VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2018 Bc. Ondřej Pospíchal

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

NEURONOVÉ SÍTĚ S OZVĚNOU STAVU PRO PŘEDPOVĚĎ VÝVOJE FINANČNÍCH TRHŮ
ECHO STATE NEURAL NETWORK FOR STOCK MARKET PREDICTION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Ondřej Pospíchal**
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. Ondřej Pospíchal ***ID:*** 164370
Ročník: 2 ***Akademický rok:*** 2017/18

NÁZEV TÉMATU:

Neuronové sítě s ozvěnou stavu pro předpověď vývoje finančních trhů

POKYNY PRO VYPRACOVÁNÍ:

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 s dalšími algoritmy pro analýzu časových řad.

DOPORUČENÁ LITERATURA:

[1] Mihaela, Turcu. "Echo State Networks." (2013).

[2] Lahore, T., EchoStateNetwork, 2013, [Online] https://goo.gl/YoYxbo

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

Tato práce se zabývá neuronovou sítí s ozvěnou stavu a urychlením jejího učení implementací na grafický procesor. V teoretické části práce jsou obecně uvedeny neuronové sítě a několik vybraných typů neuronových sítí, ze kterých vychází síť s ozvěnou stavu. Dále jsou uvedeny další algoritmy používané pro analýzu časových řad a v neposlední řadě byly také stručně popsány nástroje, které byly použity v praktické části práce. Praktická část popisuje tvorbu akcelerované varianty sítě s ozvěnou stavu. Následně je popsána tvorba vstupních datových souborů reálných finančních indexů, na kterých byla poté síť s ozvěnou stavu a ostatní algoritmy testovány. Analýzou této akcelerované varianty bylo zjištěno, že její rychlost učení nesplnila teoretická očekávaní. Akcelerovaná varianta pracuje pomaleji, avšak s větší přesností. Analýzou výsledků měření dalších algoritmů bylo zjištěno, že nejvyšších přesností dosahují řešení pracující na principu neuronových sítí.

KLÍČOVÁ SLOVA

Síť s ozvěnou stavu, GPU, CPU, akcelerace, Java, předpověď, srovnání, prediktivní analýza, analýza časové řady.

ABSTRACT

This thesis deals with an echo state network and with acceleration of its learning by implementing the echo state network on a graphics processor. The theoretical part consists of the description of neural networks and some selected types of neural networks, on which is based the echo state network. After that, there are some other algorithms described used for time series analysis and last but not least, the tools that were used in the practical part of the thesis were briefly described. The practical part describes the creation of the accelerated version of the echo state network. After that, there is described the creation of input data sets of real financial indexes, on which the echo state network and the other algorithmns were then tested. By analyzing this accelerated version it was found that its learning speed did not reach the theoretical expectations. The accelerated version works slower, but with greater precision. By analyzing the results of the measurement of the other algorithmns it was found that the highest precision is achieved by solutions based on the neural network principle.

KEYWORDS

Echo state network, GPU, CPU, acceleration, Java, prediction, comparison, predictive analysis, time series analysis.

POSPÍCHAL, Ondřej *Neuronové sítě s ozvěnou stavu pro předpověď vývoje finančních trhů*: diplomová práce. BRNO: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2018. 59 s. Vedoucí práce byl doc. Ing. Radim Burget, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Neuronové sítě s ozvěnou stavu pro předpověď vývoje finančních trhů“ jsem vypracoval(a) 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(ka) uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil(a) autorská práva třetích osob, zejména jsem nezasáhl(a) nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom(a) následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

BRNO ....

podpis autora(-ky)

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.

BRNO ....

podpis autora(-ky)

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 realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

BRNO ....

podpis autora(-ky)

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 VÝPISŮ KÓDU

SEZNAM TABULEK

ÚVOD

Neuronové sítě jsou v současné době jedním z nejčastějších a nejdiskutovanějších řešení strojového učení a umělé inteligence. Typickým problémem, který bývá řešen neuronovými sítěmi, je například rozpoznávání objektů v obrazu, rozpoznávání zvuků nebo textů a v neposlední řadě se používají i pro předvídání časových řad. Právě předvídání časových řad je klíčovou vlastností pro předpovídání vývoje určitého finančního indexu, jako je například zlato, euro nebo stále aktuálnější kryptoměny, jejichž hlavním představitelem je bez pochyby Bitcoin. Předpověď určitého finančního indexu neuronovou sítí pak může být klíčovým aspektem v úspěšnosti obchodní strategie.

Pro tuto práci byla vybrána neuronová síť s ozvěnou stavu (anglicky echo state network), která vyniká ve zmíněné předpovědi časových řad. Bohužel i přes spoustu vynikajících vlastností má tento typ sítě i několik nedostatků, mezi které patří mnohdy nedostatečná rychlost učení. Tento problém může velmi degradovat řešení pro předpověď finančních trhů, které se velmi rychle mění. Díky tomuto nedostatku se může stát, že u určitého finančního indexu dojde k neočekávanému skoku, na který se síť bude učit určitou dobu. Tato doba však může znamenat zásadní ztrátu zisku, a proto je důležité tuto dobu, kterou se síť přeučuje, snížit na minimum. Tento problém by mohlo minimalizovat spouštění upravené aplikace zmíněné sítě s ozvěnou stavu na grafickém procesoru.

Výhodou grafických procesorů je, že jsou schopny určité složité matematické operace řešit v pouhém jednom taktu procesoru, což může znamenat velmi radikální zrychlení těchto operací.

V praktické části této práce byla naprogramována akcelerovaná varianta sítě s ozvěnou stavu pracující na grafickém procesoru (GPU). Tato vytvořená varianta vychází z varianty vzorové, která pracuje na klasickém procesoru (CPU). Obě zmíněné varianty byly následně porovnány v rychlosti učení a přesnosti předpovědi. Dále byly srovnány některé ostatní algoritmy, které se používají pro analýzu časových řad. Porovnání a testování obou variant sítě s ozvěnou stavu i ostatních algoritmů, probíhalo na základě vstupních datových souborů vybraných finančních indexů, které byly taktéž vytvořeny v praktické části této práce.

Základní informace o síti s ozvěnou stavu a několika dalších příbuzných sítích, naleznete v první kapitole. Druhá kapitola se zabývá několika dalšími algoritmy, které se používají pro analýzu časových řad. Třetí kapitola se věnuje vybraným nástrojům pro programování na grafických procesorech a některým dalším nástrojům, které byly použity v praktické části této práce. Čtvrtá kapitola popisuje samotnou implementaci akcelerované varianty sítě s ozvěnou stavu pro grafické procesory. V páté kapitole je rozebrána tvorba vstupních dat vybraných finančních indexů. Jak

již bylo zmíněno, tato data byla použita pro testování sítě s ozvěnou stavu a několika dalších algoritmů popsaných ve druhé kapitole. Šestá kapitola již srovnává výsledky akcelerované varianty s vzorovou variantou pro CPU a několika dalších zmíněných algoritmů. Poslední sedmá kapitola obsahuje stručné shrnutí a zhodnocení celého projektu.

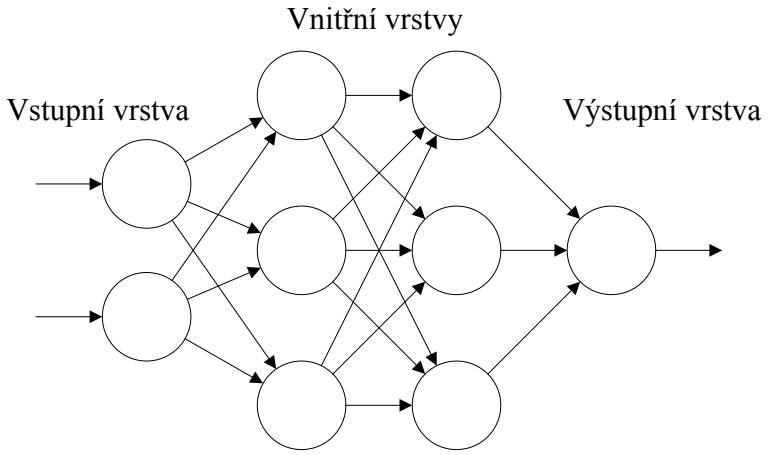
1 NEURONOVÉ SÍTĚ

Umělé neuronové sítě jsou jedním z výpočetních modelů používaných v oboru umělé inteligence. Zmíněné sítě jsou zpravidla realizovány formou softwarového řešení tak, jako je tomu i v případě této práce. Jejich funkce je inspirována funkcí lidského mozku. Stejně jako v případě lidského mozku, i neuronové sítě se nejdříve musí naučit na problém, který chceme, aby řešily. Neuronové sítě je možné učit dvojím způsobem, a to buď tzv. bez učitele nebo s učitelem.

Případ učení bez učitele umožňuje pouze několik typů neuronových sítí jako je např. Kohonenova mapa. V tomto případě má síť k dispozici pouze tréninkovou množinu a využívá soutěžní strategii učení. Základním principem je v podstatě soutěžení výstupních neuronů o to, který z nich bude aktivní.

V případě učení s učitelem má síť k dispozici tréninkovou a testovací množinu. Učení v tomto případě probíhá tak, že výstupy sítě jsou postupně porovnávány s požadovanými výstupy uloženými v tréninkové množině. Rozdíl tohoto porovnání je následně vyhodnocen a nastává úprava hodnot vah tak, aby rozdíl mezi výstupem sítě a výstupem uloženým v trénovací množině byl minimální. Testovací množina poté slouží k vyhodnocení správného natrénování dané sítě.

Strukturu umělých neuronových sítí je možné si představit jako síť umělých neuronů, jejichž vzorem je biologický neuron. Tyto neurony jsou vzájemně propojeny, díky čemuž si mohou navzájem předávat signály a transformovat je pomocí určitých přenosových funkcí. Mimo přenosové funkce je dalším důležitým parametrem tzv. váha, která každému propojení připisuje určitou důležitost. Neuron může mít libovolný počet vstupů, ale vždy má pouze jeden výstup, který je pak případně možné připojit jako další vstup jednomu nebo více dalším neuronům.



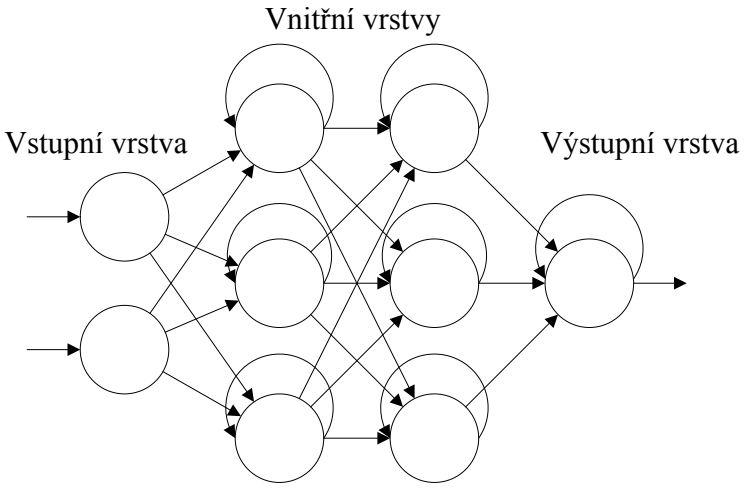
Obr. 1.1: Topologie úplně propojené dopředné neuronové sítě.

Neuronové sítě se nejčastěji používají k aproximaci funkcí, rozpoznávání a kompresi obrázků, zvuků nebo textů nebo pro předvídání časových řad (např. finančních indexů), jak bylo uvedeno v úvodu.[1]

Nejzákladnější typ neuronové sítě je dopředná neuronová síť (FFNN z anglického Feed Forward Neural Network), která je pro představu zobrazena na Obr. 1.1.

1.1 Rekurentní neuronové sítě

Rekurentní neuronové sítě se nejčastěji uvádí pod zkratkou RNN (z anglického Recurrent Neural Network) a jsou speciálním případem neuronových sítí. Na rozdíl od nejběžnější FFNN jsou sítě RNN schopny implementovat i časový kontext. Jinými slovy, na rozdíl od FFNN, která pouze šíří vstupní hodnoty směrem k výstupu sítě za působení přenosových funkcí a vah, je RNN schopna odrážet i vliv stavů, které určitému stavu předcházely díky zpětnovazebnímu přenosu signálu od vrstev vyšších zpět do vrstev nižších. Lépe je možné tuto skutečnost pochopit z rozdílu mezi Obr. 1.1 a Obr. 1.2. Vlastnost implementovat časový kontext mají RNN společné se sítěmi s ozvěnou stavu, které budou podrobněji představeny dále.[2]



Obr. 1.2: Topologie rekurentní neuronové sítě.

1.2 Stroje kapalných stavů

Stroje kapalných stavů, známé pod zkratkou LSM (z anglického Liquid State Machine), jsou ve své podstatě podobné sítím s ozvěnou stavu a vychází z rekurentních neuronových sítí. S oběma typy neuronových sítí sdílí několik vlastností. Například jsou schopny implementovat časový kontext, což je u LSM způsobeno nahrazením

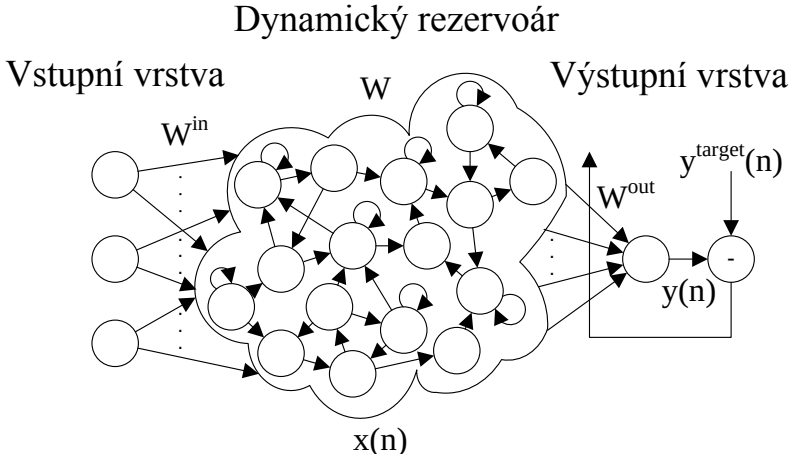
sigmoidních funkcí neuronů funkcemi prahovými. Spolu s tím, že je každý neuron také akumulační paměťovou buňkou, se pak signál šíří do následujícího neuronu až poté, co je dosažen práh zmíněné prahové funkce. Odtud plyne název těchto strojů kapalných stavů. Šíření signálu totiž připomíná šíření kapaliny, která se přelévá do druhé nádoby až po dosažení okraje nádoby první. Další společnou vlastností se sítěmi s ozvěnou stavu jsou náhodně propojené neurony v tzv. rezervoáru, který obsahuje i zpětnovazební propojení. To je popsáno v předchozí podkapitole.[3]

1.3 Síť s ozvěnou stavu

Síť s ozvěnou stavu (ESN z anglického Echo State Network) byla vyvinuta současně s LSM v roce 2002 Wolfgangem Massem a vychází z předchozích dvou typů sítí. Hlavním prvkem této sítě je dynamický rezervoár, který se skládá z určitého počtu naprosto náhodně propojených neuronů s náhodně nastavenými váhami. Právě díky zpětnovazebním spojením v rezervoáru je ESN schopna implementovat časový kontext stejným způsobem jako je tomu u RNN. Dále se ESN skládá z jednoho nebo více vstupních neuronů a obvykle jednoho výstupního neuronu. Topologie této sítě je zobrazena na Obr. 1.3.

