Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra ekonomie

Robotické investování

Diplomová práce

Autor práce: Bc. Štěpán Kameník
Studijní obor: Aplikovaná Informatika

Vedoucí práce: Ing. Jan Mačí, Ph.D.

Hradec Králové 2022

Prohlášení

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury

.....

Štěpán Kameník

27. dubna 2022

Poděkování

Děkuji Ing. Janu Mačí Ph.D. za metodické vedení, osobní přístup a hlavně trpělivost při tvorbě této práce. Děkuji také Nicole Trompischové za jazykovou korekci a podporu.

Anotace

Tato diplomová práce je zaměřena na popis aktuálně nejvíce používaných metod pro predikce finančních časových řad. Tyto modely - přesněji modely ARIMA, ANN, LSTM, Bi-LSTM, komplexní (obsahující dvě sítě LSTM / Bi-LSTM) a hybridní používající více zdrojů dat - jsou následně v práci implementovány a podrobeny testování klasickými metrikami. Všechny použité modely jsou testovány na sedmi datových sadách různých finančních instrumentů (Apple Inc., JPMorgan Chase & Co., Amazon.com, Inc., ETF S&P500, All-World ETF, Bitcoin, Ethereum). Každá tato testovací sada obsahuje množinu denních, hodinových, půlhodinových i minutových dat. Tímto způsobem dojde k testování variability navržených modelů a jejich závislosti na přesných vlastnostech datových řad. Následně jsou predikce modelů použity k virtuálnímu obchodování na časovém intervalu, na kterém model nebyl trénován. Tímto způsobem lze vyhodnotit použitelnost modelu pro reálné obchodování. V rámci praktické části práce bylo dosaženo slibných výsledků u několika modelů neuronových sítí (konkrétně LSTM a Bi-LSTM) i klasického přístupu autoregresního modelu. Naopak celkem překvapivě velmi špatných výsledků dosáhly modely, které měly více informací mimo pouhé ceny instrumentu k predikci časové řady.

Klíčová slova

akcie, kryptoměny, časová řada, strojové učení, ARIMA, RNN, LSTM, Bi-LSTM

Anotation

Title: Robotic Investing

This diploma thesis is focused on the description of the currently most used methods for the prediction of financial time series. These models - more precisely, ARIMA, ANN, LSTM, Bi-LSTM, complex (containing two LSTM / Bi-LSTM networks) and hybrid models using multiple data sources - are then implemented and tested by classical metrics. All models used in this work are tested on seven datasets of various financial instruments (Apple Inc., JPMorgan Chase & Co., Amazon.com, Inc., ETF S&P500, All-World ETF, Bitcoin, Ethereum). Each of these test sets contains a set of daily, hourly, half-hour, and minute data. In this way, the variability of the proposed models and their dependence on the exact properties of the data series will be tested. Subsequently, model predictions are later used for virtual trading on a time interval on which the model was not trained. In this way, the applicability of the model for real trading can be evaluated. Within the practical part of the work, promising results were achieved with several models of neural networks (specifically LSTM and Bi-LSTM) and the classical approach of the autoregressive model. On the contrary, models that had more information than just the prices of the time series prediction instrument achieved quite surprisingly very poor results.

Keywords

stocks, cryptocurrencies, time-series, machine learning, ARIMA, RNN, LSTM, Bi-LSTM

Obsah

Seznam obrázků

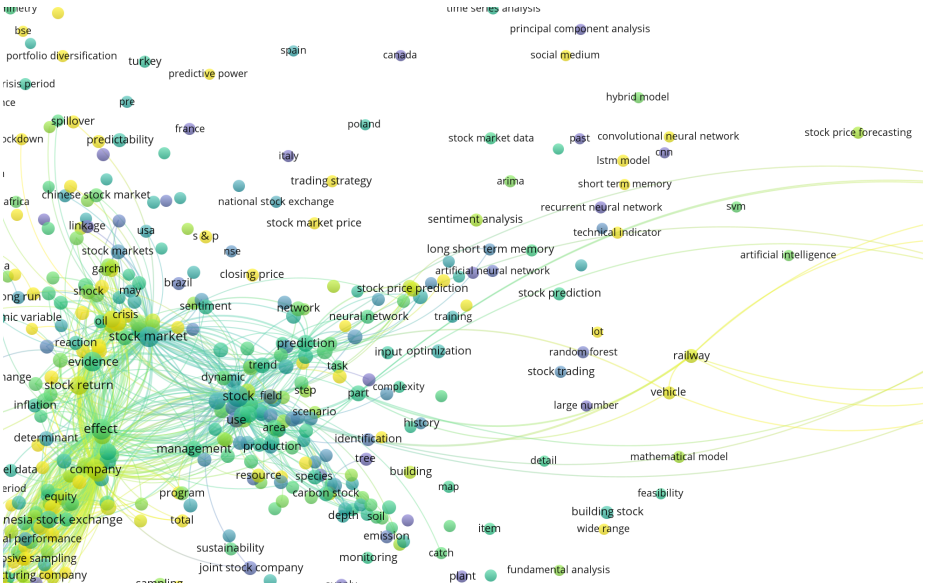
Seznam tabulek

1 Úvod

Ještě před pár lety, nebyly pojmy jako investice, akcie nebo podílové fondy ničím, co by běžný občan chtěl řešit, protože se třeba i těchto instrumentů bál, ať již z historického nebo osobního důvodu. Nejspíše to bylo dáno stabilitou trhu před ekonomickou krizí v roce 2008 či nízkou inflací, která na českém trhu byla ještě v prvním „covidovém“ roce 2020. V posledních letech však oba tyto faktory postupně mizí. Současně s nákazou Covid-19, jednáním centrálních bank (například FEDu) i vlivem zvyšování úspor obyvatel (kvůli nejistotě nebo omezení aktivit), se vytvořila nebývale velká inflace. Inflace má jednu hlavní vlastnost - ztrátu kupní síly oběživa.

Díky těmto okolnostem, se investice a možnosti zhodnocení peněz dostaly do povědomí větší míry obyvatel, avšak s investicemi je nutně spojeno riziko. Některé investice jsou považovány za velmi bezpečné, například dluhopisy republiky. Jejich výnosnost je pevně dána při koupi konkrétní emise a jako riziko je zde pouze nemožnost státu splácet své závazky, tedy (dle hodnocení stability ČR) poměrně malé. Na druhou stranu je však nutné dodat, že kromě výjimek tyto nízko-rizikové investice svým výnosem inflaci nepřevýší[28], obzvláště pokud je inflace vyšší jako například v roce 2022.

Současně se do popředí velké skupiny oborů dostávají, nebo v nedávné době již dostaly, pokročilé techniky strojového učení a neuronových sítí[29]. Za pomoci velmi intuitivních knihoven, se tyto techniky dají sestrojit v průběhu několika minut a implementátor nepotřebuje mnohdy ani znalosti, jak přesně implementovaný algoritmus funguje. I díky tomu se tyto modely začínají používat v mnohých oborech a je překvapivé, že ještě v nedávné době (do r. 2018), dle analýz publikací (VOSViewer) obsahujících klíčové slovo "stock"(akcie), nebyly začleněny do výzkumu týkajícího se problematiky předpovědí finančních řad. Je též možné, že se tyto techniky dříve používaly v soukromé sféře, avšak výzkumné publikace se o nich příliš nezmiňují jako o hlavním tématu, alespoň dle databáze CrossRef. Trend se ovšem minimálně po roce 2018 výrazně změnil.



Obrázek 1: Výřez analýzy publikací o tématu akcií publikovaných po roce 2018 [vlastní zpracování programem VOSviewer]

Právě proto bylo téma této práce uzpůsobeno pro popis aktuální situace v používání různých technik strojového učení na finančních časových řadách, s hlavním cílem sestrojit takový model strojového učení, který pro vybraný instrument překoná svou ziskovostí pasivní investování.

V první kapitole této práce je stručně vysvětleno prostředí, ve kterém se investoři pohybují a obecné principy investování. Též jsou v této kapitole zavedeny různé typy analýz, které se při investování používají k relativnímu ohodnocení prodejního instrumentu.

Druhá kapitola se věnuje převážně popisu časových řad a představuje statistické modely, které jsou pro analýzu a predikce používány.

Třetí kapitola popisuje hlavní principy strojového učení, jeho typy a obecné postupy, kterými se metody sestrojují. Na rozdíl od kapitoly analýzy časových řad, tato kapitola nepřináší přímý model využitelný pro časové řady, avšak slouží jako základní můstek k představení neuronových sítí, které jsou pilířem ostatních použitých modelů.

Čtvrtá, nejobjemnější kapitola, pojednává o obecném principu a fungování neuronu. Dále objasňuje hlavní principy propojování neuronů do neuronových sítí, problémy s tím spojené a různé techniky, které s sebou tyto sítě přinášejí - například pro omezení nechtěných stavů, či vylepšení, která se za dobu používání těchto sítí vyvinula. Mimo popisu rekurentních sítí, které se zabývají právě časovými řadami a posloupnostmi, je představen i druhý velký typ neuronových sítí a tím jsou konvoluční. Ač v této práci nejsou přímo použity, techniky těchto sítí jsou velmi lehce aplikovatelné i na rekurentní a klasické neuronové sítě[30], proto znalost tohoto oboru může pomoci v dalším rozšiřování implementovaného řešení, či k lepšímu pochopení některých použitých technik.

Další, pátá kapitola obsahuje samotnou praktickou část práce, vytvoření modelů představených v části praktické, zavedení testovacích technik, testování a následné vylepšování těchto modelů. Zároveň se tato kapitola snaží představit vhled do iterativního procesu, jakým je vývoj neuronových sítí.

I přesto, že část výsledků je již prezentována v praktické části, poslední kapitolou je právě shrnutí výsledků. Kapitola se věnuje především porovnání jednotlivých modelů na různých typech dat. Popisuje též možnost reprodukce těchto výsledků pomocí připravených implementovaných funkcí.

2 Cíl práce

Cílem této práce je vyvinout model strojového učení, pomocí kterého by bylo možné roboticky obchodovat. Dále pak za pomoci srovnání určit vhodnost a ziskovost obchodní strategie používající vytvořený model oproti ne-automatizovaných metodám investování, případně pasivnímu investování.

Pro kompletnost díla je nutné definovat jako dílčí cíle: seznámení čtenáře se základními metodami finančního trhu z pohledu malého investora, seznámení se strojovým učením a následně i se samotnými neuronovými sítěmi, které by měly tvořit hlavní část této práce.

Součástí praktické části je následné sestrojení několika typů modelů, které by mohly mít vlastnost predikce těchto řad, současně s alespoň jednou statistickou technikou predikce časové řady nespadající do oblasti strojového učení.

3 Metodika zpracování

V rámci implementace řešení, by se měla u jednotlivých metod strojového učení prokázat vlastnost rozpoznat vzory v časové řadě, resp. na základě minulých hodnot správně predikovat hodnotu budoucí. Dále by se mělo prokázat, která z těchto metod predikuje hodnoty nejblíže hodnotám reálným a zda jsou při předpovědi (či modelovém obchodování) úspěšnější než strategie nepoužívající strojové učení. Úspěšnost by se dále měla zlepšit kombinací dvou výše zmíněných metod, tedy specifické metody strojového učení a metody klasické analýzy.

3.1 Prokazované hypotézy

Hypotéza 1 (H1): *Použitím obchodování na doporučení modelů strojového učení vznikne při testování na historických datech zaručený vyšší zisk, než který by plynul z nákupu instrumentu na začátku testovaného období a prodeje na konci.*

Hypotéza 2 (H2): *Kombinací vybraných metod strojového učení a technické analýzy se prokazatelně zlepší profitabilita celkového modelu.*

Hypotéza 3 (H3): *Úspěšnost predikce cen bude odlišná při předpovědi na cenových datech různých instrumentů.*

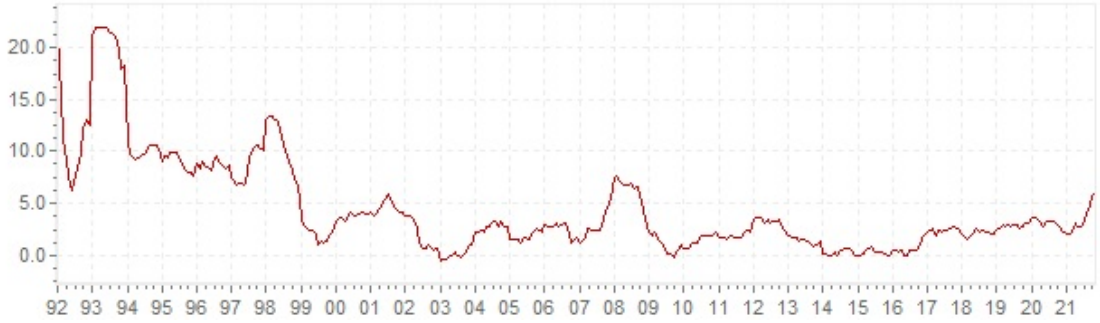
3.2 Způsoby ověření

Ověření výše popsaných hypotéz bude prováděno na historických datech tří akcií společností (Apple Inc., JPMorgan Chase & Co., Amazon.com, Inc.) jakožto reprezentantů akciových titulů. Dále na ETF S&P500, All-World ETF a poslední data budou pocházet z kryptoměn Bitcoin a Ethereum. Toto rozložení datových sad bylo vybráno, aby byla odhalena případná vyšší úspěšnost na některém z instrumentů. Pro každou z těchto datových sad budou použity všechny modely k prokázání jejich úspěšnosti na specifických datech.

4 Investiční prostředí

Jedním z prvních pravidel, které se začátečník naučí, pokud začne číst a zajímat se o střednědobé a dlouhodobé investování, je „Trh časovat nelze“[31]. Toto pravidlo vychází hned z několika jevů. Prvním z nich je, že nikdo dopředu nezná budoucnost. Může zítra přijít nová pandemie, objevit se meteorit letící k zemi či tornádo v USA, které zasáhne továrnu Tesly. Ať již to jsou takto náhodné věci, či nenáhodné jako například výsledky prodejů, není možné predikovat, jak moc na danou událost akciový trh zareaguje.

Nehledě však na faktory ovlivňující cenu, existují i studie, které uvádí, že když se vybere jakékoliv dvacetileté období historie trhu a investor na začátku období zainvestuje, pak v žádném z těchto období nebude ve ztrátě[32]. Samozřejmě, že toto pravidlo nemá žádnou technickou oporu a tedy kdykoliv může přestat platit. Na druhou stranu, bez investování do jakéhokoliv instrumentu, hotovost snižuje inflací svou hodnotu. Pokud se jedná například o posledních dvacet let, pak průměrná roční inflace od roku 2001 do 2021 činila 2.19% a z ušetřených sta tisíc by po dvaceti letech zbylo reálně pod pětašedesát tisíc korun českých.



Obrázek 2: Historie inflace ČR [1]

4.1 Základní pojmy

Obecně tedy lze říci, že každý člověk, který má v jakékoliv podobě větší částku peněz, či nevyužitou část měsíčních příjmů, by měl nad investováním přemýšlet. Mimo investování je též možné využít i jiné způsoby, které udrží prosté spoření. Jedná se o spořicí účty nebo (většinou lépe hodnocené) termínované vklady[28]. Hodnota úrokového výnosu těchto služeb obvykle nepřekonává inflaci, tedy z dlouhodobého hlediska je tento vklad úročením nevýdělečný. Pokud je úrok menší než aktuální míra inflace, hodnota vkladu se dokonce snižuje.

