Central Processing Unit
GPU	-	Graphic Processing Unit

Seznam příloh

DVD obsahující:

- Elektronická verze textu
- Zdrojové soubory programů
    - ELM pro CPU
    - ELM pro GPU
    - ESN pro CPU
    - ESN pro GPU