Hlavní myšlenkou ESN je fakt, že vstupní signál vytváří v rezervoáru reakci, která by se dala přirovnat k ozvěně. Jedná se o to, že výstupní signál se neskládá jen z transformovaného signálu posledního vstupu, jako je tomu u dopředné neuronové sítě, ale skládá se také z transformovaných signálů všech předchozích vstupů.[4]

Stejně jako každá jiná neuronová síť se i ESN musí nejdříve naučit na problém, který má za úkol řešit. U ESN existují dva algoritmy učení, a to tzv. online a offline.



Obr. 1.3: Topologie sítě s ozvěnou stavu.

1.3.1 Algoritmus online učení sítě s ozvěnou stavu

Základní myšlenka algoritmu online učení spočívá v tom, že po každém průchodu vstupní hodnoty je získána výstupní hodnota, která je pak porovnána s žádanou výstupní hodnotou, což může být u časové řady například příští vstupní hodnota. Po tomto porovnání je proveden výpočet výstupních vah, které jsou následně upraveny. Tyto kroky jsou provedeny při průchodu každé další vstupní hodnoty, díky čemuž v tomto případě ESN nepotřebuje ukládat výstupní stavy do paměti. Níže jsou chronologicky popsány jednotlivé kroky online učení.

1. Vygenerování ESN se zadaným počtem neuronů avšak s naprosto náhodnými propojeními a náhodnými váhami těchto propojení.
2. Načtení vektoru vstupních hodnot do ESN, přičemž dimenze vektoru odpovídá počtu vstupních neuronů. V tomto kroku dochází k postupnému šíření vstupních hodnot směrem k výstupu sítě, přičemž v každém neuronu je vstupní hodnota transformována podle přenosové funkce neuronu. Zároveň je vstupní signál měněn i zpětnovazebním charakterem rezervoáru. Jinak řečeno aktuální vstupní signál je v rezervoáru měněn i „ozvěnami“ předchozích vstupních hodnot. Níže uvedený vztah popisuje výpočet vnitřní dynamiky stavů:

$$x(n+1) = tahn(u(n+1) \cdot W^{in} + x(n) \cdot W + y(n) \cdot W^{back}). \tag{1.1}$$

   Kde u je vektor vstupních hodnot, x je vektor vnitřních stavů a y je vektor výstupních hodnot. Dále W^{in}, W a W^{back} jsou vstupní váhy, vnitřní váhy a váhy zpětné vazby výstupních neuronů zpět do rezervoáru.
3. Dalším krokem je výpočet výstupní hodnoty ESN, která je vypočítána podle jednoduchého vztahu:

$$y(n+1) = x(n+1) \cdot W^{out}(n), \tag{1.2}$$

   kde W^{out} značí matici výstupních vah v čase n.
4. Následuje aktualizace výstupních vah. Hodnota na kterou jsou výstupní váhy aktualizovány je získána z rozdílu mezi výstupem sítě a požadovaným výstupem sítě. Výstupní váhy jsou aktualizované podle následující rovnice:

$$W^{out}(n+1) = W^{out}(n) + \eta \cdot x(n+1)^T \cdot e_y(n+1) + \gamma \cdot x(n)^T \cdot e_y(n), \tag{1.3}$$

   kde η je koeficient učení a γ je momentový koeficient. Hodnota těchto momentů odpovídá rozsahu [0,1], tak jako je tomu i u ostatních neuronových sítí.
5. Celý algoritmus se opakuje pro každý vstupní signál.[5]

1.3.2 Algoritmus offline učení sítě s ozvěnou stavu

Algoritmus offline učení se liší od online učení v tom, že síť nejdříve načte všechny vstupní hodnoty tréninkové množiny, čímž získá stavovou matici výstupních hodnot.

Poté jsou váhy dopočítány, nejčastěji metodou pseudoinverze. Při výpočtu výstupních vah hraje zásadní roli matice učitele, která obsahuje požadované výstupy sítě. Níže je popsán tento algoritmus podrobněji v jednotlivých krocích.

1. Stejně jako u online učení je třeba nejdříve vygenerovat ESN tak, aby byla splněna podmínka ozvěn předchozích stavů. Tudíž náhodně propojená síť se zpětnými vazbami.
2. Následuje cyklické načítání vstupních hodnot do ESN a aktivace vnitřní dynamiky rezervoáru, stejně jako je tomu u online učení. Změna nastává v tom, že se všechny výstupní hodnoty ukládají do stavové matice bez okamžitého upravování vah.
3. V tomto kroku dochází k vymazání počáteční paměti dynamického rezervoáru tím způsobem, že je vymazáno několik prvních výstupních hodnot ze stavové matice. Tyto výstupní hodnoty jsou totiž poznamenané náhodně vygenerovanými váhami (např. nulový stav) na počátku učení. Kolik prvních vstupních hodnot je vymazáno záleží na délce vstupní sekvence a povaze systému. Od určitého okamžiku lze předpokládat, že je u výstupních hodnot eliminován vliv počátečních stavů sítě po jejím vygenerování. Tímto postupem získáme upravenou stavovou matici.
4. Nyní dochází k výpočtu výstupních vah podle vztahu:

$$W^{out} = (M^{-1} \cdot T)^T, \tag{1.4}$$

   kde M^{-1} je upravená stavová matice po pseudoinverzi. Tzv. upravenou matici učitele T získáme odstraněním stejného počtu prvních hodnot, jako tomu bylo v minulém kroku při úpravě stavové matice.[5]

Nyní bude základně objasněna pseudoinverze. Ta je zapotřebí, protože upravená stavová matice zpravidla není čtvercová a nelze tak provést klasickou inverzi matice. Nejčastěji se používá tzv. Moore-Penroseova pseudoinverze, jejímž základem je singulární rozklad, který je dále uváděn pod zkratkou SVD (z anglického Single Value Decomposition). Tento SVD rozkládá matici M na 3 matice podle následujícího vzorce:

$$M = U\Sigma V^T, \tag{1.5}$$

kde U a V jsou unitární čtvercové matice a Σ je diagonální matice s nenulovými reálnými čísly na diagonále. Bohužel v případě že M není čtvercová, s čímž u pseudoinverze počítáme, tak Σ také není čtvercová, z čehož plyne problém, který bude uveden dále. Z uvedeného vzorce plyne, že matice V je získána v transponovaném tvaru.

SVD pseudoinverzní matice M^{-1} se pak rovná:

$$M^{-1} = V\Sigma^{-1}U^T, \tag{1.6}$$

z čehož plyne, že pokud chceme pseudoinverzi matice M, získáme ji singulárním rozkladem této matice a následnou inverzí všech 3 dílčích matic. Vzhledem k unitárnosti matic U a V lze tvrdit, že $U^{-1} = U^T$. To samé platí pro matici V. Díky tomu není třeba počítat u matic U a V inverzi, ale stačí jejich transpozice.

Problém nastává při pokusu o inverzi matice Σ. Vzhledem k tomu, že tato matice není čtvercová, mohlo by bystrého čtenáře napadnout, že bude třeba další pseudoinverze. Tímto způsobem bychom však mohli počítat pseudoinverzi do nekonečna. K vyřešení tohoto problému by případně mohla sloužit tzv. tenká pseudoinverze, ale ani to není efektivním řešením. Další možností je využít vlastností SVD a matice Σ. Z těchto vlastností, které jsou nad rámec tohoto textu, plyne, že abychom získali čtvercovou matici Σ, stačí pouze odstranit přebývající nulové sloupce nebo řádky. Pak již stačí provést běžnou inverzi této matice.[6][7]

2 DALŠÍ ALGORITMY PRO ANALÝZU ČASOVÝCH ŘAD

Pro analýzu časových řad se používají nejen určité typy neuronových sítí (několik z nich bylo popsáno v předchozí kapitole), ale i mnoho dalších algoritmů. Několik těchto algoritmů bude popsáno v této kapitole. Zmíněné algoritmy můžeme dělit na regresní techniky a techniky strojového učení.

2.1 Regresní techniky

Regresní techniky, resp. modely, jsou základem prediktivní analýzy. Fungují na principu matematických rovnic, které popisují vzájemné vztahy mezi prediktory a predikovanou proměnnou. Tento princip bude pravděpodobně nejlepší vysvětlit na následujícím příkladu lineární regrese, ve kterém budeme odhadovat výšku člověka v závislosti na jeho věku. Predikovanou proměnnou v tomto případě představuje výška a prediktor představuje věk. Pokud potvrdíme, že výška a věk člověka spolu souvisí, můžeme na základě věku odhadovat výšku člověka, případně naopak, pokud prediktorem bude naopak výška a predikovanou proměnnou věk. Výsledek predikované proměnné se pak odvíjí od různých parametrů regresní funkce a dalších složek, které jsou do modelu zahrnuty. Zmíněná lineární regrese je nejzákladnější regresní technikou a používá se k proložení souboru bodů v grafu přímkou. S ohledem na časové řady si lineární regresi můžeme představit jako průměr hodnot v určitém okamžiku. Z tohoto důvodu nebyla lineární regrese použita v praktické části.[8]

2.1.1 Logistická regrese

Logistická regrese je jednou z regresních technik, která se na rozdíl od lineární regrese zabývá odhadem pravděpodobnosti určitého jevu. Používá se pro kategorizaci modelu, jehož výstup nabývá pouze dvou hodnot, obvykle 0, pokud jev nenastal a 1, pokud jev nastal. V praxi lze logistickou regresi využít například k předpovědi výstupů typu student udělá/neudělá zkoušku, pacient je zdravý/nemocný atp. Prediktory související s výsledkem modelu není složité si představit. U úspěšnosti studenta na zkoušce se jedná například o dobu, po kterou se student učil, u predikce zdravotního stavu je možné si představit mnohem více prediktorů, jako je například krevní tlak, tělesná teplota, hladina cukru v krvi atp.[9]

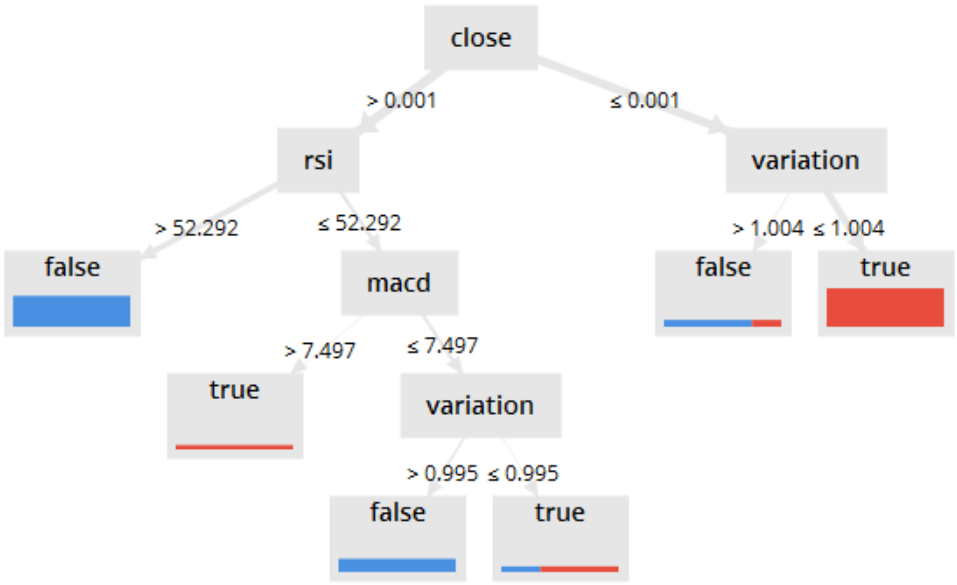
Kategorizaci modelu, jehož výstup nabývá více než dvou hodnot, nazýváme multinomickou logistickou regresí.

2.1.2 Rozhodovací stromy

Rozhodovací stromy slouží ke klasifikaci a predikci podobně jako Logistická regrese. Výsledkem je také jeden zvolený cílový atribut, který má zpravidla binární podobu a nabývá tedy tradičně hodnot 0 nebo 1. Rozhodnutí stromu o výsledku atributu poté závisí na všech existujících datových záznamech. Princip je možné zase přiblížit na příkladu, kdy se rozhodovací strom rozhoduje, zda je pacient nemocný nebo zdravý. Rozdílem oproti logistické regresi je aspekt „učení“. Rozhodovací strom je tedy schopný se naučit, které kombinace atributů zdraví pacienta (například krevní tlak, tělesná teplota, hladina cukru v krvi atp.) vedou k tomu, zda je pacient opravdu nemocný nebo zdravý.

Rozhodovací stromy mají poměrně široké využití. Pomocí regresních stromů, spadajících mezi rozhodovací stromy, lze predikovat i spojité veličiny, případně pokud klasifikujeme cílový atribut s více než dvěma kategoriemi, jedná se o variantu C4.5.

Ačkoliv již existují mnohem komplexnější algoritmy pro predikci a klasifikaci, rozhodovací stromy mají několik výhod, díky kterým jsou stále hojně využívané. Například nepotřebují žádnou speciální přípravu dat, díky čemuž jsou vhodné i pro začátečníky v oblasti klasifikace a predikce. Další výhodou je, že jejich výsledkem je graficky znázorněný strom, který lze zpravidla jednoduše vysvětlit nebo ukázat na prezentaci. Jeden takový rozhodovací strom je zobrazen na Obr. 2.1.



Obr. 2.1: Rozhodovací strom aplikovaný na finanční index.