4.1.1 Alternativní možnosti úschovy peněz

Na druhou stranu však tyto nástroje nemají praktické riziko ztráty peněz, tedy pokud chce investor držet hotovost po kratší dobu, například i po tři roky, je spořicí účet (či termínovaný vklad) z tohoto důvodu dobrou volbou[33]. Pokud stejnou částku vloží do konkrétní akcie (místo na spořicí účet) a ta zanedlouho poté z nějakého důvodu poklesne, během tří let se ztráta nemusí zcela vykompenzovat. To ani v případě, že byla firma investorem správně zanalyzována a jednalo se pouze o náhodný pokles[34]. Z tohoto důvodu se investování do akciového trhu nedoporučuje pro krátké horizonty. Kromě klasických spořicích účtů existují

i další podobné produkty jako je stavební spoření a penzijní spoření. Výjimku tvoří část peněz určená pro nenadálé situace - tato část tvoří například 15 % hodnoty portfolia a slouží pro případ nutných neplánovaných výdajů (viz níže). V tomto případě je spořicí účet též vhodným nástrojem úschovy[2].

4.1.2 Možnosti investování

Kromě vkladu prostředků do již výše zmíněných akcií, je možné investovat i do jiných nástrojů, jak **reálných**, tak **finančních**.

Investování do reálných instrumentů

Pod názvem investování do reálných instrumentů se skrývá velké množství možností v klasickém pojetí, od koupě nemovitosti přes zlato, plyn, ropu až po rýži, nebo třeba čaj. V historii byl tento trh čistě praktický a pro jednotlivé zboží dovoloval tvorbu ceny dle poptávky, nyní se však tyto instrumenty používají spíše jako možnost spekulace na pokles, resp. růst ceny daného zboží a to přes nástroj zvaný CFD (contract for difference), či opce[35].

Investování do finančních instrumentů

Druhou, obsáhlejší skupinou jsou finanční instrumenty. Do této skupiny patří většina tzv. cenných papírů, od již zmíněných akcií, dluhopisů, fondů, podílových listů přes forex směny, swapy až po opce. Na rozdíl od předešlé skupiny, tyto instrumenty obvykle nemají fyzický podklad (rýži), ale představují majetkovou hodnotu například podílu ve firmě.

4.1.3 Příklady investování

Do těchto nástrojů investuje jednotlivec i ve chvíli, kdy si zřizuje penzijní spoření, které nabízí jednu z největších mír návratnosti právě proto, že poskytnuté prostředky banka reinvestuje do několika z těchto nástrojů [2] a kromě odebraných poplatků distribuuje investorům výnos dle částky jejich investice.

Další velmi rozšířenou formou je investice pomocí podílových fondů. Podílové fondy fungují na obdobném principu, kde si investor zvolí konkrétní fond a zainvestuje libovolnou částku. Prostředky od všech klientů fond vezme a zainvestuje do předem daných instrumentů, které volí manažeři fondu[35]. Úkolem fondu je splnit účel fondu, kterým je, na základě přesně stanovených rizik, generovat zisk. Fond si ze svěřených peněz samozřejmě odebírá poplatky jak roční, tak i na vyplaceném zisku.

Na rozdíl od výše popsaných metod je investování bez prostředníka časově velmi náročné, investor v tomto případě vlastně přebírá roli manažera svého vlastního fondu a vybírá typ finančního instrumentu, jeho zastoupení v portfoliu i možnou diverzifikaci mezi jednotlivými investicemi.

4.1.4 Diverzifikace

Portfolio se tzv. diverzifikuje, aby umělo vzdorovat přirozeným výkyvům v hodnotách akcií. Diverzifikace je proces, kterým investor portfolio analyzuje a předpokládá korelace mezi jednotlivými instrumenty vzhledem k jejich zastoupení. Jinými slovy se investor snaží odolat případnému pádu jednoho segmentu trhu a nemít proto více firem, či komodit se stejným nebo podobným segmentem vývoje.

Obrázek 3: Styl diverzifikace [2]

Obecně se diverzifikace doporučuje na více úrovních, jak je naznačeno na obrázku 3. První diverzifikací je diverzifikace prostředků, tedy například rovnoměrné rozdělení do třetin - akcie, reality a spořicí účty. Tento nejvyšší stupeň diverzifikace brání proti ztrátám při událostech jako je finanční krize, kdy povětšinou akcie velmi razantně klesnou, zatímco nemovitosti naopak v tu chvíli často svou hodnotou stoupají a peníze na spořicích účtech pokrývají možné zvýšené náklady, či nečekané výdaje spojené s krizí; celkový majetek je tak ochráněn proti větší ztrátě. Nižší diverzifikací je diverzifikace jednotlivých složek - například portfolia akcií. Zde je již nutné počítat s různými faktory jako je lokalita daného instrumentu a firmy, také odvětví působení, nebo korelace s ostatními subjekty[33].

Během let se způsob diverzifikování portfolia měnil, jednou z aktuálně nejpoužívanějších metod je diverzifikace pomocí modelu PMPT (Post-modern portfolio theory). Diverzifikace je v těchto modelech běžně kalkulována jako poměr volatility a korelace s trhem. [36]

Jelikož se jedná o komplexní způsob, který závisí pouze na specifickém nastavení portfolia, resp. na upřesnění, jak moc rizikové může být, jaká je jeho nutná stabilita a další proměnné, nebude součástí této práce. V rámci praktické části bude tedy samotné složení portfolia abstrahováno.

4.1.5 Sestavení portfolia

Právě uspořádání portfolia do podoby, aby každý instrument splňoval podmínky rizika, diverzifikace a měl pro investora význam, je většinou velmi obtížné. Při budování takového portfolia, je každý krok závislý jak na technickém zkoumání jednotlivých instrumentů (finančních výkazů, konkurence, ...), tak na samotných cenách prvků, které samy o sobě vypovídají o jevech ve firmě, nástroji, či v jejich sektoru. Mít alespoň nějakou možnost predikce této ceny by mohlo zajistit neocenitelnou výhodu v boji „proti trhu“.

4.2 Fundamentální analýza

Pokud investor uvažuje o koupi některé z akcií, je pro něho vždy velmi důležité porozumět danému sektoru a analyzovat, zda sektor samotný odpovídá požadavkům investora (je růstový, má budoucnost nebo je stálý a vyplácí velké dividendy ap.). Analýza sektoru je většinou velmi subjektivní až sentimentální, protože obvykle neexistují přímé hodnoty, dle kterých by se investor rozhodoval o hodnotě sektoru. Musí tak, například po přečtení aktuálních informací o sektoru, dojít k nějakému vlastnímu subjektivnímu názoru na sektor[37]. Pokud se tedy investor do sektoru rozhodne investovat, přichází jedna z nejtěžších částí

investování. Tou je analýza samotné společnosti a nalezení tzv. „vnitřní hodnoty“ akcie, která se také nazývá investiční hodnota společnosti.

4.2.1 Zhodnocení akcie

Investiční hodnota je definována IVS jako „Hodnota majetku pro investora, nebo třídu investorů pro stanovené investiční cíle.“ [1]. Tento subjektivní pojem spojuje specifický majetek se specifickým investorem, skupinou investorů nebo jednotou s určitými investičními cíli nebo kritérii. Investiční hodnota může být vyšší, nebo nižší než tržní hodnota tohoto majetkového aktiva. Pro výpočet investiční hodnoty se využívá tzv. fundamentální analýza, která se dle výše popsané definice snaží odhadnout „spravedlivou“ cenu akcie firmy, za kterou by se měla prodávat, pokud by byl kapitálový trh naprosto objektivní.

Typy analýz: cenu akcie však neovlivňuje pouze firma samotná, ale, jak již je psáno výše, i sektor, globální dění, dokonce i politické mechanismy na úrovni států. Z toho důvodu, investiční hodnota obvykle neodpovídá reálné tržní hodnotě firmy[35]. Pro obraz a analýzu reálné ceny akcie se tak využívá technická analýza, sestávající se z analýzy časové řady grafu akcie a dostupných nástrojů agregujících tyto hodnoty (například indikátory, oscilátory).

Po zanalyzování globální situace a situace ve specifickém odvětví, je analýza samotné firmy nejdůležitější částí při rozhodování pro, či proti koupi. Je tomu tak, protože investorovi dokáže naznačit budoucí možnost prosperity firmy a mimo jiné i právě její investiční hodnotu. Jako základ pro stanovení hodnoty se většinou používá metoda DCF (discounted cash flow) [1].

4.2.2 Diskontované peněžní toky

V modelu diskontovaných peněžních toků neboli DCF se rozlišují tři základní techniky výpočtu:

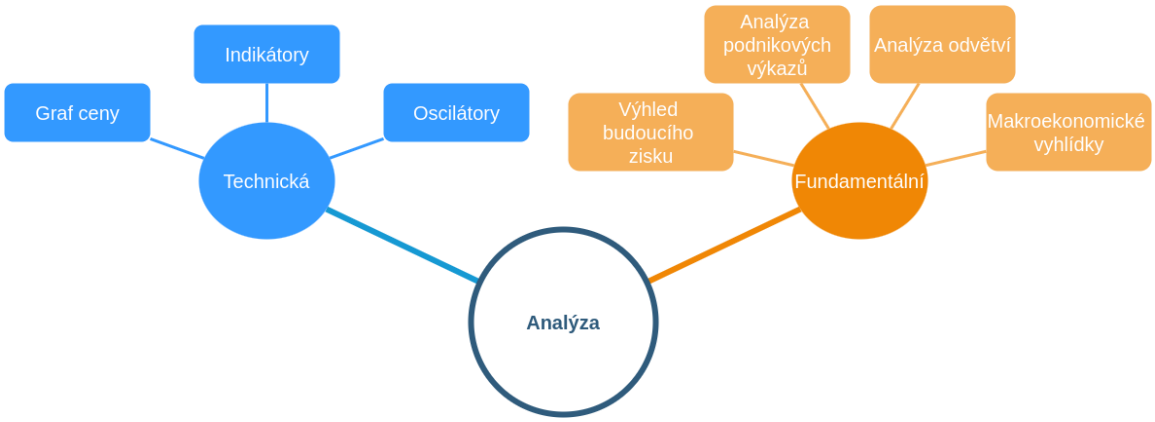
- entity – podnik jako celek
- equity – vlastní kapitál
- APV (adjusted present value) – upravená současná hodnota

Účelem této metody je zjištění současné hodnoty čistého majetku či hodnoty kapitálu konkrétní firmy. Čistým majetkem tohoto objektu se rozumí obchodní majetek po odečtení závazků.

Z popisu vyplývá, že pomocí DCF lze odhadnout hodnotu firmy a následně vydělením počtu akcií „spravedlivou“ cenu. Investor, pomocí této ceny zjistí případné nadhodnocení společnosti na trhu a může tak porovnat ceny akcií a hodnot DCF konkurenčních společností. Bohužel, tato ani příbuzná metoda nemá prakticky žádnou výhodu při sledování krátkodobého horizontu (do jednoho kvartálu) [1], z důvodu aktualizace informací o firmě právě pouze jednou za daný kvartál.

4.3 Technická analýza

Technická analýza je svým způsobem opakem fundamentální, protože většinou zanedbává jakékoliv bližší informace o firmě či sektoru a řídí se čistě pouze tržní hodnotou akcií firmy (či instrumentu). V této analýze se investoři pokouší předpovídat budoucí vývoj ceny akcie,



Obrázek 4: Typy analýzy (inspirováno: [1])

vývoj trendu, či jeho úplné otočení. Pro tuto předpověď se používají vyvinuté nástroje, které se označují jak indikátory (ukazatele) a upravují data pomocí vzorců [34]. Indikátory používají deskriptivní techniky časových řad jako jsou například klouzavé průměry, index RS, ap. Tedy spoléhají na předpoklad, že historie má tendenci se opakovat.

Ukazatele se řadí do několika skupin:

- trendové
- cenové a oscilátory
- objemové
- ukazatele sentimentu

4.3.1 Trendové indikátory

Trendové ukazatele představují jednu z nejvyužívanějších skupin indikátorů, i přes technické nedostatky, jako například to, že mají tzv. lagging charakter, tedy, že reagují na změnu ceny až opožděně a nemají žádnou předpovídající hodnotu. Jejich výhoda je však v možnosti agregace hodnot, či charakteristik z grafu do jednodušších struktur. Lze pomocí nich například identifikovat trendy trhu, nebo změny trendu [34]. Další nevýhodou těchto ukazatelů je právě jejich využití trendu. Pokud trh nemá trendový vzestup, nebo sestup („cena jde do strany“), pak identifikátory obvykle nemají žádnou vypovídající hodnotu.

Klouzavý průměr (Moving average/MA) je jedním z nejpoužívanějších ukazatelů mezi investory a tradery, vypočítává průměrnou hodnotu za určitý časový úsek. Většinou se jedná o průměr mezi lichým počtem období (například padesát dnů - MA-50). Tento ukazatel dokáže nejen prozradit celkový trend ceny za poslední časový úsek, ale v případě protnutí grafu zaznamená i potencionální změnu trendu.

V praxi se používají hlavně tři implementace tohoto MA:

- jednoduchý - jednoduchý aritmetický průměr ceny v obdobích
- vážený - vážený aritmetický průměr
- exponenciální - pro období se přidělí exponenciální násobek

Často jsou pro analýzu použity dva indikátory MA s různým počtem agregovaných období, kdy menší reprezentuje krátkodobý trend kurzu a větší naopak trend dlouhodobý[38].

Klouzavý průměr konvergence a divergence: další z velmi používaných indikátorů je tzv. MACD - Moving Average Convergence Divergence (volně přeloženo: „Klouzavý průměr konvergence a divergence“). Často se tento ukazatel řadí do skupiny indikátorů klouzavého průměru, hlavně proto, že se obvykle jedná o rozdíl mezi intervalem šestadvaceti a dvanácti dnů exponenciálního klouzavého průměru.

$$MACD_t(12, 26) = E_t(12) - E_t(26) \tag{1}$$

Dále se tento indikátor kombinuje s exponenciálním klouzavým průměrem devíti dnů, který se nazývá trigger (neboli signální křivka). Pokud hodnoty MACD protnou trigger jedná se o signál k nákupu, či prodeji. Dále se MACD používá i k indikaci výrazné převahy poptávky, nebo nabídky, jestliže hodnoty indikátoru rapidně vzrostou, poklesnou[38].

Obrázek 5: MACD indikátor při protnutí signální křivky (vlastní zpracování - [3])

4.3.2 Cenové indikátory a oscilátory

Na rozdíl od trendových, jsou cenové ukazatele využívány hlavně mimo trendové období, kdy se cena pohybuje typicky v určitých hranicích a osciluje mezi nimi. Tyto indikátory, jako jedny z mála, dokáží poskytnout signál dříve, než se zobrazí v kurzu ceny. Tedy například předpoví obrat k druhé hranici oscilace před samotným otočením. Hlavními reprezentanty této skupiny jsou *Aroon*, *RSI* nebo *Stochastic indikátor*[38].

Momentum, které patří do kategorie cenových indikátorů je jedním z nejjednodušších indikátorů, vyjadřuje pouze rozdíl hodnot mezi dvěma kroky (většinou po sobě jdoucími). Výpočet je tedy pouze:

$$momentum = close_{today} - close_{N\_days\_ago} \tag{2}$$

4.3.3 Objemové indikátory

Oproti tomu objemové indikátory zavádí do výpočtu i objem uzavřených obchodů. S myšlenkou zapojení tohoto prvku přišel již Charles Dow, který usoudil, že objem potvrzuje

trend a tedy pokud existuje trend a objem obchodu je velký, trend je silný. Naopak, pokud trend existuje, ale objem obchodů je malý, je možné, že se trend brzy změní [34]. Příklady těchto indikátorů můžou být právě *Price and Volume trend* ukazatel, či *volume rate of change*, *balance volume* a další.