Ostatní složitější algoritmy zpravidla tak jednoduché vysvětlit není. Uveďme si jako příklad neuronové sítě, které s několika desítkami neuronů obsahují mnoho

propojení a vah. U takové sítě je velmi těžké představit si, jak konkrétně je získána výsledná hodnota. Další neposlední výhodou rozhodovacích stromů je schopnost určit si klíčové atributy a naopak rozhodnout, které atributy příliš důležité nejsou, a které je možné úplně vypustit. U ostatních algoritmů se tento problém řeší pomocí selekce příznaků (anglicky feature selection nebo forward selection), která předchází samotnému algoritmu.[10]

Speciálním případem rozhodovacích stromů je náhodný les (anglicky Random forest), který je metodou pro klasifikaci a regresi stejně jako předchozí algoritmy. Hlavním rozdílem oproti předchozím rozhodovacím stromům je skutečnost, že náhodný les se skládá z několika rozhodovacích stromů. V oblasti predikce je výsledek náhodného lesu vypočten jako průměr výsledků dílčích rozhodovacích stromů. V oblasti klasifikace je pak výsledkem kategorie, která se vyskytuje nejčastěji a je tak modem klasifikovaných kategorií.[11]

2.2 Techniky strojového učení

Strojovým učením se nazývá určitý typ algoritmů, které spadají do oblasti umělé inteligence a jsou schopny se učit. Jinými slovy, jsou schopny naučit se řešení určitého problému pomocí určitých pravidel a technik (např. zpětné šíření chyby, anglicky back-propagation).

Učení je možné rozdělit na dvě podskupiny, a to učení s učitelem a učení bez učitele, jak již bylo přiblíženo v kapitole 1. Pro připomenutí, pokud se jedná o učení s učitelem, tak je součástí vstupních dat i správný výstup, který od algoritmu očekáváme. Pokud se jedná o učení bez učitele, pak vstupní data očekávaný výstup neobsahují. Následující popis se věnuje algoritmům s učitelem.

Jednou z nejvýznamnějších technik strojového učení jsou i neuronové sítě, které však byly popsány v první kapitole. Následující popis se tedy věnuje ostatním algoritmům.

2.2.1 Hluboké učení

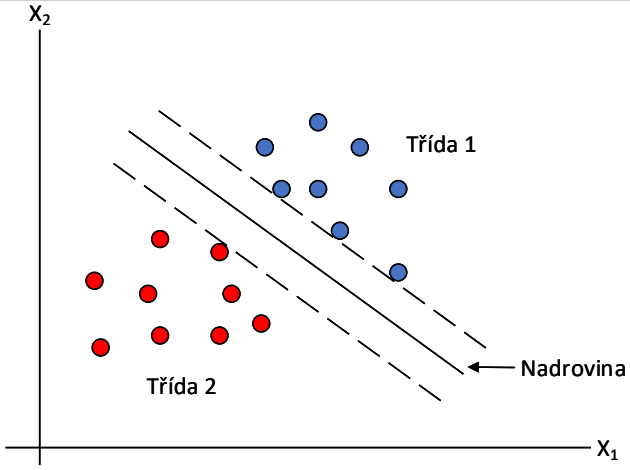
Popisy jednotlivých algoritmů se v mnoha oblastech překrývají a tak není jednoduché naprosto jednoznačně definovat každý z nich. I pro samotné hluboké učení (anglicky Deep learning) existuje několik definic. Všechny tyto definice se shodují pouze na dvou vlastnostech. První vlastností je, že použitý model hlubokého učení obsahuje několik vrstev nelineárních procesních jednotek. Druhou vlastností je, že každá následující vrstva využívá jako svůj vstup výstup předchozí vrstvy. Učení tímto způsobem tedy probíhá od vstupních vrstev k výstupním, případně je možné říct k vrstvám hlubším.

Hluboké učení je nejčastěji využíváno společně s neuronovými sítěmi, jak uvedená definice hlubokého učení může napovědět. Speciálním případem hlubokého učení jsou autoenkodéry. Jedná se o neuronové sítě, které mají stejné rozměry vstupní i výstupní vrstvy, ale zpravidla menší rozměr skrytých vrstev. Tyto autoenkodéry se nejčastěji používají pro kompresi a následnou obnovu dat. Jejich principem je naučení modelu pracovat tak, aby bylo z výstupních dat stále možné rozhodnout, jak vypadala data vstupní. Je nutné podotknout, že polovina topologie autoenkodéru data komprimuje a druhá je zase dekomprimuje. Topologie je tedy zpravidla souměrná. Pokud chceme tedy uložit komprimovaná data, výstupní vrstva bude zmíněnou redukovanou skrytou vrstvou. [12]

2.2.2 Stroje podpůrných vektorů (SVM)

Stroje podpůrných vektorů (anglicky Support vector machines) jsou technikou klasifikace dat do dvou tříd. Cílem tohoto algoritmu je nalézt nadrovinu, která rozdělí vstupní data v n-rozměrném prostoru. Optimální nadrovina pak rozděluje data takovým způsobem, že minimální vzdálenost dat obou tříd od nadroviny je co největší. Tento „pruh bez bodů“, který je tedy v optimálním případě co nejširší se pak nazývá pásmo necitlivosti nebo hraniční pásmo. Nejjednodušší lineární dvourozměrné rozdělení tříd je zobrazeno na Obr. 2.2.

Název této techniky pochází z podpůrných vektorů, což jsou právě ty body, které leží na okraji hraničního pásma. Tyto body definují nalezenou nadrovinu.



Obr. 2.2: Rozdělení dat pomocí optimální nadroviny.

Důležitou součástí této techniky je jádrová transformace (anglicky kernel transformation), která umožňuje převést původně lineárně neseparovatelnou úlohu na lineárně separovatelnou pomocí převedení dat do prostoru typicky vyšší dimenze. V takovém prostoru je již možné nalézt požadovanou nadrovinu.[13]

2.2.3 Vícevrstvý perceptron (MLP)

Vícevrstvý perceptron (anglicky Multilayer perceptron) je speciálním případem neuronové sítě a skládá se alespoň ze tří vrstev. Od obecné neuronové sítě se liší tím, že neobsahuje zpětné vazby a jedná se tak vždy o dopřednou neuronovou síť. Kromě vstupních neuronů používají všechny ostatní neurony algoritmu MLP nelineární aktivační funkci, čímž se také může odlišovat od ostatních neuronových sítí. MLP využívá pro své učení techniku backpropagation.[14]

V praktické části této práce je použit vícevrstvý perceptron s jednou skrytou vrstvou, která obsahuje přibližně dvakrát více neuronů, než vrstva vstupní. Tím se použitý vícevrstvý perceptron liší od použité dopředné neuronové sítě, jejíž skrytá vrstva obsahuje méně skrytých neuronů, než vrstva vstupní. Vícevrstvé perceptrony, jejichž skrytá vrstva obsahuje více neuronů, než vrstvy ostatní, jsou pak označovány jako „vanilkové“ neuronové sítě.[15]

2.2.4 Bayesovské sítě

Bayesovské sítě (anglicky Naive Bayes) patří do skupiny pravděpodobnostních klasifikátorů. Princip jejich fungování bude pravděpodobně nejlepší vysvětlit na následujícím příkladu: Chceme klasifikovat jablko jako druh ovoce. Jablko má na pohled několik základních vlastností. Například je kulaté, má asi 10 centimetrů v průměru a je červené. Bayesovská síť se pak domnívá, že každá z těchto vlastností přispívá nezávisle k pravděpodobnosti, že se opravdu jedná o jablko. Na základě těchto vlastností je tedy schopná určit s jakou pravděpodobností se jedná o jablko.

Bayesovské sítě jsou dnes stále používanou metodou například pro kategorizaci textů nebo v lékařské diagnostice. Při správném nastavení jsou schopné konkurovat i pokročilejším metodám jako jsou například již zmíněné stroje podpůrných vektorů.[16]

2.2.5 Metoda k nejbližších sousedů (k-NN)

Základní myšlenkou tohoto algoritmu je předpoklad podobnosti. Například mňouká jako kočka, chodí jako kočka, nejspíše se jedná o kočku. Tento algoritmus v podstatě klasifikuje data v prostoru na základě vzdálenosti od již klasifikovaných dat. Z toho vyplývá, že algoritmus potřebuje ke svému fungování již klasifikovaná data, aby na základě jejich podobnosti mohl klasifikovat další vzory. Jádrem tohoto algoritmu je výpočet vzdálenosti mezi vzorky, který je nejčastěji realizován euklidovskou metrikou:

$$d(a, b) = \sqrt{\sum_i (a_i - b_i)^2},$$

Je však možné využít jakoukoli jinou metriku (Chamberrovská, Kosínová atp.).[17]

Výhodou tohoto algoritmu je jednoduchost přidávání dalších dat. Tento model se totiž nemusí jakkoli trénovat. Nevýhodou tohoto algoritmu je nutnost uchování celé databáze trénovacích dat, díky čemuž může být tento algoritmus velice omezen při klasifikaci velkého objemu dat. Tento problém je možné řešit uložením pouze střední pozice bodů jednotlivých kategorií. Jedná se o tzv. centroid. Je vhodné však uložit i informaci z kolika bodů byl centroid vypočten, díky čemuž je následně možné později rozšířit trénovací data.[12]

3 POUŽITÉ NÁSTROJE

V této kapitole jsou uvedeny nástroje, které byly použity při tvorbě akcelerované varianty sítě s ozvěnou stavu pro grafické procesory. Dále je popsán nástroj, pomocí kterého byly vytvářeny vstupní soubory pro testování zmíněné sítě s ozvěnou stavu a ostatních algoritmů uvedených v kapitole č. 2. V neposlední řadě je představen nástroj, pomocí kterého byla měřena přesnost klasifikace zmíněných ostatních algoritmů.

3.1 Maven

Maven je nástroj vytvořený neziskovou organizací Apache Software Foundation. Tento nástroj je určen pro správu a sestavování neboli kompilaci aplikací. I přes to, že je možné Maven použít pro projekty vytvořené v různých programovacích jazycích, nejvíce je spojován s jazykem Java. Důvodem jeho vzniku byl pokus o zjednodušení sestavovacího procesu u projektu Jakarta Turbine. Ten sestával z několika projektů, z nichž každý měl vlastní knihovny a zdrojové soubory, které byly všechny poněkud odlišné. Cílem bylo standardizovat způsob sestavování projektů, sdílení knihoven mezi projekty a získání jasné definice z čeho se projekt skládá. Vznikl tak nástroj, který lze nyní použít pro vytváření a správu jakéhokoli projektu založeného na jazyce Java.[18]

Maven nedisponuje žádným grafickým uživatelským rozhraním. Pracuje pouze pod příkazovou řádkou, díky čemuž ho tak mohou využívat všechny nástroje, které dokáží komunikovat pomocí standardních vstupů.

Výchozí nastavení Mavenu vyžaduje určitou adresářovou strukturu. Kořenový adresář projektu musí obsahovat soubor pom.xml a adresář src, který obsahuje zdrojové soubory projektu, který má být zkompilován. Základní adresářovou strukturu lze měnit pomocí změny konfigurace v souboru pom.xml, který je pro kompilaci prostřednictvím nástroje Maven stěžejní. Dále bude stručně nastíněna struktura tohoto konfiguračního souboru.

Soubor pom.xml se skládá z několika částí. Jako první se obvykle uvádějí informace o projektu. Příkladem těchto informací mohou být řádky 2 až 5 ve výpisu 3.1. V případě potřeby je možné informace v této části libovolně rozšířit pomocí dalších příkazů, například o odkaz na webové stránky apod. Další část, velmi stručně zobrazená v řádcích 7 až 11, obsahuje všechny použité zásuvné moduly, cestu ke kompilačním souborům apod. Tato část je ve výpisu 3.1 zobrazena pouze názorně, protože každé nastavení jednotlivého zásuvného modulu obsahuje několik desítek řádků konfigurace. Poslední část obsahuje závislosti na externích knihovnách. Příklad závislosti

na jedné z knihoven je zobrazen v řádcích 13 až 19.[19]

```
1  <project>
2    <modelVersion>4.0.0</modelVersion>
3    <groupId>MojeSpolecnost</groupId>
4    <artifactId>MujProjekt</artifactId>
5    <version>0.0.1-SNAPSHOT</version>
6
7      <build>
8      <plugins>
9          ...
10     </plugins>
11   </build>
12
13  <dependencies>
14     <dependency>
15       <groupId>com.spolecnost</groupId>
16       <artifactId>knihovna</artifactId>
17       <version>1.2.3</version>
18     </dependency>
19   </dependencies>
20 </project>
```

Výpis kódu 3.1: Struktura konfiguračního souboru pom.xml.

3.2 Deeplearning4j

Deeplearning4j, dále uváděná pod oficiální zkratkou DL4J, je knihovna vyvinutá skupinou programátorů ze San Francisca a Tokia. Tato skupina je vedena Adamem Gibsonem. DL4J je určena pro programovací jazyk Java a Scala a obsahuje implementace těchto neuronových sítí:

- Boltzmanovy stroje
- Hluboké neuronové sítě
- Hluboké autokodéry
- Konvoluční sítě
- Rekurentní sítě
- Rekurzivní autokodéry
- Rekurzivní tensorové neuronové sítě

Tato knihovna obsahuje vlastní výpočetní knihovnu ND4J, která bude uvedena dále. Stejně jako ND4J i DL4J je uzpůsobeno práci na CPU i GPU a dokáže pracovat i s architekturou CUDA.

DL4J je vydaná pod licencí Apache 2.0 jako tzv. otevřený software (anglicky open-source nebo open software), což je označení softwaru, který je volně šiřitelný a je poskytnut zároveň s jeho zdrojovým kódem. Současně je tato knihovna k dispozici bez poplatku.[20]

3.3 ND4J

ND4J je vědecká knihovna napsaná v programovacím jazyce C++. Byla vyvinuta na podzim roku 2014 skupinou programátorů ze San Francisca, kteří vyvinuli také DL4J. Do této doby neexistovala žádná obdobná knihovna pro programovací jazyk Java, která by umožňovala programování aplikací pro GPU bez zásadních změn v kódu. Nutno podotknout, že pokud není k dispozici GPU, na kterém by mohl být spuštěn vytvořený kód, ND4J je schopné pracovat i na procesoru (CPU).[21]

ND4J je uzpůsobena především práci s lineární algebrou a maticemi. Objekt, kterým tato knihovna ztvárňuje matice, se nazývá INDArray. Jedná se o rozhraní, které obsahuje třídy N-dimenzionálních polí, které v kódu ztvárňují reálné a komplexní matice. Každá třída je obsažena dvakrát, a to jak pro CPU, tak pro GPU výpočty. Toto rozhraní obsahuje velké množství metod, mezi které patří i metoda pro singulární rozklad. Zmíněná metoda byla uvedena v podkapitole 1.3.2. Příklad syntaxe a vytvoření jednoduchého objektu INDArray je zobrazen ve výpisu 3.2.[22]

ND4J je stejně jako DL4J vydána jako open-source pod licencí Apache 2.0.[21]

```
1 INDArray NDpole = Nd4j.create(new float[] { 2.f, 4.f, 1.f, 3.f} new
      int[] { 2, 2 });
2 System.out.println(NDpole);
3 // Vypis v konzoli:
4 // [[2.00,  4.00],
5 // [1.00,  3.00]]
```

Výpis kódu 3.2: Konstruktor a vypsání objektu NDpole typu INDArray

3.4 TA4J

TA4J je zkratkou anglického názvu – Technical Analysis For Java. Z tohoto názvu vyplývá, že se jedná o knihovnu určenou k technické analýze v jazyce Java. Stejně jako zmíněné knihovny DL4J a ND4J je i knihovna TA4J vydána jako open-source, avšak pod licencí Massachusettského technologického institutu (MIT).