Bollingerova pásma, která budou využita i v praktické části práce, si zde zaslouží větší zmínku. Metoda nazvaná po svém tvůrci - Johnu Bollingerovi, tvoří pomocí dvou křivek pásmo zobrazující volatilitu trhu s relativní cenovou úrovní. V praxi se tato metoda kombinuje s klouzavým průměrem (většinou dvacetidenním) a již zmíněné dvě křivky znázorňují hranici, ke které se hodnota instrumentu přibližuje a následně klesá, resp. vzrůstá ke křivce druhé. Bollingerovy křivky tedy poskytují obraz rozmezí, kde se cena instrumentu bude pohybovat, pokud se její chování rapidně nezmění.

Obrázek 6: Bollingerova pásma (vlastní zpracování - [3])

Výpočet těchto pásem je možné provést z klouzavého průměru (např. dvaceti dnů) a následně od průměru odečíst, resp. přičíst standardní odchylku ceny * 2[38]:

$$B_{upper,lower} = MA_n \pm 2 * \sqrt{\frac{\sum X_j - MA_n}{N}} \tag{3}$$

4.3.4 Indikátory sentimentu

Poslední kategorií obsahující převážně komplikovanější ukazatele je skupina indikátorů sentimentu. Tyto indikátory se snaží odhadnout náladu na burze a zachytit emoce, které investoři v danou chvíli pro konkrétní cenu mají. Důvod těchto indikátorů je jednoduchý, do ceny se vždy promítají emoce investorů, proto vznikly dvě skupiny těchto ukazatelů:

- cyklické - identifikují chování zkušených investorů a traderů, například na burze opcí, kde se předpokládá úspěšnost obchodů i přes menší celkový zisk, díky například velmi včasně zpracovaným informacím z trhu. Investor sledující tyto ukazatele by se jimi měl povětšinou řídit a předpokládat, že ti „zkušení“ mají lepší informace.

- anti-cyklické - tyto indikátory sledují chování všech investorů, tedy i méně zkušených, kde většina investorů jedná se značným zpožděním a jsou tedy méně úspěšní. Investor, sledující tyto ukazatele, by měl postupovat přesně naopak výsledkům těchto ukazatelů.

Příklady cyklických indikátorů mohou být *Stock Mutual Fund Cash/Assets Ratio*, *Barron's Confidence*, naopak reprezentanty anti-cyklických indikátorů jsou *Short Interest Ratio*, *Put/Call Ratio*, *Public Short Ratio*[39].

5 Analýza časových řad

V této kapitole jsou popsány postupy klasických statistických metod, které mají za účel objasnit a modelovat danou časovou řadu. Tyto metody se zakládají zejména na vysvětlení jednotlivých procesů, které časovou řadu tvoří.

5.1 Stochastický proces

Stochastický proces popisuje průběh v čase a obsahuje jednu nebo více náhodných složek. Oproti procesu deterministickému, se stochastický nedá vyjádřit pomocí matematické funkce a díky působnosti náhodné složky se může pouze odhadovat. Často reprezentuje změny v závislosti na neznámé události, či jevu. Časová řada je následně realizací takovéhoto procesu[34].

5.2 Dekompozice časové řady

Časové řady mohou být obecně dekomponovány na složky [34]:

- trendová složka
- sezónní složka
- cyklická složka
- náhodná složka (reziduální)

Trendová složka

reprezentuje změnu průměrného chování řady na určitém časovém úseku. Je tvořena systematickým působením náhodného faktoru ve stejném směru[34].

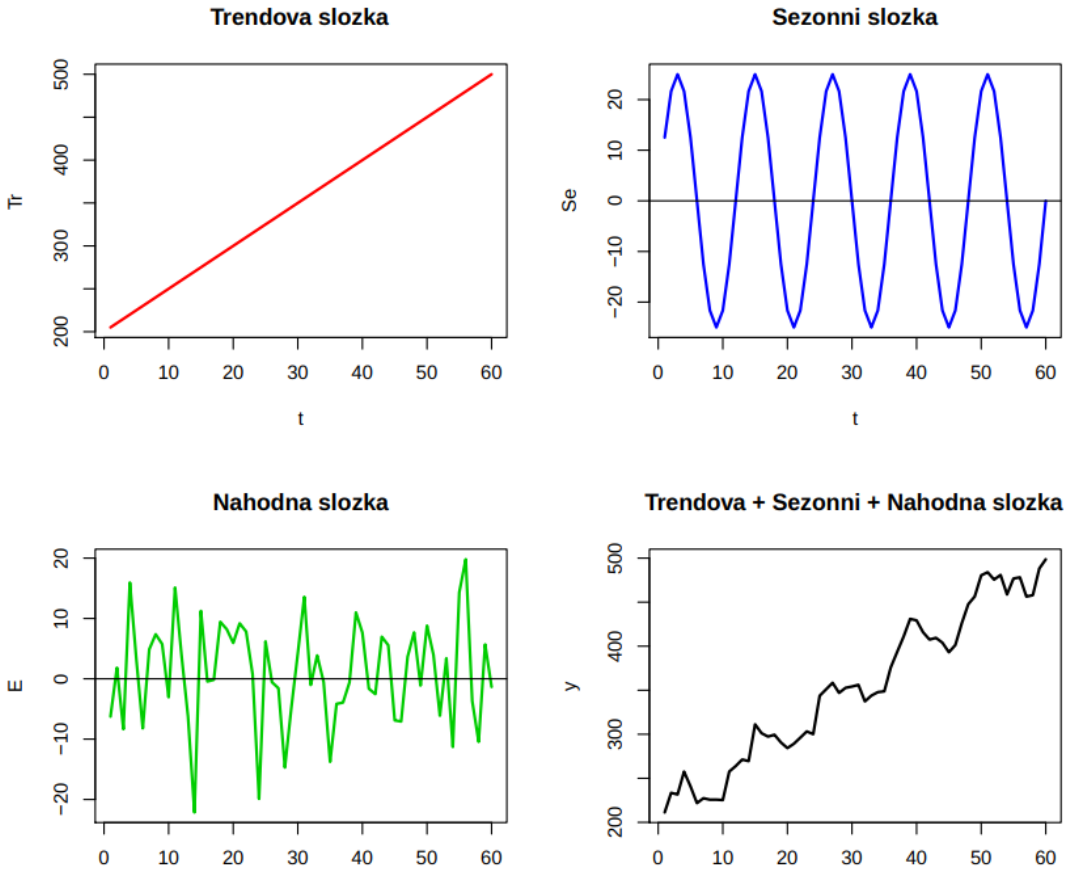
Pro popsání této složky časové řady se používají převážně techniky ze dvou kategorií, mechanické a analytické. Mechanické techniky určení trendu již byly popsány v kapitole 4.3.1, jedná se zejména o metodu klouzavých průměrů nebo metodu klouzavých mediánů časové řady. V tomto případě lze využít větší paletu různých rozšíření, od klasických prostých průměrů, vážených, centrovaných až po exponenciální nebo exponenciální vážené[34].

Oproti tomu analytická technika pro určení trendové složky je zejména regresní analýza, která analyzuje časovou řadu po očištění od složky cyklické i sezónní.

Sezónní složka

Vytváří periodické změny časové řady, které se v daném časovém rozmezí stále opakují, například kvartálně. Jsou způsobeny faktorem, který je závislý například na daném ročním období, nebo na ohlášení výsledků firmy.

Sezónní složka je obvykle popisována méně deskriptivními modely. Protože se analyzuje až po očištění od složky trendové, je viditelná zejména z grafu a pro její určení se používají sezónní indexy. Ty jsou většinou specifické pro jednotlivé obory, ze kterých časové řady vznikly, například ekonomické řady. Cílem těchto indexů je vyjádřit **sezónnost** řady v určených obdobích, kdy se nastavuje parametr S, vyjadřující délku sezóny ve složce (například čtyři pro čtvrtletní sezónnost, nebo dvanáct pro měsíční). V případě nepravidelné sezónnosti řady je nutné stanovit tzv. *sezónní faktory*, které se musí vždy součtem rovnat nule[34].



Obrázek 7: Dekompozice časové řady [4]

Cyklická složka

je podobná té sezonní, nicméně se neopakuje jen v přesném časovém úseku, ale i dle ne vždy známých pravidel například v ekonomice (recese - expanze). Tyto cykly se většinou stále opakují, ale nejsou vždy lehce odhadnutelné nebo vysvětlitelné vnějšími vlivy[34].

V případě dekompozice časové řady je cyklická složka často spojována se sezónní a k jejich rozdělení je nutné použít specifických znalostí oboru, protože se jedná u obou složek o periodické chování.

Náhodná složka

je tvořena náhodnou fluktuací v časové řadě, která nemá systematické vysvětlení nebo příčinu. Součástí náhodné složky je i chyba v měření, či technický / zaviněný výpadek měření. Statisticky je tato složka reprezentována bílým šumem v normálním rozdělení. Jako jediná složka není funkcí času, ale pouze posloupností náhodných veličin[34].

Každou z výše popsaných složek je možné popsat pomocí specifické techniky analýzy.

5.3 Box-Jenkinsova metodologie dekompozice časové řady

Box-Jenkinsova metodologie je striktně založena na vyšetření náhodné složky časové řady a snaží se pomocí korelační analýzy nalézt náhodné, ale zároveň závislé veličiny, které tuto

složku tvoří [34]. Po nalezení, je možné na základě těchto závislostí zpětně komponovat časovou řadu a s určitým poměrem úspěšnosti ji vysvětlit. Kromě toho, že tyto modely lépe vysvětlují neumělé časové řady, mají též lepší adaptabilitu na změny v časových řadách. Na druhou stranu, obecně je doporučováno tuto metodologii používat pouze pro časové řady, které mají více pozorování (například nad padesát).

5.4 Lineární modely

Jedním z předpokladů lineárních modelů metodologie je autokorelační vlastnost časové řady, resp. stacionarita.

Stacionarita popisuje časovou řadu jako stochasticky ustálenou, tedy že dvě libovolné pozorování závisí pouze na jejich vzájemné časové vzdálenosti, a nikoliv na umístění v řadě. Dle chování autokorelační funkce se může identifikovat vhodný model, dle kterého bude možné řadu co nejlépe popsat[34].

5.4.1 AR model

Autoregresní model popisuje časovou řadu s předpokladem, že jakoukoliv hodnotu je možné vysvětlit několika předchozími hodnotami s přispěním náhodné složky[34]. Obecný vzorec AR modelu: *AR(p)*, zapsaný následovně:

$$y_t = c_p + \phi_p y_{t-p} + \epsilon_t \tag{4}$$

, kde

- ϕ_p jsou koeficienty AR modelu
- ϵ_p je náhodný prvek modelu
- y_t je prvek časové řady v čase t

Autokorelační funkce (ACF)

Autokorelační funkce popisuje sílu lineární závislosti mezi dvěma pozorováními včetně náhodné složky[40].

Vyjadřuje se pomocí autokovarianční funkce:

$$\rho_k = \frac{c(y_t, y_{t-k})}{\sqrt{D(y_t)}\sqrt{D(y_{t-k})}} \tag{5}$$

, kde

- $c(y_t, y_{t-k})$ je autokovarianční funkce mezi prvky z časové řady
- $D(y_t)$ je směrodatná odchylka pozorování

Ve stacionárních řadách je však rozptyl konstantní, proto si jsou oba prvky ve jmenovateli rovny.

V reálné situaci je k dispozici pouze určité období pozorování, proto se používá odhad této ACF funkce, který lze vyjádřit[34]:

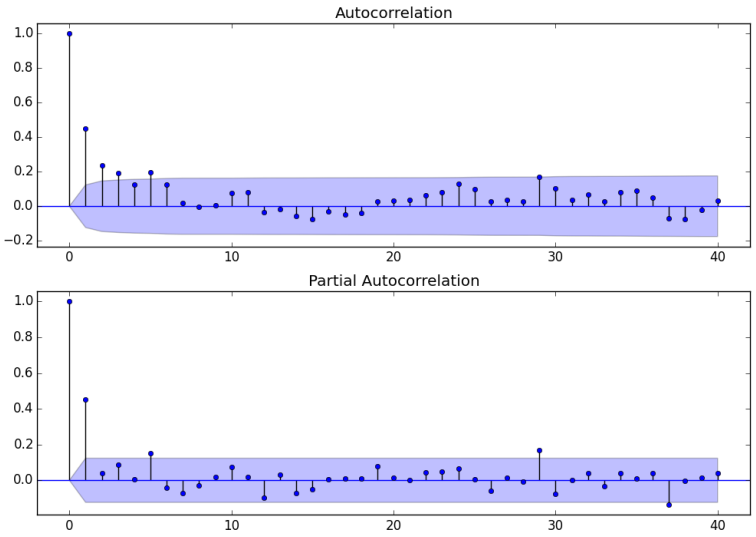
$$\hat{\rho_k} = \frac{\sum_{t=k+1}^{T}(y_t - \bar{y})(y_{t-k} - \bar{y})}{\sum_{t=1}^{T}(y_t - \bar{y})} \tag{6}$$

Parciální autokorelační funkce (PACF)

Pomocí ACF je možné vyjádřit vliv specifického prvku na prvek jiný, ale parciálně autokorelační funkce toto vyjadřuje bez vlivu ostatních pozorování, tedy pouze přímý vliv x_1 na x_5, bez vlivů prvků 2, 3 a 4. Kvůli této vlastnosti, je funkce též nazývána jako autokorelace podmíněná, protože v rámci jejího výpočtu se odebírá vliv ostatních hodnot a tím se podmiňuje výsledná hodnota[41].

Obecně lze funkci definovat jako:

$$\rho_k k = \frac{c(y, x_3|x_1, x)}{\sqrt{var(y|x_1, x_2)var(x_3|x_1, x_2)}} \tag{7}$$



Obrázek 8: Příklad ACF a PACF korelogramů [5]

5.4.2 MA model

MA model neboli model klouzavých průměrů je, jak již bylo zmíněno v kapitole 4.3.1, založen na průměrech několika posledních, za sebou jdoucích hodnot. Průměry ve stacionární časové řadě se však rovnají reziduím bílého šumu ϵ_t, proto se mohou použít místo samotných hodnot časové řady do předešlého modelu AR[40]. Funkce pro výpočet klouzavých průměrů q-tého řádu:

$$y_t = \epsilon_p + \theta_1 \epsilon_{t-1} \tag{8}$$

Střední hodnota tohoto procesu je nulová, tedy je stacionární pro jakýkoliv jeho parametr.

5.4.3 Model ARMA

Výše popsané procesy klouzavých průměrů a autoregresních posloupností lze kombinovat do modelu ARMA, který se zapíše pomocí funkce:

$$y_t = \theta_1 y_{t-1} + ... + \theta_q y_{t-q} + \epsilon_t + \theta_1 a_{t-1} - ... - \theta_q a_q \tag{9}$$

Střední hodnota procesu zůstává nulová a autokorelační funkce stále odpovídá té modelu AR[40].

5.4.4 Model ARIMA

Model ARIMA (I = integrovaný) má od modelu ARMA výhodu v tom, že vstupní časová řada nemusí být stacionární. Podpora nestacionárních časových řad je dána zavedením diferenčního operátora, který řadu na stacionární převede. Časová řada však musí být převoditelná do stacionární podoby [34].

Tento model se může zapsat pomocí zpětného posunutí jako:

$$b(B)W_t = w(B)\epsilon_t \tag{10}$$

$$W_t = \Delta^d Y_t \tag{11}$$

, kde

- d je řád diference
- Δ je diferenční operátor, tedy: $(1 - B)^d Y_t$

Finální podobu modelu tedy lze zapsat jako:

$$b(B)(1 - B)^d Y_t = w(B)\epsilon_t \tag{12}$$

Parametry tohoto modelu je pak nutné určit následující:

- **p:** je počet průměrovaných pozorování (MA)
- **d:** počet diferenciací původní časové řady
- **q:** počet reziduí, které budou použity pro AR model

Tyto parametry se často určují dle funkcí ACF a PACF[40].

5.4.5 Extenze modelu ARIMA

Pro zpřesnění odhadů, vznikly k modelu ARIMA i další odvozené modely.

SARIMA model vkládá do procesu odhad sezónnosti veličin

VARIMA kombinuje ARIMA model několika časových řad zároveň

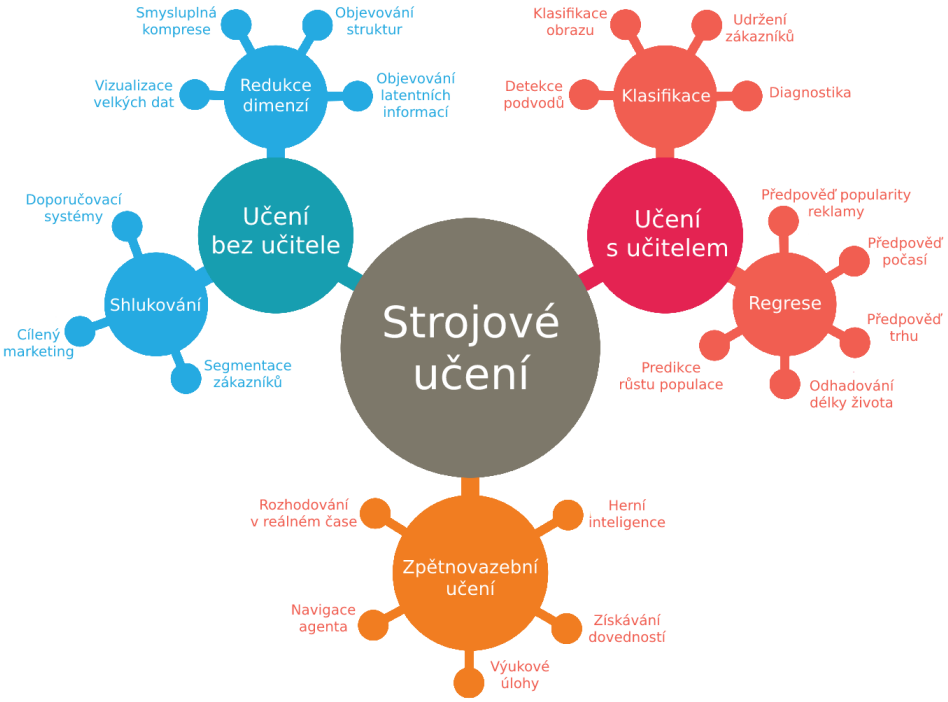
ARIMAX dodává do modelu proměnné relevantní pro časovou řadu

6 Strojové učení

Strojové učení *(ang. Machine learning - ML)* je podmnožinou oboru Umělé inteligence *(ang. Artificial Inteligence - AI)*. Umělá inteligence sama o sobě nemá přesně danou definici, avšak je většinou vnímána jako schopnost stroje napodobovat biologické schopnosti, například učení, plánování nebo uvažování. Strojové učení je následně jedna z částí, která se snaží napodobit učení se ze zkušeností. Oficiálně je často předkládána definice Arthura Samuela: „Strojové učení je obor, který dává počítačům schopnost učit se, aniž by byly předem přesně naprogramovány“[42].

Velmi často je u aplikací posuzováno, zda se opravdu jedná o typ strojového učení, nebo nikoliv. Mnohdy je tato skutečnost sama o sobě nerozhodnutelná. Existují programy, které učení pouze předstírají, tedy mají striktně předepsaná pravidla chování. Naopak však některé učící se systémy mohou konvergovat do stavů, kdy se chovají „až moc jednoduše“ vůči složitosti samotného modelu (nedoučená neuronová síť)[43].

Samotný obor strojového učení se dále dělí, toto dělení je dané typem úlohy, kterou aplikace bude vykonávat, nebo daty, které autor aplikaci poskytne.



Obrázek 9: Typy strojového učení [6]

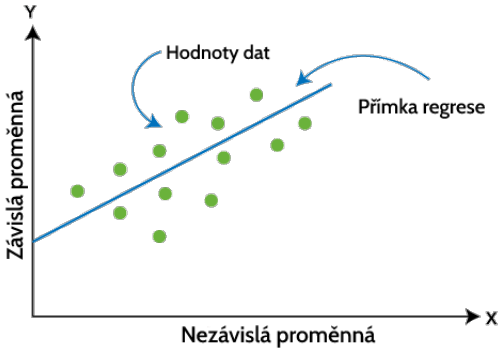
6.1 Učení s učitelem

První a základní kategorií ML je učení s učitelem, nebo též ang. Supervised learning. Toto učení, již dle názvu, funguje na principu opravy modelu dle reálného výsledku. Pro tento typ modelů musí autor poskytnout data s jasně daným cílem, tzv. „labeled“ data a mechanizmus sám na základě předem daných vlastností odhadne výsledek „label“ konkrétního případu. Často se s tímto typem lze setkat při odhadu cen, například domů - podle rozměrů, místa, vlastností a stavu[43].

6.1.1 Regrese

V rámci kategorie učení s učitelem, lze nalézt ještě dělení na problémy regresní a klasifikační. Regresní modely obecně odhadují vztah mezi závislou a jednou nebo více nezávislými proměnnými. Výstupem těchto modelů je často jedna spojitá proměnná.

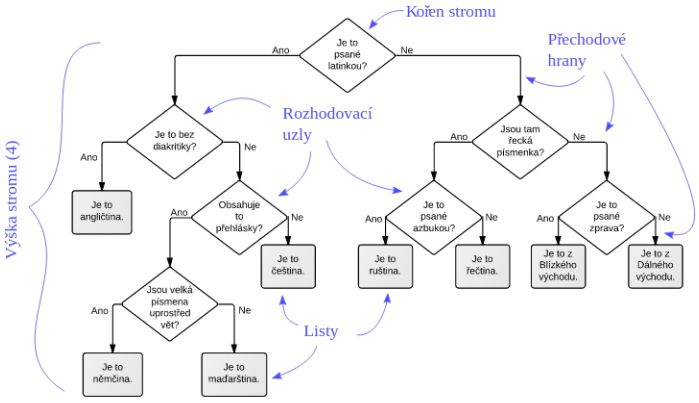
Lineární regrese je jeden z nejjednodušších modelů strojového učení vůbec. Tento model se snaží pomocí jedné, nebo více proměnných predikovat jednu výstupní hodnotu. Tato predikce probíhá za pomocí lineární přímky, která prochází prostorem proměnných a snaží se nejvíce přiblížit poskytnutým hodnotám.



Obrázek 10: Lineární regrese (přeloženo z: [7])

Kromě lineární, se velmi často využívá i polynomiální regrese s křivkou (polynomiální funkcí) nebo také multi-lineární regrese s lineární rovinou[44].

Rozhodovací strom, řazený do kategorie umělého učení s učitelem, je velmi často používaný model. Ve fázi učení, vytváří rozvětvený rozhodovací strom, který obsahuje rozhodovací uzly oddělující jednotlivé podstromy. Na koncových listech stromu je následně predikována hodnota, ta se rovná agregaci hodnot z poskytnuté trénovací sady, která se do příslušného listu kategorizovala. Následně, ve fázi rozhodování a za pomocí vstupních proměnných, prochází algoritmus celý strom a na každém z rozhodovacích uzlů vybere podstrom, kterým dále postupuje, dokud nenalezne cílový list stromu.



Obrázek 11: Rozhodovací strom [8]

Velkou výhodou je jednoduchost návrhu a implementace, avšak v případě složitějších rozhodování se strom enormně zvětšuje, čímž se zvyšuje i výpočetní zátěž[44].

Náhodný les (random forrest) je úzce spjat s metodou rozhodovacích stromů. Při sestavování modelu náhodného lesa se seskupuje předen určený počet rozhodovacích stromů. Každý z těchto stromů se „učí“ na jiném datovém setu, který vzniká za pomoci metody Bootstrap. Díky tomu je každý strom jiný a tak ve fázi dotazování predikuje i jinou hodnotu. V případě regrese se spočítá průměr predikovaných hodnot, avšak metod, jak zvolit výstupní hodnotu, je mnoho. Tento mechanizmus má hlavní výhodu v tom, že minimalizuje riziko chyby jednoho stromu, který například nedokonvergoval do správného stavu při učení[44].

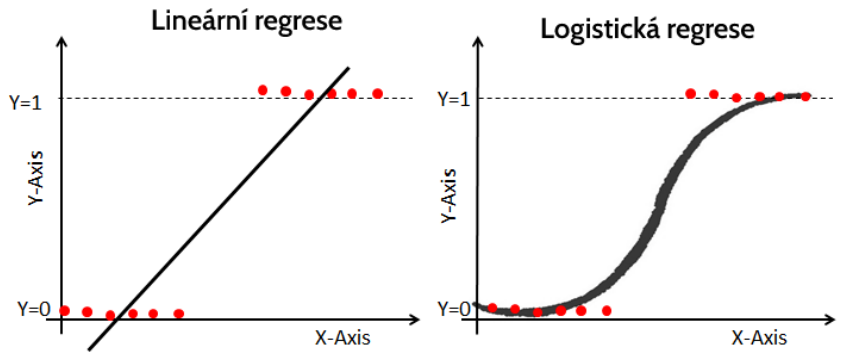
6.1.2 Klasifikace

Klasifikační modely, na rozdíl od regresních, poskytují jako výstup jednu hodnotu a ta označuje kategorii, do které model vstupní proměnné zařadil. Jedná se tedy o metody, které na základě vstupních proměnných klasifikují/rozhodují, do jakých (z předem stanovených) kategorií konkrétní pozorování patří. Reálným příkladem zde může být klasifikace jedince do skupin chata, domek nebo vila, dle rozlohy stavby.

Klasifikace se dále dělí na:

- **Binární klasifikaci**, která vždy předpovídá jednu ze dvou možností (0,1), ano / ne, ovčák / pudl.
- **Kategorickou klasifikaci**, ta generuje hodnotu označující jednu z několika skupin - švýcarský ovčák, bernardýn, retrívr, pudl atd.

Logistická regrese je velmi podobná lineární regresy. Zatímco v lineární regresi se model snaží mezi daty vytvořit přímku, logistický regresní model využívá funkce jako je například sigmoid, kterým rozhoduje o pravděpodobnosti, že pozorování spadá do určité skupiny. Výstup je tedy v intervalu <0,1>, kde nulu lze označit za „ne“ a jedničku za „ano“, případně jednu ze dvou případů (pes / kočka).



Obrázek 12: Porovnání regrese lineární a logistické (přeloženo z: [9])

Jak již bylo zmíněno, logistická regrese se často používá s funkcí sigmoid. Tato funkce je specifická svým tvarem písmene S, při správném nastavení parametrů dokáže rozhodnout,

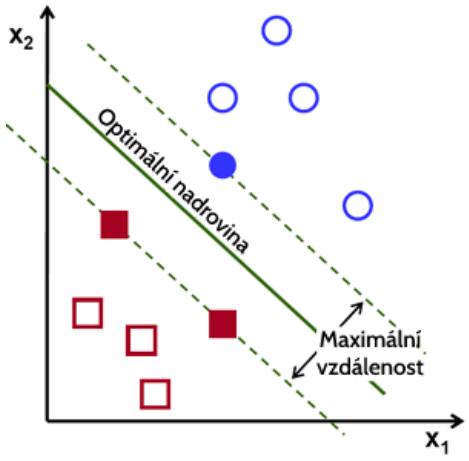
zda vstupní hodnota spadá pod kategorii jedna, nula a dle výstupu určí i pravděpodobnost této kategorie[43].

Definice této funkce je následující:

$$S(x) = \frac{1}{1 + e^{-x}} \tag{13}$$

Tato funkce byla též velmi důležitá při vývoji neuronových sítí, kde je vedle softmax, nebo ostatních tanh, relu, leaky relu a dalších, stále často používána.

Metoda podpůrných vektorů (SVM) neboli ang. *Support Vector Machine* (SVM) je metoda komplikovanější. Obecně vychází z myšlenky, že jednotlivé kategorie dat jsou od sebe, pomocí specifické kombinace vlastností, separovatelné. Z toho důvodu, metoda SVM hledá v bázi vlastností takovou nadrovinu, která maximalizuje vzdálenost mezi jednotlivými třídami. V praktické části si lze tento princip představit tak, že v bázi dat existují data o různých obyvatelích ČR a model má roztřídit obyvatele do kategorií mladí, středního věku a senioní. Kvůli tomu, že ze standardních vlastností člověka tyto kategorie definuje prakticky pouze věk, nadrovina se vytvoří právě v dimenzi vlastnosti věku.



Obrázek 13: Příklad SVM mechanizmu roviny (přeloženo z: [7])

Metoda Naive Bayes: další z velmi používaných metod je tzv. metoda Naive Bayes, která vychází z klasické Bayesovi věty:

$$P(y|X) = \frac{P(X|y) * P(y)}{P(X)} \tag{14}$$

Pomocí tohoto teorému je následně vytvořen model, který předpokládá, že všechny proměnné v modelu jsou na sobě nezávislé. Model si ve fázi učení dokáže vytvořit tabulku pravděpodobností, ve které se vlastnosti vyskytují vzhledem ke kategorii; dle nich následně vyhodnocuje[45].

Například:

Název kategorie	Výška	barva	směr uší
Švýcarský ovčák	vysoký(0.8)/ malý(0.2)	bílý(0.8)/ hnědý(0.2)	nahoru(1.0)/ dolů(0.0)
Zlatý retrívr	vysoký(0.8)/ malý(0.2)	bílý(0.3)/ hnědý(0.7)	nahoru(0.0)/ dolů(1.0)
Mops	vysoký(0.2)/ malý(0.8)	bílý(0.8)/ hnědý(0.2)	nahoru(0.6)/ dolů(0.4)

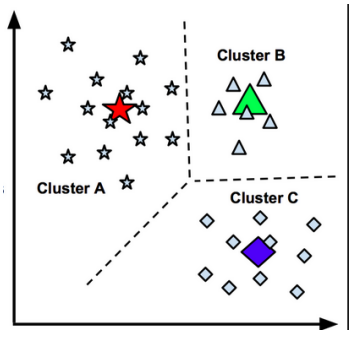
Model si ke každé kategorii a vlastnosti přiřadí pravděpodobnost, tedy při rozhodování o bílém vysokém psu s ušima nahoru, bez nutnosti pravidla, rozhodne o Švýcarském ovčákovi.

Metoda Bayeských sítí vychází ze stejného předpokladu jako metoda Naive Bayes. Bayesovské sítě však oproti předešlé metodě využívají i závislé proměnné. Všechny proměnné následně situují do acyklického grafu, kde orientovaná hrana značí pravděpodobnostní vztah proměnných. Hrany mezi uzly následně reprezentují pravděpodobnost vlastnosti za podmínky vlastnosti předchozí (P(B|A))[43].