TA4J obsahuje více než 130 technických indikátorů, jako jsou například klouzavé průměry, RSI, MACD atp., díky čemuž je tato knihovna významným nástrojem v oblasti technické analýzy. Tyto indikátory jsou počítány z historických dat určitého finančního indexu. Zmíněná historická data pak musí být ve tvaru OHLC (Open, High, Low, Close), což znamená, že pokud mluvíme o určitém časovém intervalu, je třeba vkládat data, která obsahují 4 ceny z tohoto intervalu. A to otevírací, nejvyšší, nejnižší a konečnou, tedy uzavírací. Pro některé indikátory je třeba vložit i údaj o obchodovaném objemu za tento interval (anglicky volume).

Knihovna TA4J obsahuje mnoho dalších užitečných funkcí, jako je například nastavování pravidel pro nákup a prodej nebo test a následnou analýzu celé obchodní strategie.[23]

3.5 RapidMiner

RapidMiner je softwarový nástroj, který vznikl v roce 2001 na technické univerzitě v německém Dortmundu pod názvem YALE (Yet Another Learning Environment). Vyvinuli ho Ralf Klinkenberg, Ingo Mierswa a Simon Fischer z ústavu umělé inteligence. Po založení firmy Rapid-I Ingem Mierswem a Ralfem Klinkenbergerem byl v roce 2007 název softwaru změněn z YALE na stávající RapidMiner.[24] Roku 2013 byla zmíněná firma Rapid-I přejmenována taktéž na RapidMiner.

Jak již název může napovědět, RapidMiner se používá především na zpracování dat a textů, prediktivní analýzu nebo ke strojovému a hlubokému učení.

RapidMiner obsahuje grafické uživatelské rozhraní, což usnadňuje návrh a realizaci analytických pracovních postupů. Tyto pracovní postupy se v RapidMineru nazývají procesy. Každý tento proces je možné sestavit z množství bloků, které představují například metody strojového učení, křížovou validaci, dopřednou selekci příznaků apod. Nepřeberné množství funkcí je možné rozšířit o další zásuvné moduly prostřednictvím služby RapidMiner Marketplace nebo skriptů psaných v programovacím jazyce Python nebo R.[25]

4 IMPLEMENTACE AKCELEROVANÉ VARIANTY PRO GRAFICKÉ PROCESORY

Kapitola 4 popisuje vlastní postup při tvorbě akcelerované varianty vzorové ESN. Tato varianta byla vytvořena v programovacím jazyce Java. Úkolem této varianty je zjistit potenciál sítě s ozvěnou stavu fungující na GPU.

Samotná práce na tvorbě této akcelerované varianty probíhala v první části na osobním počítači v programu Eclipse. Vzhledem k tomu, že tento osobní počítač nedisponuje příslušnou grafickou kartou Nvidia potřebnou pro základní funkci akcelerované varianty, bylo třeba vývoj následně přesunout na školní server VUT, který disponuje grafickou kartou GeForce GTX 1080 od výrobce Nvidia. K tomuto přesunu došlo až po důkladném prostudování varianty pro CPU.

Zmíněná CPU varianta funguje na principu algoritmu online učení, který byl uveden v kapitole 1.3.1. Bohužel, implementace sítě s ozvěnou stavu na GPU, která funguje na tomto principu, není možná. Vzhledem k tomu, že jsou při tomto algoritmu výstupní váhy upravovány po každém průchodu nového vstupu celou sítí, je nutné, aby úprava vah probíhala sekvenčně. Síla funkce GPU však spočívá v tom, že GPU je schopné řešit určité problémy paralelně při pouhém jednom taktu CPU. Z tohoto důvodu bylo nutné přistoupit k přeprogramování vzorové sítě s ozvěnou stavu na princip algoritmu offline učení, který byl rozebrán v kapitole 1.3.2.

4.1 Přesun vývoje na server

Po prostudování CPU varianty s dílčími změnami kódu došlo k přesunutí dalšího postupu na VUT server zvaný Gravy. Při tomto kroku vznikl problém, jak přesunout velké množství knihoven a jak co nejefektivněji zprovoznit vývojové prostředí na vzdáleném serveru. Nejdříve byla snaha vytvořit jeden velký spustitelný soubor formátu JAR s knihovnami a spouštět kód s definovanou cestou, kde přesně má každou s knihoven najít. K tomu měly sloužit vytvořené spouštěcí skripty. Bez nich by bylo nutné umístění každé knihovny psát ručně pokaždé, když by bylo potřeba spustit nebo kompilovat kód, a to i přes to, že se nacházejí ve stejném adresáři.

Toto řešení se ale nakonec ukázalo jako zdlouhavé, neefektivní a zbytečně složité vzhledem k velkému množství knihoven a tím pádem i velkého množství ukazatelů na jejich umístění. Nakonec tento krok velmi usnadnil nástroj Maven, který byl uveden v kapitole 3.1. Díky němu stačilo překopírovat knihovny z osobního počítače na Gravy a vytvořit konfigurační soubor pom.xml, kde byl nadefinován adresář se všemi knihovnami. Dále byly do pom.xml nadefinovány závislosti na knihovny, které adresář neobsahoval. Maven si tyto knihovny během kompilace sám stáhl z centrálního

repositáře Mavenu, na který je uveden odkaz také v pom.xml.

Kompilace a následné spuštění kódu pomocí Mavenu bylo vždy provedeno sledem několika příkazů, které budou nyní vysvětleny. Jako první byl vždy použit příkaz *mvn clean*, který vymaže zkompilované soubory po posledním spuštění testovaného kódu. Tím je zajištěno, že nedojde k částečnému přepsání zkompilovaných souborů minulého testu, což by v krajním případě mohlo působit problémy při samotném běhu programu. Po tomto kroku si můžeme být jisti, že je adresář pro zkompilované třídy projektu prázdný a je tedy možné pokračovat ke kompilaci. Ta byla spuštěna příkazem *mvn compile*. V tomto kroku jsou vytvořeny nové zkompilované třídy, následně připravené ke spuštění, které bylo provedeno pomocí příkazu *mvn exec:java*. Po každém zadání některého ze zmíněných příkazů nás Maven informuje, zda vše proběhlo úspěšně systémovou hláškou „Build success“, nebo neúspěšně systémovou hláškou „Build failure“. Při neúspěchu zároveň vypíše do příkazového řádku, k jakým dílčím chybám došlo.

Místo příkazu *mvn compile* lze použít *mvn package*. Tento příkaz v sobě obsahuje zmíněný kompilovací příkaz. Navíc však kompilovaný kód skládá do distribuovatelného souboru, například typu JAR, což kompilaci v našem případě prodlouží z několika sekund na více než minutu. Doba sestavování se pak odvíjí i od počtu knihoven. Z těchto důvodů bylo využíváno spíše příkazu *mvn compile*. [26]

4.2 Tvorba akcelerované varianty

Po zprovoznění vývojového prostředí na Gravy mohla začít samotná tvorba akcelerované varianty. Základem této varianty je knihovna ND4J zmíněná v kapitole 3.3.

Jak již bylo zmíněno dříve, aby akcelerovaná varianta mohla pracovat na GPU, bylo třeba přeprogramovat algoritmus učení z typu online na typ offline. Toho bylo docíleno změnou v hlavní spouštěcí metodě celého vzorového projektu pro CPU. Celý průběh vzorové implementace sítě s ozvěnou stavu, který je ztvárněn jedním cyklem for, se dělí na 3 sekce. Prvních 100 vzorů vstupní matice o velikosti 1500x1 vzorů je načteno sítí. Dochází k aktivaci sítě a vytváření ozvěn v rezervoáru. Dalších 900 vzorů slouží k učení sítě formou inicializace vah a posledních 500 vzorů slouží k testování výstupu sítě. Přepracována byla pouze sekce s učením sítě.

Při průběhu učení bylo nejdříve třeba získat výstupní stavovou matici, která byla získána postupným průchodem 900 vstupních vzorů celou sítí s ozvěnou stavu a následným ukládáním výstupních vzorů do pole. Následně bylo třeba z této stavové matice vytvořit objekt typu INDArray, se kterým pracuje zmíněná knihovna ND4J. Toho bylo docíleno upravením rozměrů pole se stavovou maticí tak, aby bylo možné

použít metodu *Nd4j.create*. Výsledkem této metody je stavová matice ve formě INDArray s rozměry 900x1. Uvedený postup je zobrazen ve výpisu 4.1. Stejný postup bylo třeba provést i pro matici učitele, tedy pro matici žádaných výstupů, která byla získána ze vstupní matice posunutím v čase o jeden vzor.

```
1 double[] flatOut = ArrayUtil.flattenDoubleArray(outputArray);
2 int[] shapeOut = {train_period,1};
3 INDArray outData = Nd4j.create(flatOut,shapeOut,'f');
```

Výpis kódu 4.1: Transformace stavové matice do typu INDArray.

Následuje výpočet počtu sloupců a řádků stavové matice a první část Moore-Penroseovy pseudoinverze, tedy singulární rozklad. Ten musí být proveden, protože stavová matice není čtvercová a nemůže tak být provedena inverze, která je potřeba k získání výstupních vah podle kapitoly 1.3.2. Výpočet sloupců a řádků stavové matice a její následný singulární rozklad je znázorněn ve výpisu 4.2. V řádcích 6 až 8 je možné si všimnout, že je třeba nejdříve vygenerovat matice, do kterých bude SVD uložen. Funkce pro SVD, jejíž provedení se nachází na posledním řádku zmíněného výpisu, si sama tyto matice nevytvoří. Důležitá je matice *extender*, která slouží k úpravě rozměrů. Bez ní by neodpovídaly rozměry matic při konečném součinu a pseudoinverze by neproběhla.

```
1  INDArray Input = outData;
2  int nRows, nColumns;
3  nRows = Input.rows();
4  nColumns = Input.columns();
5  INDArray S, U, V, VT, extender, D, output1, UTran, temp;
6     S = Nd4j.zeros(1, nColumns);
7     U = Nd4j.zeros(nRows, nRows);
8     VT = Nd4j.zeros(nColumns, nColumns);
9     extender = Nd4j.zeros(nColumns,Math.abs(nRows-nColumns));
10
11    Lapack svd = new JcublasLapack();
12    svd.gesvd(Input, S, U, VT);
```

Výpis kódu 4.2: Funkce singulárního rozkladu.

Nyní tedy máme rozloženou stavovou matici, následně je třeba u jejích dílčích prvků provést inverzi. Vzhledem k tomu, že stavová matice je ve formě pouze jednoho sloupce, vyjde matice S pouze jako jednoprvková. Bohužel knihovna ND4J neumí provést inverzi jednoprvkové matice. Tento problém je vyřešen tak, jak je zobrazeno ve výpisu 4.3.

```
1  double Pom;
2  Pom=1/S.getDouble(0,0);
3  S.putScalar(0 , Pom);
4  temp = Nd4j.hstack(S, extender);
5  UTran = U.transpose();
6  output1 = temp.mmul(UTran);
```

Výpis kódu 4.3: Inverze jednoprvkové matice S.

Zmíněný jediný prvek je vyňat z matice S a je provedena jeho inverze převrácením zlomku. Následně je tato inverzní hodnota vrácena zpět do matice S. V dalším kroku je matice S rozšířena maticí *extender*, protože je nutné, aby odpovídaly rozměry matice S a mohlo dojít k vynásobení s transponovanou maticí U. Tím je získána stavová matice M po pseudoinverzi, tedy M^{-1}. Matice V není úmyslně započtena do pseudoinverze, protože v našem případě je stavová matice pouze sloupcová a matice V by díky tomu měla vycházet rovna 1. Bohužel funkce *svd* vrací buď matici V rovnou nule, nebo rovnou číslu v řádu asi 10^{20}. Správnost výsledku pseudoinverze byla několikrát ověřena matematickým online nástrojem WolframAlpha.

Zbývá již pouze získat výstupní váhy vynásobením matice M^{-1} s maticí učitele a celý výsledek transponovat. Tyto váhy je třeba nakonec implementovat do sítě s ozvěnou stavu. Toho je dosaženo kódem zobrazeným ve výpisu 4.4. První dva řádky obsahují výpočet výstupních vah. Další 2 řádky upravují váhy do takového tvaru, aby je bylo možno implementovat do ESN. Poslední řádek pak předává výstupní váhy funkci *next*, která byla upravena tak, aby implementovala váhy do ESN.

```
1  INDArray Wout = output1.mmul(targetData);
2  INDArray WoutT=Wout.transpose();
3
4  double[] OutWeights=new double[output_dimension];
5  OutWeights[0] = WoutT.getDouble(0,0);
6  output = esn.next(input, target_output, OutWeights);
```

Výpis kódu 4.4: Výpočet a nastavení výstupních vah ESN.

5 TVORBA VSTUPNÍCH DAT

Pro předpověď vývoje finančního trhu, a tedy i pro predikční analýzu vytvořenou pomocí strojového učení, jsou potřeba již označkovaná trénovací a testovací data. Každý z algoritmů strojového učení potřebuje nejdříve vědět jaká data má klasifikovat nebo předpovídat a jaký má být výsledek této klasifikace nebo predikce. Proto bylo třeba stáhnout z internetu a následně označkovat data zvoleného finančního indexu. Pro testování sítě s ozvěnou stavu a ostatních algoritmů byly zvoleny indexy:

- XAUUSD - vývoj ceny zlata v dolarech.
- EURUSD - vývoj ceny eura v dolarech.
- BTCUSD - vývoj ceny kryptoměny Bitcoin v dolarech.
- LSKBTC - vývoj ceny kryptoměny Lisk v závislosti na kryptoměně Bitcoin.

Zmíněné finanční indexy byly staženy z webového portálu www.investing.com. Tento portál byl zvolen, protože obsahoval všechny zvolené kryptoměny, a to ve formě OHLC, která byla zmíněna v podkapitole č. 3.4. Forma OHLC byla nutná z důvodu pozdější tvorby indikátorů pomocí knihovny TA4J.

Data každého z výše uvedených indexů byla stažena za posledních několik let v denním časovém intervalu. Data indexu zlata (XAUUSD) odpovídají časovému rozsahu od 1.1.2013 do 28.2.2018, data indexu eur a kryptoměny Bitcoin (EURUSD, BTCUSD) odpovídají časovému rozsahu od 1.1.2014 do 28.2.2018. Poslední index, kryptoměny Lisk, odpovídá časovému rozsahu od 26.5.2016 do 28.2.2018. Důvodem krátkého časového rozsahu dat kryptoměny Lisk je, že tato kryptoměna je velmi mladá a vznikla až v roce 2016.