6.2 Učení bez učitele

Opakem k učení s učitelem je kategorie učení bez učitele, v prvním případě měla aplikace jasně stanovený cíl nebo žádaný výsledek. V učení bez učitele, musí aplikace cíl sama vytvořit na základě podobných vlastností objektů nebo odpozorovaných vzorů v datech. Tato varianta učení je velmi často užívána pro doporučování nebo odhalování podvodů. Zajímavým případem je i použití tohoto mechanizmu pro určení podvodů, kdy má aplikace za úkol roztřídit data (například chování uživatelů v systému) do několika skupin, většina dat o chování bude v několika velmi blízkých skupinách, zatímco vymezené případy vzdálené od ostatních budou s velkou pravděpodobností anomálie - chyby systému, či pokusy o zneužití.

Shlukování neboli clustering je technika, při které se pozorování shlukuje do skupin na základě rozdílů a podobností. Objekty se stejnými nebo podobnými vlastnostmi jsou seskupeny do skupiny, zatímco ty, které nemají podobnost, do skupiny přidány nejsou. Počty skupin jsou pro tuto metodu určené již před samotným učením. Využití těchto metod lze nalézt při segmentaci obrazu, nebo trhu a uživatelů, ale také při analýze velkých dat.



Obrázek 14: Příklad shlukování [10]

Nejpoužívanější algoritmy této metody jsou K-means, hierarchické shlukování nebo shlukování středním posunem[43].

Redukce rozměrů je jednou ze složitějších metod, avšak zakládá se na jednoduché myšlence - data popsaná velkým množstvím vlastností se mohou popsat pouze vlastnostmi hlavními. Složité je při této metodě až rozhodnutí, které z těchto vlastností jsou ty hlavní.

Existují hlavní dva způsoby redukce:

- **Extrakce vlastností** (ang. *feature extraction*) je speciální technika, která se používá ve chvíli, kdy není možné určit hlavní vlastnosti. Algoritmus musí z vlastností extrahovat vlastnosti nové, které co nejvíce odpovídají rozložení původních. Především se tato metoda používá pro doporučování, kde není možné odhadnout, který produkt / uživatel je důležitější pro analýzu.
- **Odstranění vlastností** (ang. *feature elimination*) - z dat jsou vybrány nebo vytvořeny pouze hlavní vlastnosti, ty závislé nebo vysvětlitelné hlavními jsou odstraněny.

Jako příklady metod redukce rozměrů je možné uvést asi nejznámější metodu Analýzy hlavních komponent (PCA analýzu) a Lineární diskriminační analýzu (LDA)[44].

6.3 Učení se zpětnou vazbou

Třetím a nejspíše nejsložitějším typem strojového učení je učení se zpětnou vazbou. Tento typ učení se využívá především při delším procesu, kdy je nastaveno, nebo odhadnuto prvotní chování a každý další výsledek mechanizmu je následně posouzen zpětně. Do aplikace je poté vložena zpětná vazba, čímž se algoritmus průběžně zlepšuje. Klasickým příkladem, využití učení se zpětnou vazbou, je řízení auta, kdy má aplikace zadané určující chování. Při řízení následně točí volantem určitým směrem a zpětná vazba aplikaci poskytuje metriku, zda není moc blízko krajnici silnice, a pokud je - aplikace musí model upravit.

Obvykle se učení se zpětnou vazbou technicky charakterizuje jako benefitní cyklus agenta, kdy agent (aplikace) za každou správnou akci benefituje, zatímco za špatnou akci obdrží postih - negativní zpětnou vazbu. Tento styl učení je nejspíše nejblíže lidskému, protože agent interaguje s prostředím a realizuje cíl, kterého má dosáhnout pomocí zkoušky, tedy výhry / prohry.

Jednotlivé, výše uvedené typy strojového učení nejsou přesně vymezené a částečně se i překrývají, naopak lze v praxi velmi často vidět spojení dvou a více algoritmů do jediného, čímž například vznikla i samostatná kategorie **semi-supervised learning**. Tato méně četná kategorie kombinuje vlastnosti obou výchozích, tedy využívá menší množství dat se známým výsledkem a současně velké množství dat s neznámým výsledkem. Zároveň metoda učení bez učitele, která data separuje do skupin, pomáhá algoritmu učení s učitelem, který tak odhaduje výsledek vnitro-skupinově[43].

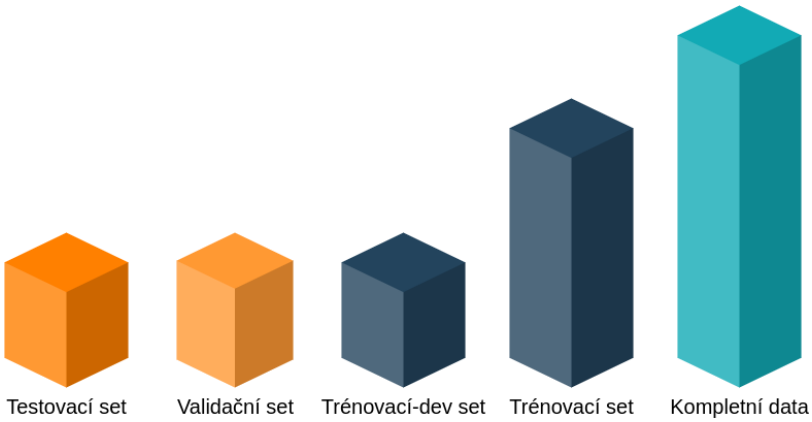
6.4 Obecné principy strojového učení

V této kapitole jsou popsány vlastnosti a přístupy, které jsou často aplikovány na modely strojového učení, zejména pak na neuronové sítě. Některé poznatky a pojmy budou zmiňovány v dalších pasážích práce, proto je nutné pojmy definovat již nyní. Dosud nevysvětlené oblasti jsou zde zmíněny pouze pro ucelení pojmů a budou detailněji popsány v následujících kapitolách.

6.4.1 Data

Pro většinu klasických algoritmů, které denně píší programátoři, jsou data potřebná až ve finální fázi - v testování produktu. Z toho důvodu, že píší kód imperativně, tedy pomocí příkazů přesně popisují, jak se s daty má nakládat. Strojové učení je však v tomto naprosto jiné, více podobné deklarativnímu programování, kde vývojář definuje „co“ se má udělat místo „jak“. Proto jsou data používána naprosto opačně, tedy jsou potřeba již při samotném začátku procesu.

Jak již bylo popsáno výše, mnoho těchto metod se nějakým způsobem „učí“, nebo alespoň agregují, proto při vzniku potřebují často velké množství dat, aby dospěly do stavu, kdy fungují dle očekávání. Jak ale poznat tento moment, nazývaný stav konvergence? Pomocí jiných, neučících se dat, kterými se model otestuje.



Obrázek 15: Používané datasety [vlastní zpracování]

Testovací množina: musí tedy existovat dvě množiny dat (ve strojovém učení tzv. datasety), jedny nazývané **trénovací**, pomocí kterých se model „naučí“, ty budou potřeba ihned po jeho vytvoření. Druhá množina dat se označuje **testovací**, ta přichází na řadu po trénování modelu a ověřuje, zda model pracuje dle předpokladů[19]. Tyto předpoklady se testují pomocí metrik, které jsou popsány dále v kapitole 8.4.

Validační množina: v modelech neuronových sítí však může nastat stav, kdy naučený model perfektně popisuje trénovací data, avšak testovací vůbec nepředpovídá. V tu chvíli se model musí nějakým způsobem upravit. Jeden proces trénování je však u některých modelů velmi časově náročný, z toho důvodu by nebylo efektivní model znovu přeučovat kvůli každé změně, protože těchto změn mohou proběhnout stovky. Proto se zavedl další set dat, nazývaný **cross-validation set**, nebo též **dev set**. Tato množina dat má za úkol natrénovaný model testovat s různými parametry a vybrat nejlépe vyhovující množinu. Ta musí být vždy ze stejné distribuce jako testovací množina, tedy data, pro která bude výsledná aplikace používána[19].

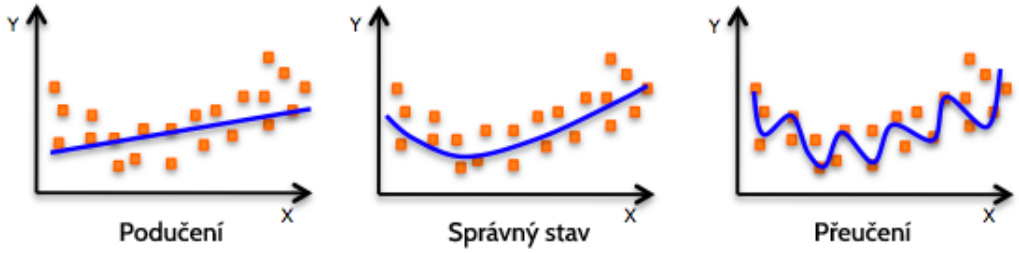
Training-dev set je čtvrtou a méně používanou množinou dat, která se zavádí ve chvíli, kdy existují data ze dvou rozdílných distribucí. Proč by však takováto data měla být použita? Kvůli tomu, že modely, obzvláště pak neuronové sítě, potřebují spousty dat a v některých aplikacích je sběr dat problematický. Tato množina se využívá pro situaci, kdy se

model učí na jiných datech, než na která bude reálně aplikován v provozu. Více o této technice bude popsáno v kapitolách níže, avšak v kontextu dat, v případě, že jsou data ze dvou zdrojů a jsou tedy nějakým způsobem rozdílná, se zavádí training-dev set, který zjišťuje efektivitu modelu po naučení[19].

Rozdělení: obecně se předpokládá rozdělení do těchto skupin v poměru 60 % trénovací data, 20 % validační a zbývajících 20 % testovací. Toto doporučení však není platné, pokud pro model existuje velké množství dat, například jeden milion. V tomto případě, by ve validačním i testovacím setu bylo dvě stě tisíc dat a to je jak zbytečné, tak i výpočetně velmi náročné, protože modely jsou optimalizovány na dávkové zpracování a toto množství by při testování enormně zatížilo výkonnou část. Proto je v těchto případech doporučeno postupovat dle svých možností, při datasetu o velikosti jednoho milionu záznamů je nejspíše dobré rozhodnutí 1 % pro testovací a 1 % pro validační set, protože i 10 tisíc pozorování je více než dostatečné množství pro evaluaci modelu[19].

6.4.2 Stavy modelů

Ve většině modelů strojového učení, resp. hlavně neuronových sítí, není možné předem předpokládat správnost modelu, kvůli tomu vznikají testovací množiny dat, které potvrzují správnost, ale i aplikovatelnost modelu na určitý problém.



Obrázek 16: Underfitting vs. overfitting (přeloženo z: [11])

Podučení: model, i přesto, že se přesně na problém hodí, nemusí kvůli specifickým podmínkám dosáhnout jakéhokoliv výkonu a to proto, že nedokonvergoval do bodu, kdy rozumí datům, která jsou mu vkládána. Tento stav se nazývá podučení (underfitting) a vyskytuje se hlavně z několika důvodů:

- Model je příliš jednoduchý pro porozumění datům
- Model se nestihl přizpůsobit datům, je nutné učit déle (epochy), případně přidat trénovací data

V případě prvního důvodu je nutné u modelu zvýšit složitost, například přidáním vrstev neuronové sítě (změnou architektury), nebo úpravou velikostí polynomů (například u regresí).

Vlastnost, kdy model nevyjadřuje data, je též nazývána biasem modelu[19].

Přeučení: naprosto opačným problémem je přeučení neboli overfitting. Tento stav je navozen poté, kdy se model datům natolik přizpůsobí, že akceptuje pouze vlastnosti již „viděných“ pozorování. V případě přeučení modelu je situace komplikovanější, protože nelze jednoznačně potvrdit příčinu tohoto stavu. Běžným řešením je přidání trénovacích dat, pro zlepšení variability množiny, nebo celkové zjednodušení modelu - u neuronových sítí se toto zjednodušení aplikuje pomocí regularizací.

Vlastnost, kdy model vyjadřuje pouze trénovací data, je též nazývána jako variance modelu[19].

6.4.3 Postup učení modelu

Při tvorbě a testování modelu je dle stavů nutné v tabulce upravovat data a strukturu modelu k omezení nežádoucích chování modelu:

Název množiny	Špatný výkon	Dobrý výkon
Trénovací set	bias - podučení	správně
Trénovací-dev set	variance - přeučení	správně
Validační set	variance (pokud neexistuje train-dev set), jinak bias	správně
Testovací set	variance - přeučení	správně

Každý řešený problém má též jinou míru řešitelnosti, proto se velmi často počítá průměrná lidská chyba, která vyjadřuje správný, nebo též chtěný stav výkonu modelu. Model by se měl k lidské chybě přiblížit, případně ji překonat. Po překonání procentní lidské chybovosti však bude zlepšení modelu spíše minimální, protože je v tu chvíli pro člověka již velmi obtížné model jakkoliv upravovat bez znalosti chybových vlastností.

Místo konkrétních hodnot je při testování modelu nutné sledovat poměry chyb na jednotlivých setech. Je například možné, že model bude mít chybovost 20 % na trénovací množině, kde by tato hodnota klasicky znamenala velký bias modelu, avšak pokud je lidská chyba 25 %, znamená to spíše dobrý výkon modelu. Naopak pokud model bude chybový z 1 % na trénovacím setu a 11 % na testovacím, pak model s největší pravděpodobností trpí přeučením, i když by 11 % klasicky znamenalo dobrou hodnotu.

V oblasti ML se lidské chyby přijímají jako aproximace optimální, nebo též Bayesovské chyby - teoretická minimální hodnota chyby modelu, kterou model nikdy nemůže z technických důvodů přesáhnout, například poškozením snímků, vadným měřením apod.

Jak je zřejmé z odstavců výše, tyto doporučení mohou být v praxi hůře proveditelná, ať již kvůli relativnosti, nebo složitosti problému. Proto se zavádí takzvaný bias - variance takeoff neboli rovnováha mezi biasem a variancí, tedy mezi podučením a přeučením. Model je v tomto momentu rovnováhy co možná nejvíce stabilní, s co možná nejlepšími výsledky bez dopadu na jiný učící problém[19].

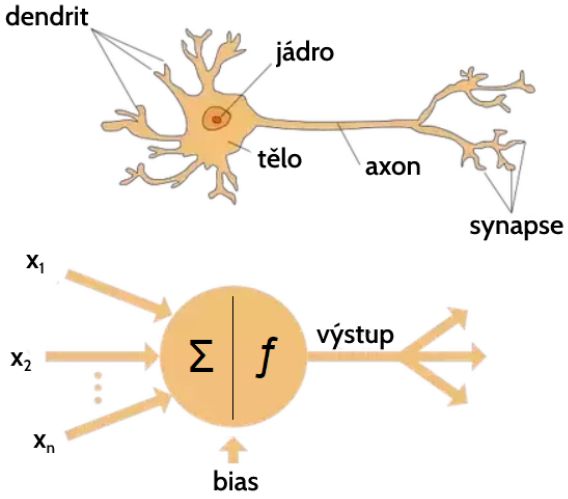
7 Neuronové sítě

Neuronové sítě *(ang. Neural network - NN)*, jak již název vypovídá, jsou inspirované tím nejsložitějším strojem, který vědci prozatím zkoumají - mozkem člověka. Lidský mozek zpracovává informace pomocí sítí neuronů, které přijímají vstup, zpracovávají ho a podle toho vydávají elektrické signály následujících neuronů, na které jsou napojeny.

7.1 Biologický původ

Původní snaha vědců byla vytvořit takový matematický model, který by vystihoval, jakým způsobem funguje lidský mozek. Neurofyziologie tedy sloužila jako zdroj inspirace při výzkumu neuronových sítí, avšak dnes známé modely se nesnaží přesně modelovat lidský mozek, ale snaží se napodobit jeho základní funkce nebo chování.

Navzdory původní inspiraci se však nyní stále méně vědců odkazuje na samotné fyziologické schéma, kvůli tomu, že sami neurovědci často nejsou schopni funkce neuronové sítě mozku popsat a také z druhého důvodu, některé typy umělých neuronů se totiž natolik vzdálily původní myšlence, že nejsou porovnatelné s neurony biologickými. [19]



Obrázek 17: Biologický vs. umělý neuron (přeloženo z: [12])

7.2 Neuron

Neuron, základní jednotka neuronové sítě, přijímá signály na vstupu (x_x), provádí vnitřní výpočet a odesílá do výstupu (y_y)[19].

Základní funkcí této jednotky je agregace informací obsažených v datech a tímto způsobem vytvoření informace nové.

7.3 Princip neuronu

Uvnitř neuronu se pro získání nové informace provádí výpočet:

Vynásobení vstupů vahami:

V rámci neuronu je první fází vynásobení vstupů váhami, které jsou různé pro každý ze vstupů. Prakticky se dá říci, že neuron v této chvíli filtruje data na ta, která ho nejvíce „zajímají“ a naopak omezuje ta, která pro neuron nejsou užitečná. Hodnota těchto vah vychází z fáze učení, která je popsána v dalších kapitolách, prozatím však lze říci, že tyto váhy jsou nastaveny tak, aby co nejlépe předzpracovali vstupy pro výpočet chtěného výsledku. Pokud je výsledek specifického neuronu rozpoznání psa a v rámci učení neuron zjistí, že vstupy 1 a 2 dodávají kočku, zatímco vstupy 3 a 4 generují rozpoznání psa; neuron intuitivně nastaví velké váhy (důležitost) na vstupy 3 a 4, zatímco váhy pro vstupy 1 a 2 budou minimální[19].

Vyjádřit tento výpočet lze jednoduše takto:

$$x_1 \rightarrow x_1 * w_1 \tag{15}$$

$$x_2 \rightarrow x_2 * w_2 \tag{16}$$

$$x_x \rightarrow x_x * w_x \tag{17}$$

Sečtení s odchylkou (bias):

Po znásobení váhami se všechny vstupy následně sečtou, čímž vznikne proměnná, která ilustruje celkový vstup do neuronu. Dále však každý neuron může obsahovat takzvaný bias. Bias je váha samotného neuronu a často se zobrazuje jako vstup 0. Tato doplňková odchylka (váha) má za úkol posouvat hodnotu vzniklého součinu o danou hodnotu, která je sama učitelná, podobně jako jsou samotné váhy vstupů. Učení tedy místo ovlivnění všech vah o konstantní hodnotu ovlivní pouze bias a tím posune všechny vstupy neuronu. Výsledek z této fáze součtu vstupů a biasu lze označit za proměnnou z[19].

$$z = x_1 * w_1 + x_2 * w_2 + ... + b \tag{18}$$

7.4 Aktivační funkce

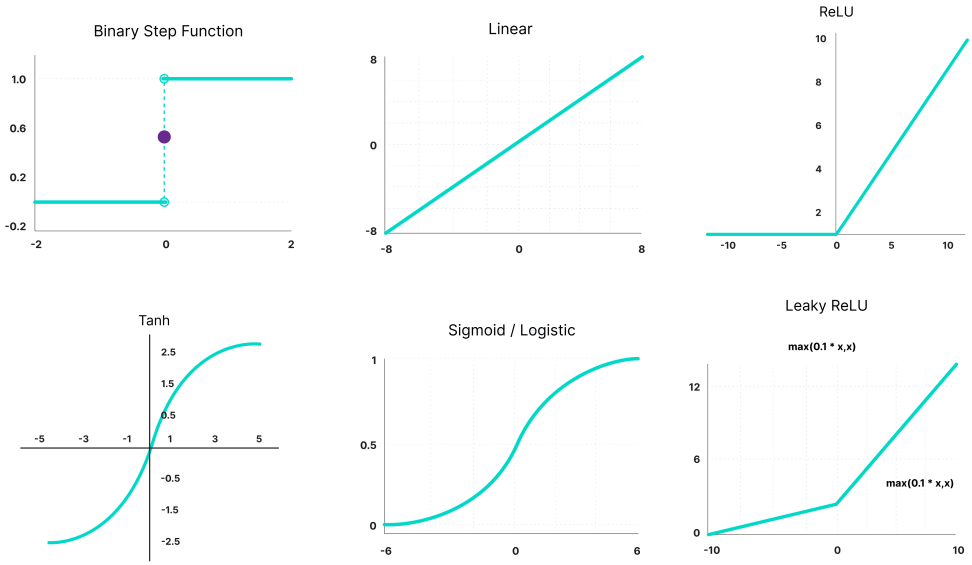
Aktivační funkce je nejspíše hlavní komponentou inspirovanou biologickým procesem v neuronech. V lidském neuronu se po přijetí informací skrze tzv. dendridy nahromadí akční potenciál, který určí, zda dojde k odeslání informace axonem dále, nebo k zadržení bez odeslání. V umělém neuronu tento princip funguje pomocí pravděpodobnostní funkce zvané aktivační. Proč je však tento princip potřeba? Kvůli nelinearitě modelu. Pokud by neuron pouze odesílal spočtenou proměnnou z, jeho výsledek by byl prakticky pouze lineární regresí a nebyl by schopen řešit složitější problémy[19].

Krok aplikace aktivační funkce je tedy možné vyjádřit jako:

$$\hat{y} = \sigma(z) \tag{19}$$

, kde

- σ je právě použitá aktivační funkce
- $\hat{y}$ je predikovaná hodnota/výstup z neuronu, někdy též označovaná jako a



Obrázek 18: Výběr aktivačních funkcí [13]

7.4.1 Kroková funkce

Nejjednodušší aktivační funkcí je kroková, která zajistí odeslání informace, pokud je proměnná z větší než 0 a informaci neodesílá, pokud je proměnná menší než 0 (resp. odesílá 0). Tato funkce je ale nediferencovatelná a z další kapitoly vyplyne, že je tedy nepoužitelná pro učení neuronu[13].

$$f(x) = \begin{cases} 0, & \text{if } x < 0 \\ 1, & \text{if } x \geq 0 \end{cases} \tag{20}$$

7.4.2 Lineární funkce

Jako další doplňkovou aktivační funkci je možné uvést prostou lineární funkci. Stejně tak jako kroková, ani lineární však není použitelná, protože její derivace je konstantní, neproběhlo by tedy žádné učení. Navíc, v případě neuronové sítě by jakékoliv zanoření pozbylo platnost, protože by se pouze sčítaly lineární funkce do jedné, tedy jediný význam by měla poslední výstupní vrstva jako pouhá lineární regrese[13].

$$f(x) = x \tag{21}$$

7.4.3 Sigmoidní funkce

Nejspíše první využívanou aktivační funkcí se stala známá funkce sigmoid. Stále běžně používaná funkce generuje pouze čísla v rozsahu (0,1). Jedná se vlastně o transformaci z intervalu $(-\infty, +\infty)$ na (0,1) - velká záporná čísla se stanou 0 a velká kladná čísla 1.

Zápis této funkce je:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{22}$$

Sigmoidní funkce se nejvíce používá pro rozhodnutí/pravděpodobnost mezi hodnotami 0 nebo 1 a je plně diferencovatelná, i z toho důvodu je dodnes běžně využívána. Hlavním negativem je však plochost její derivace, kdy jediné smysluplné gradienty obsahuje v intervalu <-3,3>, gradienty na jiných hodnotách jsou blízké nule. To způsobuje, že neuron se při vysokých a nízkých hodnotách neučí. Tato vlastnosti se v neuronových sítí nazývá mizející gradient. Druhým problémem této funkce je nesymetrie kolem 0, tato vlastnost pro neuron znamená, že jeho výstup bude vždy pouze kladný nebo pouze záporný[13].

7.4.4 Funkce tanh

Velmi podobnou funkcí je i funkce hyperbolického tangentu, s předpisem:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{23}$$

Graf této funkce je velmi podobný sigmoid funkci, avšak s rozdílem symetrie kolem 0, tím odstraňuje nevýhodu předchozí funkce a výstupní hodnoty generuje jak negativní, tak pozitivní. Často je možné se s funkcí tanh setkat ve skrytých vrstvách neuronových sítí, zvláště v rekurentních. Stále se však potýká s problémem mizejícího gradientu[13].

7.4.5 Funkce ReLu

Rectified linear unit aneb zkráceně ReLu, je nyní též velmi využívanou funkcí. Její graf působí dojmem linearity, avšak derivace není konstantní a následkem zlomu v hodnotě 0 jsou za různých hodnot aktivovány různé neurony (neuron nebude aktivován, pokud je výsledek neuronu pod hodnotou 0). Díky tomu a též díky linearitě v kladném definičním oboru výrazněji podporuje učení než například funkce tanh.

$$f(x) = max(0, x) \tag{24}$$

Funkce však obsahuje jednu nevýhodu, kterou je takzvaná vlastnost umírajícího ReLu. Váhy, které jsou popsány v předešlé kapitole, se v krátkosti aktualizují dle derivací aktivačních funkcí, tato skutečnost brání použití funkce krokové i lineární. Funkce ReLu má derivace záporných hodnot vždy nulové. To způsobuje, že se neurony se zápornými hodnotami již nikdy neaktualizují a nemají šanci se jakýmkoliv způsobem znovu zapojit do výpočtu, tedy změnit své váhy[13].

7.4.6 Funkce Leaky ReLu

Pro odstranění problému umírajícího ReLu, byla funkce upravena na definici:

$$f(x) = max(0.1x, x) \tag{25}$$

Tato nová funkce, stejně jako klasická funkce ReLu, záporné hodnoty velmi penalizuje, avšak povoluje neuronům učení i přes záporné hodnoty pomocí malého nenulového gradientu[13].

7.4.7 Funkce Softmax

Speciálním typem aktivační funkce, zároveň poslední z nejpoužívanějších, je funkce Softmax. Softmax je definována předpisem:

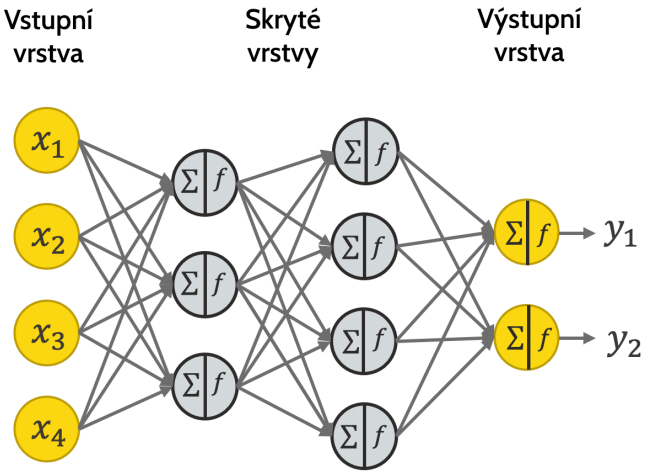
$$f(x) = \frac{exp(z_i)}{\sum_j \exp(z_j)} \tag{26}$$

Softmax je velmi často přirovnávána k funkci sigmoid, která generuje pravděpodobnost v intervalu <1,0>. V případě potřeby pravděpodobností více tříd však není použitelná, protože v tu chvíli je potřeba mít součet pravděpodobností všech tříd roven 1. Softmax tento problém řeší vypočítáním každé pravděpodobnosti všech tříd zvlášť a je tedy používána pro poslední vrstvy, kde dochází ke kategorizaci.

Existují i další aktivační funkce jako je rozšíření ReLu - parametrické ReLu, ELU, Swish, Dish, Gelu, Selu apod. Avšak tyto funkce jsou obvykle používány pouze ve specifických případech[13].

7.5 Struktura

Neuronová síť není nic jiného než shluk neuronů spojených dohromady. Podle svého posazení do sítě, má však každý neuron zvláštní vlastnosti a někdy i vlastní vnitřní funkce (o tom až v další kapitole). Dle pozice se neurony v síti řadí do následujících kategorií:



Obrázek 19: Základní neuronová síť (přeloženo z: [14])

Vstupní vrstva

V těchto vrstvách se vkládá informace z okolního světa pro neuronovou síť ke zpracování. Každý neuron této vrstvy představuje jednu informaci, která může mít jakoukoliv reprezentaci, plochu domu v m_2, hodnotu pixelu obrázku nebo hodnocení konkrétního zákazníka pro konkrétní produkt.

Skryté vrstvy

Skryté vrstvy provádějí veškeré zpracování pro neuronové sítě. Počet neuronů v této vrstvě není nijak definován a je většinou pouze doporučen v rámci řádů. Obecně lze říci, že čím více skrytých vrstev je v neuronové síti, tím přesnější síť bude, avšak je zde větší pravděpodobnost přeučení. Skrytých vrstev může být v modelu i více. Každá vrstva (prakticky

i neuron) může mít aplikovanou jinou aktivační funkci, většinou se však používá stejná u všech neuronů jedné vrstvy[19].

Výstupní vrstva

Je již pouze výstupem z celé neuronové sítě.

Síť, jak vyplývá z popisu vrstev, funguje následovně: do vstupních vrstev se z vnějšího světa zašlou data, ve skryté vrstvě se zpracují a předají do dalších skrytých vrstev (v případě plně propojené sítě, každý z neuronů zasílá data do všech neuronů další vrstvy v pořadí), z poslední skryté vrstvy se požadované informace zasílají na vrstvu výstupní, kde vzniká konečný výsledek modelu. Celý tento proces běhu neuronové sítě se nazývá dopředné šíření *(ang. forward propagation - feed forward neural network - FFNN)*[19].

Speciálním typem neuronové sítě je tzv. **Perceptron**. Perceptron je pouze jediný neuron, fungující samostatně. Jeho využití je však v dnešní době spíše menší.

7.6 Zpětné šíření chyby

Jak již bylo naznačeno v kapitolách výše, neuron se učí pomocí takzvaného zpětného šíření chyby. Zjednodušeně se jedná o funkci, kdy každý neuron upravuje své váhy podle toho, jak moc ovlivňuje výsledek celé sítě. Stejný princip se však aplikuje i uvnitř neuronu, kdy každá jednotlivá váha je upravena dle toho, jak ovlivňuje výstup z neuronu, pokud ovlivňuje výstup více, bude více upravena i její váha, pokud méně, bude váha upravena méně.

7.6.1 Chybová funkce

Chybová funkce - angl. Loss function je funkce, která definuje, jak moc se model zmýlil. Existuje velké množství chybových funkci a každá z nich je použitelná pro jiný typ problému, například pro rekurentní neuronové sítě se velmi často používá jednodušší funkce MSE - mean squared error neboli střední kvadratická chyba, definovaná jako: $MSE = (\sum(\hat{y} - y)^2)/n$. Pro počítačové vidění je běžná křížová entropie, účel této funkce je však stejný a to vyjádřit, jak moc se model spletl od reálné hodnoty.[19]

Speciálním případem jsou modely pro učení bez učitele, kdy si model vytváří chybovou funkci sám, například porovnáváním dvou výsledků a následným minimalizováním rozdílů mezi nimi.

Pro úplnost, u neuronových sítí se dále vypočítává tzv. cost funkce, která je chybou celé trénovací množiny dat, zatímco popsaná loss funkce je chybou pouze jednoho běhu trénování[44]. Cost funkce však vstupuje do učení pouze při aplikaci tzv. batch učení, který je popsán v dalších kapitolách.