Každý jednotlivý řádek, již uvedených stažených dat, obsahuje datum, otevírací cenu, uzavírací cenu, nejvyšší cenu a nejnižší cenu za jednotlivý den. U indexů kryptoměn (BTCUSD a LSKBTC) data obsahovala také obchodovaný obsah (anglicky volume). U indexu zlata (XAUUSD) a eura (EURUSD) bohužel obchodovaný obsah nebyl dostupný, místo něj poslední položka v řádku obsahuje změnu ceny v procentech.

Po stažení dat všech zvolených finančních indexů, bylo potřeba označkovat data podle nákupních a prodejních signálů. Pro jednoduchost bylo rozhodnuto, že pozitivní nákupní signál, kdy cena indexu stoupá, bude označena číslem 1 a negativní nákupní signál, kdy cena indexu klesá, bude označena číslem -1. V případě negativního nákupního signálu je možné index nakupovat například zápornými čísly a vydělat tak i na pádu indexu. Signály, které nebyly investičně zajímavé byly značeny číslem 0. Tyto značky, označující nákup, byly zapsány do oddělených souborů z důvodu jednodušší manipulace s daty. Každému finančnímu indexu tedy v tuto chvíli náležely dva soubory, a to soubor s denními OHLC daty a soubor se značkami nákupních signálů.

5.1 Výpočet finančních indikátorů

Pro přesnější analýzu bylo rozhodnuto o rozšíření vstupních dat o finanční indikátory. Díky tomuto kroku jsou schopny testované algoritmy provádět komplexnější analýzu nebo predikci.

K výpočtu indikátorů byla použita již zmíněná knihovna TA4J. Jako vzor byl použit kód, který byl poskytnut vedoucím této práce doc. Ing Radimem Burgetem, Ph.D. Tento kód se skládal ze dvou tříd pro výpočet indikátorů z dat OHLC kryptoměny Bitcoin v závislosti na kryptoměně Etherium. Tyto dvě třídy byly následně přepracovány a rozšířeny o další třídu. Výsledkem je projekt o třech třídách, který umožňuje vytvořit jeden soubor s indikátory a značkami nákupu pro každý z finančních indexů. Tyto tři třídy budou následně blíže popsány.

Hlavní třída zvaná *IndicatorsToCsv* je spouštěcí třídou pro vytvoření zmíněných indikátorů a obsahuje hlavní (anglicky main) spouštěcí metodu. Na začátku funkce této třídy se načítají vstupní soubory do paměti, čemuž odpovídají řádky 2 a 3 zobrazené ve výpisu 5.1.

```
1  public static void main(String[] args) throws
       FileNotFoundException {
2    TimeSeries series = CsvTicksLoader.loadOHLCVSeries("data/EURUSD.
       csv");
3    ArrayList labels = CsvTicksLoader.loadLabels("data/LabelsEURUSD.
       csv");
```

Výpis kódu 5.1: Načítání vstupních souborů do paměti v třídě *IndicatorsToCsv*.

Tyto zmíněné dva řádky v podstatě odkazují na druhou třídu, která je zvaná *CsvTicksLoader* a obsluhuje metodu *loadOHLCVSeries*. Tato metoda se stará o načtení vstupních dat do paměti. Načítání poté probíhá řádek po řádku. Současně tato třída každý řádek ještě rozdělí na jednotlivé ceny indexu (OHLC) a obchodovaný objem (volume), případně rozdíl v procentech. Každou tuto datovou jednotku pak ukládá do pole typu *Arraylist*. Hlavní část této metody, která se o toto načítání vstupních souboru stará, je zobrazena ve výpisu 5.2.

```
1    InputStream stream = new FileInputStream(file);
2    List<Tick> ticks = new ArrayList<>();
3    CSVReader csvReader = new CSVReader(new InputStreamReader(stream
       , Charset.forName("UTF-8")), ',', '"', 1);
4     String[] line;
5     while ((line = csvReader.readNext()) != null) {
6       LocalDateTime zonedate = LocalDateTime.parse(line[0]+" "+line
```

```
            [1], DATE_FORMAT);
7       ZonedDateTime date = zonedate.atZone(ZoneId.systemDefault());
8           double open = Double.parseDouble(line[3]);
9           double high = Double.parseDouble(line[4]);
10          double low = Double.parseDouble(line[5]);
11          double close = Double.parseDouble(line[2]);
12          double volume = Double.parseDouble(line[6]);
13    ticks.add(new BaseTick(date, open, high, low, close, volume));}
```

Výpis kódu 5.2: Stěžejní část metody *loadOHLCVSeries* z třídy *CsvTicksLoader*.

Zobrazený případ se týká vstupního souboru s historickými daty ceny určitého finančního indexu. Analogicky byl načítán i soubor se značkami nákupu, který obsahoval pouze datum a značku nákupu na jednom řádku. Proto místo řádků 8 až 12 z výpisu 5.2 metoda pro načtení souboru se značkami nákupu, nazvaná *loadLabels*, obsahovala pouze jeden řádek, který načítá zmíněnou značku. Pro ujasnění je část metody *loadLabels*, zobrazena ve výpisu 5.3 a zmíněný řádek tak odpovídá třetímu řádku tohoto výpisu.

```
1  LocalDateTime zonedate = LocalDateTime.parse(line[0]+" "+line[1],
       DATE_FORMAT);
2  ZonedDateTime date = zonedate.atZone(ZoneId.systemDefault());
3        double label = Double.parseDouble(line[2]);
4        labels.add(label);
```

Výpis kódu 5.3: Část metody *loadLabels* z třídy *CsvTickLoader*.

Po načtení dat do paměti již probíhá výpočet jednotlivých indikátorů. O tento výpočet se stará knihovna TA4J a programátorovi tedy zbývá pouze definovat konstruktory pro jednotlivé indikátory. Definice několika indikátorů je zobrazena ve výpisu 5.4.

```
1  ClosePriceIndicator closePrice = new ClosePriceIndicator(series);
2  TypicalPriceIndicator typicalPrice = new TypicalPriceIndicator(
       series);
3  PriceVariationIndicator priceVariation = new
       PriceVariationIndicator(series);
4  SMAIndicator shortSma = new SMAIndicator(closePrice, 8);
5  SMAIndicator longSma = new SMAIndicator(closePrice, 20);
6  RSIIndicator rsi = new RSIIndicator(closePrice, 14);
```

Výpis kódu 5.4: Konstruktory několika indikátorů ve třídě *IndicatorsToCsv*.

Po vytvoření indikátorů pomocí konstruktorů zbývá pouze zápis těchto indikátorů do výstupního souboru spolu se značkami nákupních signálů. Toho je docíleno pomocí dvou externích tříd zvaných *StringBuilder* a *BufferedWriter*.

Nejdříve je využito třídy *StringBuilder*, pomocí které je tvořena forma výstupního souboru. Jak je vidět ve výpisu 5.5, prvním řádkem výstupního souboru bude legenda, která popisuje, který indikátor náleží určitému sloupci.

```
1 StringBuilder sb = new StringBuilder("label,timestamp,close,typical
    ,variation,sma8,sma20,ema8,ema20,ppo,roc,rsi,williamsr,atr,atr
    -1,macd,macd-1,emaMacd,emaMacd-1,DiffEmaMacd,sd\n");
2  final int nbTicks = series.getTickCount();
3         for (int i = 0; i < nbTicks - 4; i++) {
4             sb.append(labels.get(i)).append(',')
5             .append(series.getTick(i).getEndTime()).append(',')
6             .append(getMarketOpportunity(series, i)).append(',')
7             .append(typicalPrice.getValue(i)).append(',')
```

Výpis kódu 5.5: Tvorba formy výstupního souboru pomocí třídy *StringBuilder*.

V řádcích 2 až 7 uvedeného výpisu je pak možné vidět část cyklu, který načítá do mezipaměti formu celého výstupního souboru po řádcích. Zbývá už jen nachystaný formát „vytisknout“ do souboru. O to se stará zmíněný *BufferedWriter* a *FileWriter*, jejichž funkci je možné vidět ve výpisu 5.6.

```
1  BufferedWriter writer = null;
2         try {
3             writer = new BufferedWriter(new FileWriter("indicators.
                  csv"));
4             writer.write(sb.toString());
5         } catch (IOException ioe)
```

Výpis kódu 5.6: Výpis do souboru realizovaný třídami *BufferedWriter* a *FileWriter*.

Výsledkem je pak soubor *indicators.csv*, který obsahuje 21 sloupců dat. První sloupec obsahuje značku nákupu, druhý datum a zbývajících 19 odpovídá vypočítaným indikátorům.

Poslední, třetí třída, která ještě nebyla popsána má za úkol převrácení časové řady. Důvodem tohoto převrácení je dosud opomíjený fakt, že data, která byla stažena z internetu a tvořila vstup pro projekt pro výpočet indikátorů, byla seřazena od nejnovějšího záznamu k nejstaršímu. Tento aspekt by později dělal problémy při načítání vstupních dat neuronovou sítí. Pokud je cílem sítě predikce následujícího prvku (například zítřejší cena zlata), je třeba aby síť načítala data chronologicky

a ne zpětně. Tento problém bylo možné řešit dvěma způsoby, a to buď načítání prvků sítí od posledního k prvnímu, a nebo vytvořit další třídu, která pořadí řádků obrátí. Byla tedy zvolena druhá, elegantnější možnost.

Tato zmíněná třída, nazvaná *DataInvertor*, nejprve načítá znovu celý soubor *indicators.csv* do paměti řádek po řádku a poté ho zase vypisuje do souboru, taktéž řádek po řádku, ale rekurzivně. Výsledkem je soubor s obráceným pořadím řádků. Jádro funkce této třídy je zobrazeno ve výpisu 5.7.

```
1  for (String x = in.readLine(); x != null; x = in.readLine())
2     {   line++;
3         list.add(x);
4         lin=list.size();  }
5  line--;
6  BufferedWriter writer = null;
7  writer = new BufferedWriter(new FileWriter("indicators.csv"));
8  while (line != -1)
9     {   writer.write(list.get(line).toString());
10        writer.write("\n");
11        System.out.println(list.get(line));
12        line--;  }
13 writer.close();
```

Výpis kódu 5.7: Jádro třídy DataInvertor.

5.2 Konečná podoba vstupních dat

V předchozí podkapitole byla popsána tvorba vstupního souboru pro neuronovou síť s ozvěnou stavu. Tento soubor byl vytvořen pro každý zvolený finanční index. Jednalo se tedy celkem o čtyři datové soubory finančních indexů, které byly zmíněny na začátku této kapitoly. Nyní bude popsána forma zmíněných vytvořených souborů a další změny v závislosti na vlastnostech algoritmů pro analýzu časových řad.

Po posledním kroku z předchozí podkapitoly (obrácení pořadí řádků) byl pro každý z indexů vytvořen soubor, ve kterém vždy první sloupec obsahuje značku nákupu o hodnotách 0, 1 a -1, druhý sloupec obsahuje datum, kterému náleží všechny hodnoty v daném řádku, a zbývajících 19 sloupců obsahuje vypočítané indikátory. Celý jednotlivý soubor je nyní seřazen od nejstaršího záznamu po nejnovější. Část souboru, odpovídající indexu zlata, je zobrazena na Obr. 5.1.

Každý z těchto čtyř souborů obsahuje více než 1000 řádků, až na soubor odpovídající kryptoměně Lisk v závislosti na kryptoměně Bitcoin. Soubor indexu zlata (XAUUSD) obsahuje 1341 řádků, soubor indexu eura (EURUSD) obsahuje 1099

řádků, soubor indexu kryptoměny Bitcoin obsahuje 1497 řádků a soubor kryptoměny Lisk v závislosti na kryptoměně Bitcoin obsahuje 641 řádků. Poslední soubor tedy obsahuje méně záznamů z důvodu kratšího časového rozsahu, což bylo vysvětleno na začátku této kapitoly.

```
1  label,timestamp,close,typical,variation,sma8,sma20,ema8,ema20,ppo,roc,rsi,williamsr,atr,sd,macd
2  0.0,2013-01-07T00:00+01:00[Europe/Prague],0.005904821655933368,1650.70000000000045474735088646 4,
3  0.0,2013-01-08T00:00+01:00[Europe/Prague],-0.007227255672626846,1655.8466666666665454007064302764,
4  1.0,2013-01-09T00:00+01:00[Europe/Prague],6.765941480756488E-4,1658.3833333333336364982339243 09,
5  0.0,2013-01-10T00:00+01:00[Europe/Prague],-0.010671734971921287,1669.3166666666668182491169621 55
6  0.0,2013-01-11T00:00+01:00[Europe/Prague],0.007429330020582925,1664.5133333333333212067373096943,
7  0.0,2013-01-14T00:00+01:00[Europe/Prague],-0.002520896102633936,1667.2133333333333666814723983407
8  0.0,2013-01-15T00:00+01:00[Europe/Prague],-0.007087301010394736,1676.7466666666666696983156725764
9  0.0,2013-01-16T00:00+01:00[Europe/Prague],-3.628798142054801E-4,1679.1833333333333181750883037845
10 0.0,2013-01-17T00:00+01:00[Europe/Prague],-0.004923069479388487,1683.3800000000000333481390650074
11 0.0,2013-01-18T00:00+01:00[Europe/Prague],0.0022233737437034867,1687.4833333333333484915783628821
12 0.0,2013-01-21T00:00+01:00[Europe/Prague],-0.0036613766776307346,1688.6833333333333181750883037 84
13 0.0,2013-01-22T00:00+01:00[Europe/Prague],-0.0012300787250383752,1691.5000000000000000000000000 0
14 0.0,2013-01-23T00:00+01:00[Europe/Prague],0.0038102438556067045,1688.0666666666666060336865484715
15 -1.0,2013-01-24T00:00+01:00[Europe/Prague],0.010818831133724591,1672.9600000000000363797880709171
16 0.0,2013-01-25T00:00+01:00[Europe/Prague],0.005466989651985305,1662.2233333333333575865253806114,
17 0.0,2013-01-28T00:00+01:00[Europe/Prague],0.0022640184511450298,1656.5633333333333515232273687919
18 0.0,2013-01-29T00:00+01:00[Europe/Prague],-0.005387637490692622,1661.2699999999999818101059645414
19 0.0,2013-01-30T00:00+01:00[Europe/Prague],-0.008020920983334645,1674.4266666666666575717196489374
20 0.0,2013-01-31T00:00+01:00[Europe/Prague],0.008172259115095675,1667.3833333333332878585982446869,
```

Obr. 5.1: Ukázka vstupního souboru indicatorsXAUUSD.csv.