7.6.2 Výpočet změn

Dlouhou dobu v historii nebylo možné neuronové sítě široce používat, protože nastavování jednotlivých vah bylo výpočetně extrémně náročné a neexistoval jednotný způsob, jak postupovat systematicky k minimalizaci ztrátové funkce. To vše se však změnilo s příchodem zpětné propagace, ta využívá takzvaný gradientní sestup. Gradientní sestup je metoda, která

vypočítává takzvaný gradient. Gradient je vlastně parciální derivace chyby $\mathcal{J}$(výsledek loss funkce) vzhledem k a (výsledek aktivační funkce). Tento výpočet lze vyjádřit jako:

$$\check{da} = \frac{d\mathcal{L}}{da} \tag{27}$$

da zde vyjadřuje míru vlivu *a* na výsledek chybové funkce. Stejný postup lze aplikovat dále v řetězci forward propagace a vypočítat vliv *z* (součet součinů vah se vstupy a biasu) a následně též vliv samotných vah[19]. Například:

$$\check{dz} = \check{da}\frac{da}{dz} \tag{28}$$

$$\check{dw}_1 = a_1^{l-1}\check{dz} \tag{29}$$

$$\check{da}_1^{l-1} = w_1\check{dz} \tag{30}$$

$$\check{dw}_2 = a_2^{l-1}\check{dz} \tag{31}$$

$$\check{da}_2^{l-1} = w_2\check{dz} \tag{32}$$

$$\check{db} = \check{dz} \tag{33}$$

Aktualizace vah a biasu je zprostředkována odečtením násobku gradientu s takzvanou learning rate (LR) neboli mírou učení. Tato míra vyjadřuje většinou předem danou konstantu, která udává, jak moc se budou váhy přizpůsobovat. Při nastavování tohoto parametru je nutné brát zřetel na velikost této hodnoty, protože nízká hodnota může zapříčinit velmi pomalé učení, zatímco velká hodnota bude upravovat váhy nadměrně a nikdy tedy nekonvergují do minima.

Celý tento výpočet je opakován pro každý neuron vrstvy zvlášť, a tedy je počítán i gradient a dopad každého neuronu na konečný výsledek. Gradienty se však kvůli tomu v každé vrstvě zmenšují, protože se gradient rozdělí počtem prvků. To zapříčiňuje skutečnost, že se vždy nejvíce ovlivňují poslední vrstvy a vrstvy počáteční se tedy musí učit daleko déle. Z toho důvodu také rozpoznávají spíše jednodušší principy (tzv. low-level informace). Toto chování se též nazývá problém mizejícího gradientu, kdy v některých případech neuronová síť ani nedokáže ovlivnit první neurony[19].

Metoda zpětné propagace je velmi složitá a mimo možnosti této práce, proto zde byla popsána spíše zjednodušeně. V rámci samotného algoritmu se v neuronových sítích dále udržují takzvané cache mezi dopřednou a zpětnou propagací, které urychlují výpočet a veškeré výpočty jsou vektorizovány. Během implementace tento mechanizmus obstarává samotná použitá knihovna, pomocí výpočetního grafu - *gradient tape*.

7.7 Obecné principy neuronových sítí

Již po uvedení základních principů je jasné, že použití neuronových sítí přináší nejen výhody, ale i několik problémů. I přes to je však jejich využití čím dál více rozšířenější. Částečně je to způsobeno obrovským výkonnostním potenciálem dnešních počítačů a grafických karet, na kterých výpočty počítač obsluhuje, ale též novými koncepty, které neuronové sítě od klasických, plně propojených vytváří.

7.7.1 Proč neuronové sítě místo klasických metod strojového učení?

Otázka, kterou si pokládá nejspíše každý inženýr před řešením nějakého problému. Hlavní výhodou neuronových sítí je ve většině případů fakt, že se průběžně zlepšují, nebo mají potenciál se s přibývajícími daty zlepšit. Klasické metody strojového učení, po dosažení určitého množství dat, již nezlepšují svůj výkon, zatímco neuronové sítě ano, ať již samy, nebo úpravou parametrů, přidáním vrstev[19].

7.7.2 Hluboké a mělké neuronové sítě

Při popisu neuronových sítí se velmi často používají pojmy deep (hluboké) a shallow (mělké) neuronové sítě. Definici těchto pojmů je velmi obtížné najít, intuitivně: neuronové sítě pouze s jednou skrytou vrstvou a několika neurony v ní jsou mělké modely (shallow). Tedy modely, které jsou celkem jednoduché a obsahují řádově menší množství parametrů. Opakem mělkých modelů jsou hluboké, tedy deep modely, ty obsahují například pět skrytých vrstev. Hlavním rozdílem těchto modelů je možnost interpretovat data, zatímco hluboké modely mohou tvořit zjednodušeně řečeno mnoha polynomiální funkce a tedy přesně abstrahovat data, mělké modely mohou tvořit pro deskripci dat pouze méně složité funkce. Nicméně hluboké sítě daleko více trpí na problémy, které byly popsané v předchozích kapitolách, například mizející gradient, nebo přeučení modelu[44].

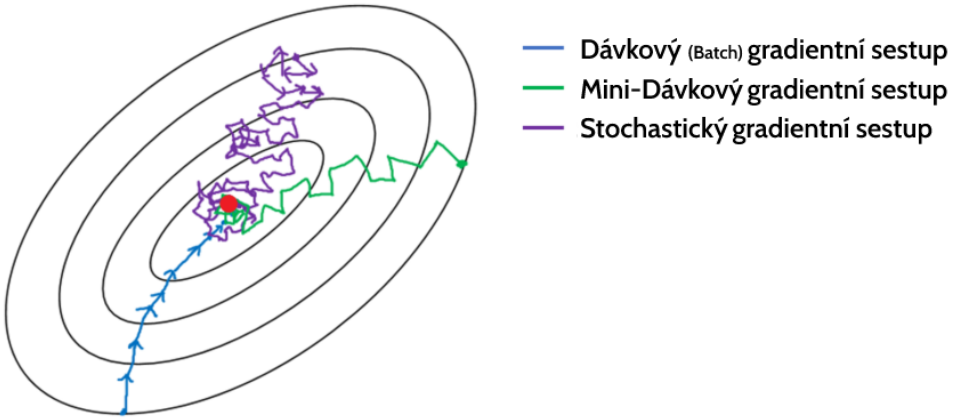
Obrázek 20: Mělká síť (vlevo) vs. hluboká síť (vpravo) [15]

Často se tyto přívlastky používají také pro algoritmy strojového učení, kde se klasická lineární nebo logistická regrese nazývá shallow metodou a naopak, kvůli své složitosti, se označuje například metoda PCA jako hluboká (to je však méně časté).

7.7.3 Batch vs. Mini-batch

Je zásadní, dle výše popsaného postupu učení a zpětného šíření chyby, že se algoritmus učí pomocí velkého množství dat. Minimalizace chyby se tedy provádí s ohledem na všechny prvky trénovací množiny. Zároveň je obecnou praxí provést tuto minimalizaci několikrát za sebou, aby se váhy upravovaly kontinuálně a dokázaly konvergovat do optima globálního místo místního. Je však nereálné učení celého množství dat, obzvláště při obrovském množství dat. Nejen kvůli velikosti paměti, ale i proto, že se výpočty provádí ve vektorizované formě a pro celé trénování musí být všechny načteny do paměti GPU. Z toho důvodu vzniklo upravení způsobu učení, z učení pomocí plného množství dat (batch trénování) na učení a trénování po menších vzorcích (mini-batch - mini-dávkové). Toto trénování je specifické v tom, že ne každý step (jeden průběh trénování na malém vzorku) optimalizuje - snižuje chybovost. Tato vlastnost je dána většinou hlavně obtížností dat v aktuálně prováděném vzorku, kde mohou být data například rozmazaná a učení nedokáže chybovost minimalizovat vzhledem k celé množině dat[19]. Obecně je však přijímáno, že po více iteracích dojde

i mini-batch trénování do stejného optima jako klasické batch trénování a kolem tohoto optima bude oscilovat. Pro zamezení oscilace se používá takzvaný learning-decay - zmenšení koeficientu učení (viz další kapitoly).



Obrázek 21: Chování gradientního sestupu (přeloženo z: [16])

Technicky se nastavení batch size, tedy počtu vzorku, na kterých se trénuje, na 1 nazývá stochastický gradientní sestup. Avšak vzhledem k velikosti dat není tato technika moc často aplikována, protože znemožňuje aplikovat vektorizaci a prodlužuje tím dobu učení[19].

Pro shrnutí: při trénování sítě je uváděn počet iterací (epoch), po které se má model učit. Při každé této iteraci vybere model počet dat dle nastavení batch-size a trénuje na takto velkém vzorku (toto trénování se nazývá step - krok).

Epocha se tedy skládá z počtu kroků rovnající se velikosti trénovacího souboru, respektive velikosti batch dávky.

7.7.4 Parametry vs. Hyperparamtery

Pro klasické strojové učení se většinou uvádí parametry algoritmu, ty dovolují autorovy algoritmu upravit jeho chování. U neuronových sítí jsou však těmito parametry váhy samotných vstupů a ty jsou aktualizovány automaticky pomocí zpětného šíření, proto se váhy a biasy jednotlivých parametrů nazývají parametry modelu. V hlubokých modelech počítačového vidění lze nalézt až stovky miliónů učitelných parametrů.

Naproti tomu však existují parametry, které se učit nedají, z již zmíněných se jedná o learning rate (míra učení), velikost dávky, počet iterací, skrytých vrstev, neuronů v jednotlivých vrstvách, nebo volba aktivační funkce (a optimizéru - viz dále). Tyto parametry se obecně nazývají hyper-parametry, protože běžně ovlivňují samotné parametry modelu a tvoří metamodel problému[19]. Hyper-parametry mají obvykle doporučené hodnoty pro různé, konkrétní problémy a je klasickým postupem začít nastavením hodnot, které již fungují na jiném, ale podobném problému. Je však praxí tyto parametry optimalizovat a zkoušet jejich přenastavování, buď na samotném trénovacím setu, nebo právě na validačním setu - viz kapitola 6.4.

7.7.5 Inicializace vah a biasů

Dalším problémem neuronových sítí je problém inicializací vah. Kvůli mechanizmu šíření chyb, je nutné na začátku učení váhy nastavit. Nejjednodušším způsobem se jeví nastavení na hodnotu 0, nebo na nějakou konstantu. Tento způsob však způsobí, že bude ze všech neuronů výsledek naprosto totožný, při zpětné propagaci se váhy upraví též stejně ve všech neuronech a tedy učení nesplní očekávaný efekt. Lepším způsobem inicializace vah je inicializace náhodnými hodnotami. To však obsahuje jednu velkou nevýhodu, pokud se váhy nastaví na vysoká čísla, pro hlubokou síť to bude znamenat, že zpětná propagace bude na konci sítě potřebovat obrovský gradient, aby mohla velká čísla vah na začátku sítě ovlivnit. Tato vlastnost se nazývá exploze gradientu a lze ji ovlivnit takzvaným klipováním (ohraničením) gradientu do mezí. To však neřeší původní problém, tedy že počáteční váhy nebudou upraveny tak, jak by bylo třeba a učení bude neefektivní. Druhým extrémem je pak již zmiňovaný úbytek gradientu. Pokud inicializace nastaví váhy nízké, gradient před propagací do prvních vrstev prakticky zmizí a tyto váhy neovlivní, učení je tedy opět neefektivní. Právě kvůli těmto problémům vzniklo několik algoritmů, jak váhy prvotně nastavovat. Tyto algoritmy jsou běžně implementovány v knihovnách a inicializují se dle zvolených aktivačních funkcí[44].

7.8 Konvoluční neuronové sítě (CNN)

Klasické, tedy plně propojené neuronové sítě však nedokáží vyřešit všechny problémy, které poskytuje reálný svět. Lze si například představit situaci, kdy má neuronová síť identifikovat chlapce na obrázku. Síť se naučí na velkém množství dat, kdy jako vstup bude vždy plný soubor všech pixelů obrázku. Po naučení, chlapce na obrázku opravdu identifikuje, avšak chlapec je na druhém obrázku posunutý a místo v pravém dolním rohu, je na druhém obrázku v levém horním, a síť ho již nedokáže identifikovat. Toto je celkem reálný příklad, protože klasická neuronová síť je závislá na konkrétních vstupech do neuronů a také očekává stále stejné vstupy, které vždy identifikují určitou vlastnost. U obrázků se však tato vlastnost může pozičně měnit a tedy není tento princip aplikovatelný[19].

Ve strojovém učení se obecně manipulace s obrazy nazývá počítačové vidění.

Konvoluční sítě nejsou hlavním oborem této práce, ale přesto jsou v tomto oboru využívané a aplikovatelné i v oboru rekurentních sítí. Právě z tohoto důvodu jsou v této kapitole definovány základní principy těchto technik, avšak použití v samotné oblasti počítačového vidění je obor přesahující možnosti této kapitoly.

7.8.1 Typy problémů počítačového vidění

Počítačové vidění řeší několik základních typů problémů.

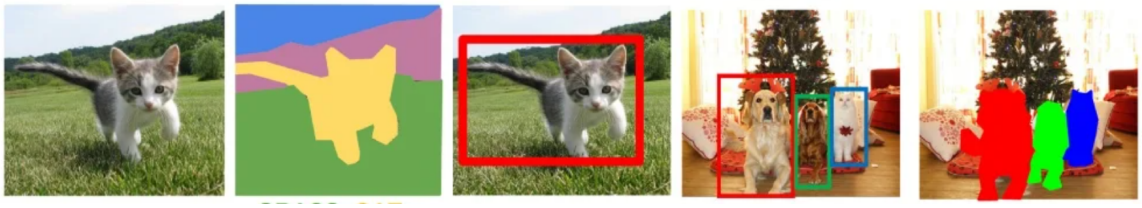
Klasifikace obrazu je nejspíše nejzákladnější metodou počítačového vidění a má za úkol rozhodnout (kategorizovat), jaký objekt je na daném obrázku. Problémem tohoto přístupu je, že algoritmy rozpoznají chlapce na fotografii na jakémkoliv místě, ale již nedokáží určit počet všech chlapců - pouze pravděpodobnost, že na fotografii nějaký chlapec je.

Klasifikace s lokalizací je dá se říci nadstavbou klasifikace, kdy pomocí různých algoritmů (jako například ohraničujících boxů) metoda přesně určí lokalitu daného jedince, avšak nedetekuje objekty ostatní.

Detekce objektů lze popsat jako další stupeň klasifikace s lokalizací, kdy tyto modely detekují všechny nalezené objekty v obrazu, vykreslí kolem nich lokalizační box (určí jejich přibližnou polohu) a následně klasifikují.

Sémantická segmentace je oproti předešlým metodám jiná - místo klasifikace se snaží přesně určit, které pixely patří, kterým objektům a místo označení přibližného místa objektu tedy definují přesné pixely.

Segmentace instance je kombinací výše uvedených metod, kdy se pro detekované objekty vyplňuje prostor pixelů pomocí segmentace.



Obrázek 22: Typy problémů počítačového vidění [17]

7.8.2 Konvoluční filtry

Jak již bylo popsáno v úvodu této kapitoly, pro zpracování obrazu je nutné použít nějakou metodu, která není závislá na přesné lokalitě informace ve vstupu. Z tohoto účelu se do neuronových sítí začaly zapojovat konvoluční filtry, které se používají i mimo strojové učení.