Po vytvoření všech čtyř zmíněných souborů byly na všechny tyto vstupní soubory aplikovány algoritmy pro analýzu časových řad, které byly popsány v kapitole 2. Tyto algoritmy měly za úkol pouze na základě hodnot indikátorů „uhádnout“, zda se jednalo o nákupní signál pozitivní (1), nákupní signál negativní (-1) a nebo o obchodně nezajímavý signál (0). Analýza výsledků tohoto testu ale ukázala, že vzhledem k velkému množství nul, ve sloupci nákupních značek, není možné na tyto vstupní soubory zmíněné algoritmy aplikovat. Důvodem je snaha těchto algoritmů dosáhnout co nejlepší přesnosti. Té je však dosaženo, pokud algoritmus hádá vždy obchodně nezajímavý signál (0).

Z tohoto důvodu byly vstupní soubory následně upraveny tím způsobem, že byly odstraněny všechny řádky, jejichž nákupní značka byla 0. Díky tomu bylo docíleno pouze binominálního rozdělení dat. Tedy rozdělení, kdy nákupní značka nabývala pouze hodnot 1 nebo -1. Tím bylo docíleno usnadnění kategorizace algoritmů pouze do dvou kategorií, místo tří. Velkou nevýhodou tohoto kroku však bylo značné ztenčení vstupních souborů, což není příliš výhodné pro variantu GPU sítě s ozvěnou stavu, jejíž velkou výhodou mělo být zpracování velkého množství dat za kratší dobu, než u varianty CPU.

Každý jednotlivý soubor po zmíněném ztenčení obsahuje pouze okolo 50 řádků. Soubor indexu zlata (XAUUSD) obsahuje 71 řádků, soubor indexu eura (EURUSD) obsahuje 45 řádků, soubor indexu kryptoměny Bitcoin obsahuje 59 řádků a soubor kryptoměny Lisk v závislosti na kryptoměně Bitcoin obsahuje 27 řádků. Je však

nutné podotknout, že díky tomuto ztenčení si již algoritmy pro analýzu časových řad nemohou „usnadnit“ práci tím, že by predikovaly pokaždé nulovou nákupní značku. Tento aspekt podtrhuje i fakt, že počet pozitivních i negativních nákupních značek (1 a −1) v každém souboru je přibližně stejný. Pokud by tedy algoritmy predikovaly vždy pouze jednu z hodnot, docílily by přesnosti pouze okolo 50%. Ve srovnání s předchozím případem, kdy při předchozí podobě datových souborů předpovídaly zmíněné algoritmy vždy nulu a získaly tak přesnost i nad 95%, je pro ně nyní takový přístup nevýhodný, a proto se raději „snaží“ o predikci. Část upraveného souboru, který odpovídá indexu zlata, je zobrazena na Obr. 5.2.

```
label,timestamp,close,typical,variation,sma8,sma20,ema8,ema20,ppo,roc,rsi,williamsr,atr,sd,macd
1.0,2013-01-09T00:00+01:00[Europe/Prague],6.765941480756488E-4,1658.383333333333636498233924309,0
-1.0,2013-01-24T00:00+01:00[Europe/Prague],0.010818831133724591,1672.960000000000363797880709171,
-1.0,2013-02-11T00:00+01:00[Europe/Prague],0.010807916068255392,1654.226666666666687888209708035 0,
1.0,2013-02-22T00:00+01:00[Europe/Prague],-0.00299133477860912,1579.236666666666787932626903057,0
-1.0,2013-03-26T00:00+01:00[Europe/Prague],0.003900622212156129,1599.903333333333454599293569724,
-1.0,2013-04-11T00:00+02:00[Europe/Prague],-0.001678675654037802,1560.956666666666630286785957496
1.0,2013-04-18T00:00+02:00[Europe/Prague],-0.010784425019866047,1377.689999999999787784569586317,
-1.0,2013-05-13T00:00+02:00[Europe/Prague],0.012424240291142603,1435.026666666666424134746193886,
-1.0,2013-06-20T00:00+02:00[Europe/Prague],0.05743658859574197,1300.903333333333454599293569724,0
1.0,2013-07-10T00:00+02:00[Europe/Prague],-0.011842650103519632,1258.109999999999757468079527219,
-1.0,2013-08-01T00:00+02:00[Europe/Prague],0.010840419826859621,1315.603333333333151434392978748,
1.0,2013-08-09T00:00+02:00[Europe/Prague],-0.00196442041740005,1312.186666666666484767726312081,0
-1.0,2013-09-02T00:00+02:00[Europe/Prague],0.0010873977846082468,1390.883333333333636498233924309
1.0,2013-10-16T00:00+02:00[Europe/Prague],-6.391512071968062E-4,1279.356666666666454451236252983,
-1.0,2013-10-31T00:00+01:00[Europe/Prague],0.014407145944388418,1328.663333333333363649823392431,
1.0,2013-12-31T00:00+01:00[Europe/Prague],-0.0077247783875051615,1201.419999999999969683509940902
1.0,2014-02-11T00:00+01:00[Europe/Prague],-0.013537906137183971,1286.363333333333060484922801455,
-1.0,2014-03-19T00:00+01:00[Europe/Prague],0.018683280723826422,1339.700000000000454747350886464,
1.0,2014-04-07T00:00+02:00[Europe/Prague],0.004736590750730026,1298.880000000000333481390650074,0
```

Obr. 5.2: Ukázka vstupního souboru indicatorsXAUUSD.csv po úpravě.

5.3 Úprava sítě s ozvěnou stavu pro nová vstupní data

Po vytvoření všech čtyř vstupních datových souborů, které odpovídají jednotlivým zvoleným finančním indexům, bylo třeba upravit i obě varianty sítě s ozvěnou stavu, aby byly schopny datové soubory načíst a zpracovat.

Zmíněné načítání vstupního souboru bylo přepracováno z původní realizace objektem *Scanner* na novou realizaci objektem *CSVReader*, pomocí kterého byly ostatně načítány soubory typu CSV i v předchozí podkapitole při tvorbě indikátorů. Tento přístup byl zvolen z důvodu snazší manipulace s dvojrozměrnou maticí. Předchozí přístup totiž pracoval pouze se sloupcovou maticí a tak nepotřebovat sofistikovanější řešení než zmíněný *Scanner*. Navíc původní vstupní soubor pro CPU variantu s názvem *mg30.dat* není typu CSV. Zmíněné načítání vstupního souboru do paměti realizuje cyklus *while* zobrazený ve výpisu 5.8.

```
1  List<List<Double>> data = new ArrayList<List<Double>>();
2  CSVReader reader = null;
3  try {
4      reader = new CSVReader(new FileReader("data/indicatorsXAUUSD.
           csv"));
5      String[] line;
6      line = reader.readNext();
7      while ((line = reader.readNext()) != null) {
8        ArrayList<Double> radek = new ArrayList<Double>();
9        for (int y = 2; y < input_dimension+2 ; y++) {
10         radek.add(Double.parseDouble(line[y]));
11       }
12       data.add(radek);
13     }
14   } catch (IOException e){
15     e.printStackTrace(); }
```

Výpis kódu 5.8: Načítání vstupního souboru sítí s ozvěnou stavu.

Ve zmíněném výpisu je možné vidět, že vstupní soubor se ukládá do dvojrozměrného pole *data* typu *Arraylist*. Toho je docíleno vždy pomocí postupného načítání hodnot v řádku do pole *radek* a následně je celé toto pole, ztvárňující jeden určitý řádek, uloženo do pole s názvem *data*. Toto zmíněné pole *data* je tak v podstatě polem jednorozměrných polí.

Řádek číslo 6 ve výpisu 5.8 znamená přeskočení legendy, která je v každém ze vstupních souborů na prvním řádku. Také je nutné vysvětlit, proč se načítá každý řádek až od třetí hodnoty, což je patrné z deklarace *y=2*, která se nachází v řádku číslo 9 zmíněného výpisu. Jedná se o to, že první hodnotou v řádku je vždy značka nákupu a druhou hodnotou je datum. Ani jednu z těchto hodnot není žádoucí vkládat jako vstup do sítě s ozvěnou stavu.

V neposlední řadě je nutné upravit počet vstupních a výstupních neuronů a rozdělit data na části odpovídající trénovací a testovací množině. Toho je docíleno pouze úpravou deklarace hodnot, které leží vždy na začátku třídy pro danou variantu sítě s ozvěnou stavu. Tuto deklaraci je možné vidět ve výpisu 5.9. Hned v prvním řádku tohoto výpisu se nachází deklarace počtu trénovacích a testovacích cyklů sítě, tzv. epoch. Následující 4 řádky pak odpovídají nastavení topologie sítě, tedy nastavení počtu vstupních neuronů, výstupních neuronů, neuronů v rezervoáru a skrytých neuronů. Další tři řádky (6 až 8) poté rozdělují vstupní soubor na data aktivační, tréninková a testovací.

```
1  int Epoch = 1000;
```

```
2 int input_dimension = 19;
3 int output_dimension = 19;
4 int liquid_dimension = 100;
5 int readout_dimension = 10;
6 int washout_period = 9;
7 int train_period = 30;
8 int test_period = 18;
```

Výpis kódu 5.9: Deklarace topologie a rozdělení vstupních dat.

Dále bylo třeba u obou variant změnit formu trénování a testování dat. U varianty CPU byla vždy zpracována jedna vstupní hodnota, díky čemuž byla získána jedna výstupní hodnota. Nyní, pokud chceme předpovídat více hodnot v jednom kroku, čemuž odpovídá i počet vstupních a výstupních neuronů, je třeba změnit zpracování jedné hodnoty na pole několika hodnot. Toho bylo vždy docíleno cyklem *for* tak, jak je pro ilustraci zobrazeno ve výpisu 5.10. Naštěstí učící algoritmus u varianty CPU je již na tento případ připraven, tudíž stačilo doplnit pouze několik *for* cyklů do těla spustitelné třídy této varianty.

```
1 for (int u = 0; u < output_dimension; u++){
2      target_output[u] = data.get(t+1).get(u); }
3 output = esn.next(input, target_output);
```

Výpis kódu 5.10: Tréninkový krok u varianty CPU.

U varianty GPU bylo toto nastavení o trochu složitější. Učení zde probíhá jiným způsobem než u varianty CPU, což bylo popsáno v podkapitole 1.3.2. U varianty GPU byl zjištěn problém s velikostí matice S. Tento problém byl popsán v podkapitole 4.2 a jeho řešení pak zobrazeno ve výpisu 4.3. Jedná se o to, že pokud varianta GPU obsahuje více než jeden výstupní neuron, tak již nezískáme sloupcovou matici na vstupu pseudoinverze, ale klasickou dvourozměrnou matici, jejímž druhým rozměrem bude počet výstupních neuronů. Díky tomu získáme matici S jako řádkovou matici, místo jednoprvkové matice. Tím pádem není možné řešení, zobrazené ve zmíněném výpisu 4.3. Metoda pseudoinverze byla tedy u varianty GPU přepracována. Tato změna je zobrazena ve výpisu 5.11, ve kterém je možné vidět, že již bylo třeba počítat inverzi matice S. Další změnou je fakt, že matice V nyní nevychází rovna jedné, tudíž je nutné s ní počítat při výpočtu pseudoinverze.

```
1 D = Nd4j.diag(S);
2 DInv = InvertMatrix.invert(D, true);
3 VT = V.transpose();
```

```
4 temp = VT.mmul(DInv);
5 temp = Nd4j.hstack(temp, extender);
6 UTran = U.transpose();
7 output1 = temp.mmul(UTran);
```

Výpis kódu 5.11: Změna výpočtu pseudoinverze u varianty GPU.

6 SROVNÁNÍ VARIANT SÍTĚ S OZVĚNOU STAVU A OSTATNÍCH ALGORITMŮ

V této kapitole budou srovnány výsledky měření přesnosti predikce a rychlosti trénování obou zmíněných variant sítě s ozvěnou stavu (CPU a GPU). Dále budou srovnány ostatní algoritmy pro analýzu časových řad, které byly uvedeny v kapitole 2 a také bude diskutováno použití těchto algoritmů v porovnání se sítí s ozvěnou stavu (ESN).

6.1 Srovnání variant sítě s ozvěnou stavu

Pro srovnání varianty CPU a varianty GPU sítě s ozvěnou stavu byly použity nezkrácené datové soubory čtyř finančních indexů, o kterých pojednává kapitola 5. Tvorba těchto nezkrácených datových souborů pak byla popsána v podkapitole 5.1.

Důvod proč bylo použito nezkrácených datových souborů, které obsahují i řádky jejichž nákupní značka je rovna 0 a jedná se tedy o obchodně nezajímavý signál, je ten, že časový rozdíl mezi jednotlivými řádky je vždy stejný – jeden den. Není tedy porušena časová návaznost, která byla porušena právě zkrácením těchto souborů, o čemž pojednává kapitola 5.2. Řádky, které obsahují nákupní značku rovnou 0, nemohou být vzhledem k charakteru finančního indexu rozloženy v souboru rovnoměrně. Pokud jsou tedy tyto řádky odstraněny, tak vzniknou časově nepravidelné záznamy, které jsou pro predikci mnohem náročnější.

V tabulce č. 6.1 je možné vidět výsledek měření přesnosti predikce varianty CPU a varianty GPU sítě s ozvěnou stavu. Výsledná tabulka obsahuje zprůměrované hodnoty z celkem tří měření, díky čemuž je snížena chyba měření.

Hodnoty uvedené v tabulce jsou zprůměrovanou absolutní chybou každé z predikovaných hodnot, které odpovídají všem 19 indikátorům v každém dílčím datovém souboru určitého finančního indexu.

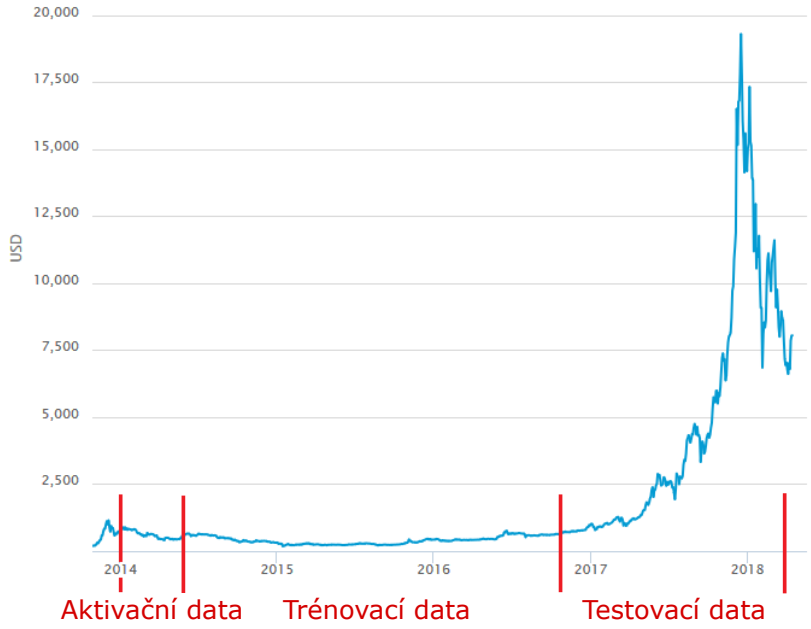
	Varianta GPU		**Varianta CPU**	
Druh chyby	**Trénovací**	**Testovací**	**Trénovací**	**Testovací**
XAUUSD	$1{,}8970 \cdot 10^{-4}$	$3{,}3192 \cdot 10^{-2}$	$3{,}54825 \cdot 10^{-3}$	$3{,}3052 \cdot 10^{-2}$
EURUSD	$5.5743 \cdot 10^{-5}$	$4{,}3135 \cdot 10^{-3}$	$1{,}1548 \cdot 10^{-3}$	$4{,}4728 \cdot 10^{-3}$
BTCUSD	$1{,}0292 \cdot 10^{-4}$	5,4446	$1{,}8967 \cdot 10^{-3}$	5,4445
LSKUSD	$5{,}7774 \cdot 10^{-5}$	$1{,}6941 \cdot 10^{-2}$	$2{,}2248 \cdot 10^{-3}$	$2{,}1886 \cdot 10^{-2}$

Tab. 6.1: Absolutní chyba variant CPU a GPU při přesnosti predikce indikátorů.

Podle této tabulky je možné říci, že varianta GPU pracuje s menší absolutní chybou než varianta CPU, což dokazuje jak porovnání sloupců s trénovací chybou, kdy varianta GPU vykazuje absolutní chybu o řád menší, tak i porovnání důležitější, testovací chyby, kdy je varianta GPU ve většině případů o něco přesnější. Jedinou výjimkou je finanční index zlata (XAUUSD), u kterého varianta GPU vykazuje chybu větší.

Nastavení obou sítí během zmíněného měření bylo totožné. Vstupní data byla rozdělena vždy na tři části. První část odpovídala stu řádků určených k aktivaci sítě a ze zbývajících dat byly vždy asi dvě třetiny určeny pro trénování sítě a jedna třetina pro testování sítě. Topologie pak odpovídala 19 predikovaným indikátorům, z čehož může být jasné, že obě varianty disponovaly 19 vstupními a 19 výstupními neurony. Pro velikost rezervoáru bylo zvoleno 100 neuronů. Dále bylo vypozorováno, že pro jednotlivá měření stačí nastavení 200 epoch, po kterých se chyba ustálila na změřených hodnotách. Při více epochách se chyba měnila již pouze zanedbatelně. Koeficient učení byl pak nastaven na 0.1.

U finančního indexu kryptoměny Bitcoin (BTCUSD) je možné vidět velmi vysokou testovací chybu v porovnání s ostatními finančními indexy. Důvodem je velmi strmý nárůst ceny této kryptoměny od roku 2017. Data, která náleží tomuto nárůstu, však leží až v testovací části datového souboru. To znamená, že síť s ozvěnou stavu nebyla na tento nárůst naučena a díky tomu předpovídá chybné výsledky. Zmíněný strmý nárůst je podle grafu kryptoměny Bitcoin nepředpověditelný z historických dat, které tomuto nárůstu předcházejí. Zmíněný graf je zobrazen v Obr. 6.1.



Obr. 6.1: Graf ceny kryptoměny Bitcoin v amerických dolarech.

Další měření, jehož výsledky jsou zobrazeny v tabulkách 6.2 a 6.3, bylo zaměřeno na rychlost učení. Cílem tohoto měření bylo ukázat výhodu varianty GPU, která podle teoretického předpokladu měla v rychlosti učení předčit variantu CPU. K tomu ale bohužel nedošlo z několika důvodů. Prvním důvodem je fakt, že obě varianty načítají vstupní data postupně, aby byly aktivovány echo stavy. V tomto kroku probíhají výpočty stavů všech neuronů a varianta GPU tak oproti variantě CPU nemá kde uspořit čas. Po tomto kroku varianta CPU počítá a nastavuje váhy, což není výpočetně ani časově náročné. Tímto končí funkce učícího algoritmu u varianty CPU. U varianty GPU se sice nenastavují váhy v každém kroku, ale jednou za učící algoritmus jsou přesouvány datové matice z paměti programu na GPU, což může zabrat až několik sekund a níže uvedené tabulky tento aspekt potvrzují. Zejména pak měření číslo 1, u kterého obě varianty pracovaly s nejmenší datovou maticí. Poměrově je však časový rozdíl mezi učením jednotlivých variant největší. To potvrzuje, že pokud pracujeme s malými maticemi, jednoznačně se vyplatí použití CPU.

	Varianta CPU				
Číslo měření	1.	2.	3.	4.	5.
Rozměr datové matice	58x19	1340x57	6700x19	16080x19	6700x57
Trénovacích řádků	30	800	5500	14000	5600
Testovacích řádků	18	440	1000	2000	1000
Trénovacích vzorů	570	45 600	104 500	266 000	319 200
Testovacích vzorů	342	25 800	19 000	38 000	57 000
Počet epoch	1000	1000	1000	100	100
Celkový čas [m:s,ms]	00:02,873	01:14,300	05:00,724	01:14,428	00:39,677

Tab. 6.2: Měření doby učení varianty CPU v závislosti na rozměru vstupní matice.

	Varianta GPU				
Číslo měření	1.	2.	3.	4.	5.
Rozměr datové matice	58x19	1340x57	6700x19	16080x19	6700x57
Trénovacích řádků	30	800	5500	14000	5600
Testovacích řádků	18	440	1000	2000	1000
Trénovacích vzorů	570	45 600	104 500	266 000	319 200
Testovacích vzorů	342	25 800	19 000	38 000	57 000
Počet epoch	1000	1000	1000	100	100
Celkový čas [m:s,ms]	00:22,660	02:48,014	12:59,106	04:54,533	01:49,781

Tab. 6.3: Měření doby učení varianty GPU v závislosti na rozměru vstupní matice.

Dalším důvodem je, že varianta GPU sice neprovádí výpočty vah po každém načtení řádku vstupních dat jako varianta CPU, ale jednou za učící algoritmus má za úkol provést spoustu časově náročných výpočtů včetně pseudoinverze, která se sama skládá z několika dalších výpočetně náročných operací, jako je zejména singulární rozklad.

Toto měření probíhalo na datech finančního indexu zlata (XAUUSD), která byla rozkopírovávána do šířky i délky, aby byly získány matice o větších rozměrech. Toto kopírování bylo provedeno až do maximálních rozměrů, dokud matice D, která je diagonalizovanou maticí S, nebyla singulární. Pokud by matice D byla singulární, znamenalo by to, že by obsahovala lineárně závislé řádky nebo sloupce, díky čemuž by k této matici D pak neexistovala inverzní matice, která je však nutná pro výpočet pseudoinverze. Bez inverzní matice D^{-1} pseudoinverze neproběhne.

Díky zmíněnému rozkopírovávání vstupních dat bylo bezpředmětné u tohoto měření sledovat současně i přesnost predikce. Jednalo se tedy o experiment zaměřen pouze na celkový čas běhu obou zmíněných variant v závislosti na rozměru vstupní datové matice.

Pokud tedy celkově porovnáme variantu CPU a variantu GPU podle výše uvedených tabulek, tak zjistíme, že varianta GPU je schopna dosáhnout menší absolutní chyby predikce než varianta CPU. S daty finančních indexů však pracuje v průměru více než dvakrát pomaleji, než varianta CPU.

Vzhledem k přednostem GPU by se měl časový rozdíl mezi variantami s rostoucími dimenzemi vstupních matic (zejména blížícím se čtvercové matici) snižovat, avšak vzhledem k naměřeným výsledkům nelze tvrdit, že by GPU varianta mohla být schopna předčit variantu CPU.

6.2 Srovnání přesnosti predikce ostatních algoritmů pro analýzu časových řad.

Pro ostatní vybrané algoritmy, které byly představeny v kapitole č. 2, byla připravena klasifikační úloha, jejíž podstatou bylo rozlišení pozitivního nebo negativního nákupního signálu, z hodnot vypočítaných indikátorů. Tvorbě zmíněných indikátorů se věnovala kapitola č. 5.

Pro tuto klasifikační úlohu byly použity zkrácené datové soubory, o jejichž podobě pojednávala podkapitola č. 5.2. Zároveň je v této podkapitole uveden i důvod použití právě zkrácených dat místo dat s nulovými nákupními značkami. Ve zkratce se jedná o to, že vzhledem k vysokému počtu nul, náležících nákupní značce, oproti ostatním hodnotám (1 a -1), si zmíněné algoritmy usnadňovaly práci a klasifikovaly vždy této proměnné hodnotu 0. Tím získaly uspokojivou přesnost více než 95%.

Proto bylo rozhodnuto o odstranění těchto řádků a klasifikování pouze do dvou skupin – pozitivní a negativní nákupní signál (1 a −1).

Pro porovnání zmíněných ostatních algoritmů, sloužících k analýze časových řad, byl využit program RapidMiner. Tento nástroj byl stručně popsán v kapitole 3.5. Pomocí nástroje RapidMiner byla dále upravena vstupní data pro toto měření a to tím způsobem, že hodnoty značky nákupního signálu byly změněny na pravda a nepravda místo 1 a −1. Tím byla eliminována jakákoliv možnost průměrování predikované hodnoty nákupní značky zmíněnými algoritmy.

V tabulce č. 6.4 jsou zobrazeny výsledky zmíněné úlohy. Tato úloha byla zaměřena na ověření přesnosti predikce vybraných algoritmů. Předmětem predikce byla zmíněná nákupní značka, která odpovídá signálu, kdy je výhodné nakoupit vybraný finanční index v kladných nebo záporných číslech. Finanční indikátory, podle kterých zmíněné algoritmy predikují jednotlivou nákupní značku, odpovídají dnu predikované nákupní značky. Jinak řečeno, indikátory podle kterých algoritmy predikují hodnotu nákupní značky vždy odpovídají stejnému řádku vstupního souboru jako predikovaná značka.

	XAUUSD	**EURUSD**	**BTCUSD**	**LSKBTC**
Logistická regrese	87,14%	**95,50%**	88,33%	88,33%
Rozhodovací strom	81,43%	84,50%	86,67%	88,33%
Náhodný les	88,57%	86,50%	90,00%	81,67%
Hluboké učení	**90,00%**	92,50%	95,00%	81,67%
Dopředná neuronová síť	87,14%	93,50%	**96,67%**	**96,67%**
Stroj podpůrných vektorů	55,71%	58,50%	44,67%	43,33%
Vícevrstvý perceptron	84,29%	91,00%	91,67%	91,67%
Bayesovská síť	74,29%	56,00%	86,67%	78,33%
Metoda k-NN	71,43%	58,50%	61,33%	60,00%

Tab. 6.4: Srovnání přesnosti predikce ostatních algoritmů.

Jak je ve zmíněné tabulce č. 6.4 vidět, všechny algoritmy se v celku úspěšně pokoušely o predikci hodnoty nákupní značky. Jediný algoritmus, který úplně selhal jsou stroje podpůrných vektorů (SVM). I přes vyzkoušení veškerého množství nastavení, které RapidMiner pro stroje podpůrných vektorů podporuje. Výsledek, který je uveden ve zmíněné tabulce je nejlepším možným dosaženým výsledkem. Žádný z typů SVM tedy v tomto případě nepředpovídal požadovanou nákupní značku. Tuto situaci značně zlepšila selekce příznaků (anglicky feature selection), která byla vložena před SVM. Po bližším zkoumání však bylo zjištěno, že selekce příznaků vybírá v každém ze zkrácených datových souborů finančních indexů odlišné indikátory.

Tomu odpovídá i pokus algoritmu SVM o predikci pouze u indexu zlata a kryptoměny Bitcoin (XAUUSD a BTCUSD). Právě díky tomu bylo zjištěno, že u SVM velmi záleží na vybraných prediktorech. U indexu zlata a kryptoměny Bitcoin selekce příznaků zvolila vždy dva stejné indikátory a to indikátor *close* a *williamsr*. U indexu eura (EURUSD) byly vybrány vždy indikátory *close* a *variation* a u indexu kryptoměny Lisk (LSKBTC) byly vybrány indikátory *close* a *rsi*.

Cílem této selekce příznaků je, jak již název napovídá, výběr prediktorů (v našem případě finančních indikátorů), které mají nejmenší podíl na správné klasifikaci algoritmů. Tyto prediktory jsou následně odstraněny. Jedná se tedy o předzpracování dat umožňující zvýšení přesnosti predikce algoritmů, které po zmíněné selekci příznaků následují.

V případě, že jsou na sobě některé prediktory nějakým způsobem závislé, je velmi výhodné tuto selekci použít. Může totiž docházet k problému, že množství na sobě závislých prediktorů bude „mást“ algoritmus, který je pro predikci aplikován. K tomu dochází z důvodu, že na sobě závislé prediktory obsahují všechny v podstatě stejnou informaci a stačí tak použít jediný z nich. Selekce příznaků tak zjednodušuje model úlohy odstraněním nevýznamných prediktorů, což má za následek zkrácení doby učení algoritmů, a zároveň zamezuje zmatení algoritmů na sobě závislými daty.[27]

Algoritmus, který má ve výše uvedeném měření nejlepší výsledky je dopředná neuronová síť. Velmi dobrých výsledků také dosáhlo hluboké učení a vícevrstvý perceptron. Je tedy možné říci, že nejvyšších přesností v tomto případě dosahují techniky fungujících na principu neuronových sítí.

V tabulce č. 6.5 je možné vidět stejné srovnání jako v tabulce č. 6.4 s tím rozdílem, že před každý z algoritmů byla nyní vložena selekce příznaků. Nastavení jednotlivých algoritmů zůstalo beze změny.

	XAUUSD	**EURUSD**	**BTCUSD**	**LSKBTC**
Logistická regrese	**90,00%**	96,00%	96,67%	**100,00%**
Rozhodovací strom	84,29%	93,50%	90,00%	86,67%
Náhodný les	81,43%	93,50%	93,33%	86,67%
Hluboké učení	87,14%	96,00%	93,33%	91,67%
Dopředná neuronová síť	88,54%	96,00%	**98,33%**	**100,00%**
Stroj podpůrných vektorů	74,29%	58,50%	80,00%	58,33%
Vícevrstvý perceptron	88,57%	96,00%	96,67%	**100,00%**
Bayesovská síť	87,14%	93,00%	93,33%	95,00%
Metoda k-NN	54,29%	**98,00%**	78,33%	45,00%

Tab. 6.5: Srovnání přesnosti predikce ostatních algoritmů s předzpracováním dat.

Po přidání předzpracování vstupních dat formou selekce příznaků stále dosahuje nejlepších predikčních výsledků dopředná neuronová síť, v jejímž závěsu je vícevrstvý perceptron. Metodu hlubokého učení však překonala nejjednodušší technika ze zvolených algoritmů, kterou je logistická regrese. Velmi zajímavé jsou také výsledky k-NN, kdy u indexu eur (EURUSD) tato metoda predikuje s výbornou přesností 98%. U ostatních indexů však predikuje oproti ostatním algoritmům neuspokojivě a nebo nepredikuje vůbec. Z tohoto faktu je možné usuzovat, že pro metodu k-NN je rozhodujícím prediktorem finanční indikátor *variation*. Velmi dobrých výsledků po selekci příznaků dosáhla také bayesovská síť.

Jak z porovnání obou tabulek vyplývá, u většiny algoritmů selekce příznaků značně pomohla zlepšit přesnost predikce. V tabulce však můžeme nalézt i hodnoty, které se díky selekci příznaků zhoršily. To vypovídá o citlivosti některých algoritmů na správný výběr prediktorů, jak bylo uvedeno výše, například u SVM.

Podobným zpracováním dat jako je selekce příznaků je i normalizace dat, která je velmi důležitá pro testovanou dopřednou neuronovou síť. Bez této normalizace dopředná neuronová síť vůbec nebyla schopna predikce, a jako SVM bez selekce příznaků, si pouze „házela mincí“, kterou hodnotu bude predikovat. Výsledkem pak byla přesnost statisticky odpovídajících asi 50%.

Principem normalizace je sjednocení prediktorů o různých velikostech do jednoho rozsahu, nejčastěji o velikosti 0 až 1. Důvodem může být například použití aktivační funkce sigmoid u neuronové sítě. Tato aktivační funkce nabývá hodnot pouze v rozsahu od -1 do 1. Proto je normalizace nutná, pokud vstupní data tomuto rozsahu neodpovídají. Normalizace navíc řeší problém odlehlých hodnot, díky kterým může dojít k významnému ovlivnění výstupu neuronové sítě. V neposlední řadě normalizace také ohraničuje data do zmíněného pevného rozsahu, což je pro spoustu typů úloh naprostou nezbytností.[28]

Dalším typem předzpracování dat je standardizace, která byla použita u hlubokého učení a logistické regrese. U logistické regrese standardizace neměla žádný vliv na přesnost predikce, u hlubokého učení měla ale standardizace zásadní dopad. Pokud byla standardizace vynechána, přesnost predikce hlubokého učení se snížila vždy na asi 60%, což je cirka o třetinu méně, než při aplikované standardizaci.

Standardizace funguje na podobném principu jako normalizace, avšak hlavním rozdílem je, že standardizovaná data mají nulový průměr a rozptyl roven jedné. Rozsah nových hodnot pak odpovídá rozsahu přibližně od -3 do 3.[29]

Celé toto měření přesnosti predikce ostatních algoritmů bylo vyhodnocováno metodou křížové validace. Jedním z jejích nejčastějších použití je právě vyhodnocování přesnosti použitého prediktivního modelu na vstupních datech. Principem je rozdělení vstupních dat na několik podmnožin, přičemž vždy jedna je testovací množinou a ostatní jsou trénovacími množinami. Model je poté natrénován na trénovacích mno-

žinách a jeho přesnost je otestována testovací množinou. Tento proces se opakuje tolikrát, jaký je počet podmnožin, přičemž vždy je zvolena jiná testovací množina. U obou výše uvedených testů přesnosti predikce ostatních algoritmů byla vstupní data rozdělena na 10 podmnožin.[30]

Síť s ozvěnou stavu nebyla do tohoto měření zapojena, protože se jednalo o jiný typ úlohy. Ostatní algoritmy v podstatě klasifikovaly nákupní signály do dvou tříd na základě hodnot indikátorů. Obě varianty sítě s ozvěnou stavu však do tříd neklasifikují, nýbrž předpovídají příští hodnotu indikátoru na základě poslední známé hodnoty indikátoru. Funkce je tedy úplně jiná a proto nebyla síť s ozvěnou stavu porovnávána při měření s ostatními algoritmy. Pravděpodobně by bylo možné obě varianty kompletně přepracovat, aby byly schopny klasifikace stejné úlohy jako ostatní algoritmy. Bylo by však třeba v jazyce Java naprogramovat i normalizaci dat a křížovou validaci, abychom mohli varianty sítě s ozvěnou stavu přímo srovnat na této klasifikační úloze s ostatními algoritmy.

Na základě velmi dobrých výsledků algoritmů fungujících na principu neuronových sítí, je možné předpokládat, že by i síť s ozvěnou stavu dosáhla v takovém typu úlohy uspokojivých výsledků.

V neposlední řadě je nutné říci, že testované ostatní algoritmy nedisponují žádnou zpětnou vazbou nebo pamětí. Díky tomu nejsou schopny tak sofistikované predikce jako síť s ozvěnou stavu.

7 ZÁVĚR

V této práci jsou stručně uvedeny neuronové sítě, je popsán základní princip sítě s ozvěnou stavu (echo state network) a také jsou zmíněny vybrané neuronové sítě, které se podílejí na vzniku sítí s ozvěnou stavu. Následně jsou představeny další algoritmy používané pro analýzu časových řad a v neposlední řadě byly také stručně popsány nástroje, které byly použity v praktické části práce.

Součástí této práce je také vytvořená akcelerovaná varianta sítě s ozvěnou stavu, která vychází ze vzorové implementace této neuronové sítě. Akcelerovaná varianta se oproti vzorové liší způsobem učení, které je přepracováno tak, aby výpočet vah efektivně pracoval na grafickém procesoru.

Bohužel, tato akcelerovaná varianta nesplnila teoretická očekávání v rychlosti učení. Běh této varianty je přibližně dvakrát pomalejší než běh varianty vzorové. Důvodem je dlouhá doba přesunu datových matic z paměti programu na GPU a také nutnost postupného průchodu vstupních dat sítí z důvodu aktivace echo stavů. Výsledná přesnost akcelerované varianty však předčila variantu vzorovou.

Dále byly srovnány ostatní algoritmy, které se používají pro analýzu časových řad. Pro tyto algoritmy byla přichystána klasifikační úloha, kdy na základě finančních indikátorů měly predikovat, zda se jedná o nákup v pozitivních nebo v negativních číslech. Jinými slovy, zda se jedná o vzestup finančního indexu, nebo pokles. Výsledkem této úlohy pak byla přesnost jednotlivých algoritmů. Po porovnání těchto výsledků bylo zjištěno, že nejlepších výsledků dosahují algoritmy fungující na principu neuronových sítí. Dále bylo zjištěno, že některé algoritmy jsou velmi citlivé na volbu prediktorů a také bylo potvrzeno, že předzpracování dat může výrazně zlepšit výsledky následujícího algoritmu.

Síť s ozvěnou stavu nebyla přímo srovnána měřením s ostatními algoritmy, protože funkce ostatních algoritmů je jiná než sítě s ozvěnou stavu. Síť s ozvěnou stavu predikuje následující hodnoty finančních indikátorů z již známých předchozích hodnot těchto indikátorů. Zatímco ostatní algoritmy klasifikují data do dvou tříd na základě hodnot finančních indikátorů. Vzhledem k výsledkům měření přesnosti ostatních algoritmů je ale možné předpokládat, že pokud by byla síť s ozvěnou stavu celkově přeprogramována na klasifikaci, dosáhla by v tomto měření také uspokojivých výsledků, tak jako všechny algoritmy fungující na principu neuronové sítě.

LITERATURA

[1] JIRSÍK, V. *Umělá inteligence.* Materiály k předmětu. Brno: FEKT VUT v Brně, 2016.

[2] VONDRÁK, I. *Neuronové sítě* [online]. 2009, [cit. 19. 11. 2017]. Dostupné z URL: <http://vondrak.cs.vsb.cz/download/Neuronove_site.pdf>.

[3] HAZAN, H.; MANEVITZ, L.; FRID, A. *Temporal Pattern Recognition via Temporal Networks of Temporal Neurons* [online]. 2012, [cit. 20. 11. 2017]. Dostupné z URL: <http://web.archive.org/web/20170724025959/http://hananel.hazan.org.il/files/Publication/2012/Temporal%20Pattern%20Recognition%20via%20Temporal%20Networks%20of%20%20Temporal%20Neurons.pdf>.

[4] *Echo state network* [online]. 2007, [cit. 21. 11. 2017]. Dostupné z URL: <http://www.scholarpedia.org/article/Echo_state_network>.

[5] KOTRAS, M. *Lokálna predikcia počasia s využitím Echo-State sietí* [online]. 2011, [cit. 21. 11. 2017]. Dostupné z URL: <http://neuron.tuke.sk/jaksa/theses/2011/Kotras-Jaksa-MSc11-thesis.pdf>.

[6] *Maticové rozklady - ČVUT - Kapitola 4.* [online]. 2009, [cit. 21. 11. 2017]. Dostupné z URL: <http://neuron.tuke.sk/jaksa/theses/2011/Kotras-Jaksa-MSc11-thesis.pdf>.

[7] *SVD and the Pseudoinverse.* [online]. [cit. 22. 11. 2017]. Dostupné z URL: <http://uspas.fnal.gov/materials/05UCB/6_SVD.pdf>.

[8] PROCHÁZKA, J. *Prediktivní analýza v prostředí internetu.* [online]. 2012, [cit. 16. 4. 2018]. Dostupné z URL: <https://is.muni.cz/th/oqem2/PA_final.pdf>.

[9] FREEDMAN, D. *Statistical Models: Theory and Practice.* Cambridge University Press. str. 128. 2009, [cit. 17. 4. 2018].

[10] TREJBAL, P. *Jak na rozhodovací stromy* [online]. 2014, [cit. 18. 4. 2018]. Dostupné z URL: <https://www.optimics.cz/jak-na-rozhodovaci-stromy/>.

[11] HO, T. K. *Random Decision Forests* [online]. 1995, [cit. 22. 4. 2018]. Dostupné z URL: <http://ect.bell-labs.com/who/tkh/publications/papers/odt.pdf>.

[12] DOLNÍČEK, P. *Trénovatelná segmentace obrazu s použitím hlubokého učení.* Diplomová práce. Brno: FEKT VUT v Brně, 2017.

[13] STEINWART, I.; CHRISTMANN, A. *Support Vector Machines.* Springer. New York, 2008.

[14] *Multilayer Perceptron.* [online]. [cit. 21. 05. 2018]. Dostupné z URL: <https://deeplearning4j.org/multilayerperceptron>.

[15] HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, j. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer. New York, 2009.

[16] RENNIE, J.; SHIH, L.; TEEVAN, J.; KARGER, D. *Tackling the Poor Assumptions of Naive Bayes Text Classifiers* [online]. 2003, [cit. 25. 04. 2017]. Dostupné z URL: <http://people.csail.mit.edu/jrennie/papers/icml03-nb.pdf>.

[17] BURGET, R. *Teoretická informatika.* Skripta k předmětu. Brno: FEKT VUT v Brně, 2013.

[18] *What is Maven* [online]. [cit. 26. 11. 2017]. Dostupné z URL: <https://maven.apache.org/what-is-maven.html>.

[19] *Apache Maven Project* [online]. [cit. 27. 11. 2017]. Dostupné z URL: <https://maven.apache.org/index.html>.

[20] *Deeplearning4j Development Team. Deeplearning4j: Open-source distributed deep learning for the JVM, Apache Software Foundation License 2.0.* [online]. [cit. 2. 12. 2017]. Dostupné z URL: <http://deeplearning4j.org>.

[21] *ND4J Development Team. ND4J: N-dimensional arrays and scientific computing for the JVM, Apache Software Foundation License 2.0* [online]. [cit. 24. 11. 2017]. Dostupné z URL: <https://nd4j.org/>.

[22] *Nd4j – Numpy for the JVM* [online]. 2016, [cit. 30. 11. 2017]. Dostupné z URL: <https://www.beyondthelines.net/machine-learning/nd4j-numpy-for-the-jvm/>.

[23] *Ta4j - Technical Analysis for Java!* [online]. [cit. 17. 4. 2018]. Dostupné z URL: <https://ta4j.github.io/ta4j-wiki/>.

[24] *RapidMiner from Rapid-I at CeBIT 2010* [online]. 2010, [cit. 18. 4. 2018]. Dostupné z URL: <http://www.data-mining-blog.com/cloud-mining/rapidminer-cebit-2010/>.

[25] HOFMANN, M., KLINKENBERG, R. *"RapidMiner: Data Mining Use Cases and Business Analytics Applications.* CRC Press. 2013, [cit. 18. 4. 2018].

[26] *Apache Maven Project - Packaging* [online]. [cit. 28. 11. 2017]. Dostupné z URL: <https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html#Packaging>.

[27] HOLČÍK, J. *Volba a výběr příznaků* [online]. [cit. 14. 05. 2018]. Dostupné z URL: <http://www.iba.muni.cz/esf/res/file/bimat-prednasky/analyza-a-klasifikace-dat/AKD-08.pdf>.

[28] *Normalize - RapidMiner Documentation* [online]. [cit. 14. 05. 2018]. Dostupné z URL: <https://docs.rapidminer.com/latest/studio/operators/cleansing/normalization/normalize.html>.

[29] *Normalization vs. Standardization* [online]. [cit. 14. 05. 2018]. Dostupné z URL: <http://www.statisticshowto.com/normalized/>.

[30] VLHA, M. *Odhad výkonnosti diskových polí s využitím prediktivní analytiky.* Diplomová práce. Brno: FEKT VUT v Brně, 2017.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
CSV	Comma-Separated Values
DL4J	Deep Learning For Java
ESN	Echo State Network
FFNN	Feed Forward Neural Network
GPU	Graphic Processing Unit
k-NN	k-Nearest Neighbors
LSM	Liquid State Machine
MACD	Moving Average Convergence/Divergence
MIT	Massachusetts Institute of Technology
MLP	Multilayer Perceptron
ND4J	N-Dimensional Arrays For Java
OHLC	Open, High, Low, Close
RNN	Recurrent Neural Network
RSI	Relative Strength Index
SVD	Single Value Decomposition
SVM	Suppor Vector Machines
TA4J	Technical Analysis For Java
VUT	Vysoké Učení Technické
YALE	Yet Another Learning Environment

SEZNAM PŘÍLOH

A OBSAH PŘILOŽENÉHO CD

- **xpospi81.pdf:** elektronická verze práce.
- **DP17xpospi81_esn:** java projekt se zdrojovými kódy akcelerované GPU varianty a s vytvořenými datovými soubory reálných finančních indexů.
- **EchoStateNetwork:** java projekt se zdrojovými kódy vzorové CPU varianty.
- **Ta4j:** java projekt se zdrojovými kódy, jejichž prostřednictvím byly vytvořeny datové soubory finančních indexů.