Tato metoda předpokládá zpracování obrazu jako matice hodnot intenzity barvy (většinou hodnot do 255, které se následně normalizují na rozsah <0,1>).

Zjednodušeně jsou konvoluční filtry matice použité jako filtr, který detekuje nějakou vlastnost obrazu. Tento filtr se aplikuje postupně na obraz zleva doprava, ze shora dolů a počítá násobek čísla ve filtru (matici) a čísla intenzity barvy v obrazu. Tento násobek se následně sečte do jediné hodnoty a filtr se po směru posune dále v obrazu.

Konvolučnímu filtru se též říká kernel neboli konvoluční jádro. Je nutné podotknout, že názvosloví strojového učení zde určuje konvoluci jako operaci vysvětlenou níže, v rámci matematiky se však tato metoda nazývá křížová korelace matic. [19]

9	9	9	1	2	2
9	8	8	2	1	1
8	9	9	1	2	2
9	8	9	2	1	1
8	9	8	1	2	2
8	9	8	1	2	2

1	0	-1
1	0	-1
1	0	-1

Obrázek 23: Průběh konvoluce [vlastní zpracování]

Jak je názorně zobrazeno na obrázku 23 - výpočet začíná vynásobením filtru vpravo s levou 3x3 submaticí $\begin{pmatrix} 9 & 9 & 9 \\ 9 & 8 & 8 \\ 8 & 9 & 9 \end{pmatrix}$ a sečtením všech násobků - tímto způsobem vyjde do výsledkové matice 192. Filtr se následně posune o jeden sloupec vpravo a nově aplikuje na

submatici $\begin{pmatrix} 9 & 9 & 1 \\ 9 & 8 & 2 \\ 8 & 9 & 1 \end{pmatrix}$, tímto způsobem se aplikuje dále v řádku, následně se posune zpět do levého okraje a aplikuje o řádek níže na submatici: $\begin{pmatrix} 9 & 8 & 8 \\ 8 & 9 & 9 \\ 9 & 8 & 9 \end{pmatrix}$.

Výsledek konvoluce z obrázku 23 je tedy: $\begin{vmatrix} 0 & 22 & 21 & -1 \\ 0 & 20 & 22 & 1 \\ -1 & 22 & 21 & -1 \\ 0 & 22 & 20 & -1 \end{vmatrix}$

Pokud se použije jako filtr matice.

Z příkladu je zřejmé, že řád původní matice se zmenšil a tím „extrahoval“ informace do menšího počtu buněk. Dále je také vidět vzor v prostředních sloupcích výsledné matice. Použitý filtr je totiž tzn. detektor vertikálních hran, který po aplikaci dosadí vysoké hodnoty těm submaticím, které obsahují hranu - změnu intenzity barvy. Těchto speciálních filtrů na různé vzory lze najít nespočetně.

Hlavní výhodou aplikace v neuronové síti je, že filtry neobsahují přesně dané detektory vertikálních nebo horizontálních hran, ale hodnoty, které spíše odpovídají vahám v klasické neuronové síti. Tato vlastnost dává konvoluční vrstvě (filtru) možnost naučit se vlastní hodnoty a tím detekovat informace nebo vzory bez předešlého určení[19].

Po naučení jsou však tyto vzory pro člověka již neznámé. Existuje metoda, která neuronové síti předkládá různé vstupy a sledováním aktivací jednotlivých neuronů dokáže odhadnout, jaké tvary jednotlivé neurony (filtry) detekují.

Stejně jako v síti plně propojené, se i v konvoluční mohou vrstvy řadit za sebe a dospět tak ke složitějším informacím. Podobně platí, že počáteční vrstvy detekují nízkoúrovňové informace (hrany) a vrstvy na konci řetězce detekují vysokoúrovňové tvary (ucho, nos).

Dále je nutné podotknout, že obraz je tvořen mřížkou odpovídající pixelům obrazu, avšak tyto mřížky jsou povětšinou tři. Každá z nich pro jednotlivou barvu (r, g, b,), tedy i konvoluční filtr je třírozměrný a k výpočtu naznačenému výše se sčítají ještě další dvě konvoluce zbylých dvou vrstev.

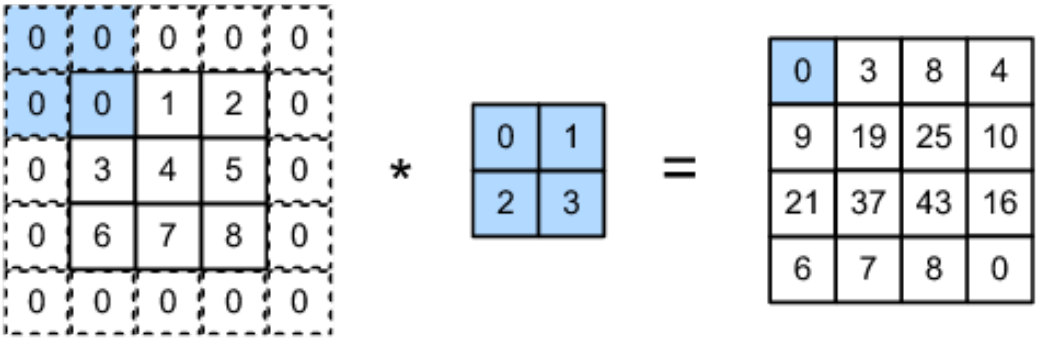
Výhodou použití těchto filtrů je hlavně přenositelnost informací z jedné pozice vstupu k jiné, tedy detekce chlapce je přenositelná z jednoho místa na jiné. Zároveň je díky extrakci vlastností každý výstup z filtrů závislý jen na malém množství vstupů, místo závislosti na celém obrazu.

Jak již bylo výše uvedeno, konvolucí obrazu o třech barvách s filtrem vznikne dvourozměrný výsledek, pro udržení třetí dimenze je možné aplikovat dva samostatné filtry a výsledek každého sloučit jako jednu vrstvu.

Pro aplikaci v neuronové síti je důležité zmínit, že se po vytvoření výstupní matice, k celé této matici (pro každý element) přičte bias a následně se použije aktivační funkce, jak je zvykem v klasických neuronových sítí. Pokud je výsledek složený několika filtry, převod na 3D objekt se provádí až po aplikaci aktivace - do té doby jsou počítány jako jednotlivé výsledky každé dimenze[19].

7.8.3 Padding

Konvoluční filtr začíná svůj výpočet na straně obrazu, avšak některé hodnoty se posouváním matice vkládají do výpočtu častěji, a mají tedy větší vliv na výsledek než jiné. Toto chování má za následek úbytek informací z krajů, které do filtru mohou vstoupit pouze jednou. Aby se tomuto jevu zabránilo používají neuronové sítě takzvaný padding, tedy obal. Aplikace je velmi jednoduchá, okraj vstupní matice se na každé straně pouze rozšíří o sloupec a řádek nul[19].



Obrázek 24: Padding [18]

7.8.4 Typy konvolucí

Při aplikaci konvoluce se používají dva speciální typy výpočtů[19]:

Valid konvoluce je konvoluce bez použití paddingu a velikost výsledné matice je rovna $n - f + 1$, kde f je velikost filtru a n velikost původní matice.

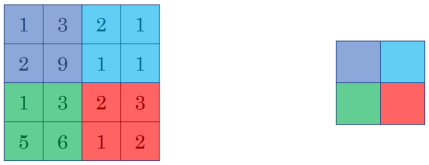
Same konvoluce naproti tomu využívá padding k dorovnání velikostí, a tedy se rozměry vstupní a výstupní matice rovnají.

7.8.5 Stride

, neboli krok je druhou technikou v aplikaci konvoluce, jedná se o chování, kdy se filtr při výpočtech neposouvá pouze o jeden sloupec / řádek, ale o více. Respektive, klasický výpočet konvoluce naznačený výše lze označit za konvoluci o kroku (stride) 1. Pokud při konvoluci bude uveden stride 3, posune se filtr po aplikaci o tři sloupce vpravo[44].

7.8.6 Pool vrstvy

Dalším typem vrstvy kromě konvolučních je tzv. pool vrstva, která funguje obdobným způsobem, avšak pro každou submatici spočítá průměr hodnot (bez vynásobení) a tuto hodnotu vloží do výsledné matice. Na obrázku 25 je znázorněn pooling o stride 2.



Obrázek 25: Pool vrstvy [19]

Využívají se dva typy pool vrstev, average pool vrstva, která pro každou submatici vypočítá průměr hodnot - pro příklad výše by byl výsledek $\begin{pmatrix} 3.75 & 1.25 \\ 3.75 & 2 \end{pmatrix}$. Druhým typem je max pooling, který vybere pouze nejvyšší hodnotu ze submatice. Každý z těchto typů je určený pro jiné funkce - v případě 25 tedy $\begin{pmatrix} 9 & 2 \\ 6 & 3 \end{pmatrix}$.

Největším přínosem pool vrstev je agregace informací do menšího počtu buněk, konvoluční filtry aplikované po pool vrstvách jsou méně závislé na lokalitě informace v matici[43].

7.8.7 Využití architektur

Po zavedení konvolučních a poolovacích vrstev lze konstruovat celou jednoduchou konvoluční síť, která je často tvořena vrstvami:

$$conv- > pool- > conv- > pool- > FC- > FC- > vystup \tag{34}$$

FC = klasická plně propojená vrstva neuronů

V klasických neuronových sítích bylo velmi problematické najít správnou architekturu, tedy správný počet neuronů a vrstev tak, aby fungovaly na specifický dataset. Díky jiným principům architektury modelů, jsou z počítačového vidění velmi dobře přenositelné na jiný dataset a dokonce i na jiný typ problému. V čase proto vznikly velmi známé architektury zabývající se různými problémy, při implementaci je tedy možné pouze přepoužít již vyvinutou architekturu a například jen vyměnit poslední vrstvy, které odpovídají klasifikátoru.

Jedny z nejznámějších architektur jsou LeNet, AlexNet, ResNet požívající reziduální vazby, Google Inception používající inception moduly, MobileNet s hloubkově oddělitelnou konvolucí a parametrizovatelný EfficienNet[19].

Reziduální vazby: díky technikám jako je pooling, nebo i samotná konvoluce, se informace agregují a tím matice zmenšují. To má za následek ztrátu části informace, která by v pozdějších vrstvách mohla být užitečná. Z toho důvodu vznikly takzvané reziduální vazby, které z určitého bodu sítě přenáší výstupy do hlubšího místa v síti a tím obnovují informace, které byly předešlým zpracováním odstraněny.

1x1 konvoluce je speciální konvoluce agregující pouze prostorovou informaci, po aplikaci se tedy z matice $nxnxm$ stává $nxnx1$. Velmi často je používáno několik 1x1 konvolucí za sebou, které tvoří znovu původní rozměr $nxnxm$[43].

Inception bloky vychází z myšlenky, že není možné předem definovat konvoluci jakého rozměru by bylo nejlepší aplikovat, nebo zda před konvolucí (ne)aplikovat ještě pooling. Z toho důvodu tento blok aplikuje všechny možnosti, respektive max. pooling s 5x5, 3x3 i pouze 1x1 konvolucí[43].

Hloubkově oddělitelná konvoluce: při aplikaci konvolučních filtrů se v klasickém výpočtu sčítají násobky všech barevných vrstev do jedné hodnoty. Pro ušetření paměti lze barevné vrstvy počítat každou zvlášť. Výstupů z tohoto řešení je stejný počet matic, jaká byla hloubka vstupu (3 barevné vrstvy). Aby byl výsledek dimenzionálně shodný, jako při aplikaci klasické konvoluce, provádí se ještě konvoluce bodová (konvoluce 1x1), která odstraní třetí dimenzi[43].

7.8.8 Transfer learning

Jak bylo naznačeno v úvodu kapitoly, v rámci počítačového vidění se velmi často používají již vytvořené architektury. Dokonce je však možné použít i celý předučený model, včetně vah. V takovém modelu jsou základní vrstvy naučené na primitivní informace a pro specifický problém je možné přeučit pouze vrstvy poslední, které identifikují obecné informace. Tento princip velmi zkracuje čas trénování až z týdnů na hodiny a zároveň dovoluje experimentovat s různými architekturami bez nutnosti je týdny učit.

7.8.9 Augmentace dat

Jednou z výhod počítačového vidění je velká možnost augmentovat data. Augmentace slouží hlavně pro dva účely, prvním z nich je zvětšení trénovací množiny, druhým pak donucení modelu k menší závislosti na přesných datech.

Z pohledu obrázků, augmentace znamená chtěné poškození obrazu, například otočením kolem os, rotací, zkreslením, či rozmazáním. Po aplikaci těchto transformací lze vzniklý výsledek považovat za nový prvek trénovací množiny. Kromě toho však tyto transformace donutí model více zvariabilnit své váhy pro predikci, a to i na takto poničených datech.

7.8.10 Shrnutí dalších přístupů počítačového vidění

Obor počítačového vidění je velmi složité téma a využití neuronových sítí v něm je obrovské, proto zde spousta technik nebyla vůbec jmenována, například skip konekce, techniky slicing window pro lokalizaci, encoder-decoder schémata nebo populární siamské a generativní sítě. Definovaná témata jsou však natolik univerzální, že se v posledních letech využívají i mimo počítačové vidění, kupříkladu v predikcích časových os.

7.9 Pokročilé techniky v tvorbě neuronových sítí

Použitím složitějších vrstev, a především tvorbou hlubších struktur, se tyto sítě staly více náchylné k problémům. Je stále těžší pro implementátory vybírat a upravovat hyperparametry tak, aby byl model co nejvíce přesný a přitom netrpěl jedním z úskalí popsaných výše. Kvůli tomu byly vyvinuty techniky, které část těchto problémů řeší, aniž by explicitně zasahovaly do struktury (architektury) sítě.

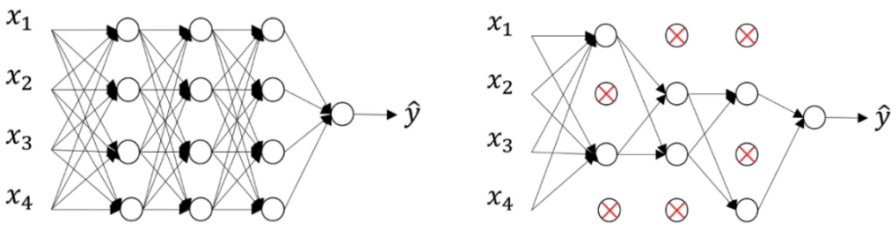
7.9.1 Regularizace

Je jednou z těchto technik. Zjednodušeně se jedná o metody, které se snaží model donutit k větší abstrakci nad daty. Nesnižují přitom komplexitu modelu ani nevynucují zvětšení množiny vstupních dat.

L1 a L2 regularizace jsou dvě techniky zakládající se na myšlence, že model s menšími hodnotami vah méně spoléhá na určité vstupy. Jinými slovy, pokud nebude existovat převládající vstup (vstup s velkou vahou), bude jednodušší popsat objekt ostatními vlastnostmi a model nebude spoléhat na jednu konkrétní vlastnost.

V případě regularizace L2, při každé aktualizaci vah je konkrétní váha zmenšena koeficientem její druhé mocniny, větší váhy se v důsledku toho zmenšují rychleji než váhy původně menší, trénováním je tedy model donucen váhy udržovat nízké. Pokud model snížil svou chybovost, velikost gradientu se také sníží a tedy ani aktualizace, ani koeficient zmenšení L2 nebude váhu upravovat. Tato regularizace se též nazývá Ridovou regresí[19]. Regularizace L1 neboli Lassova regrese obdobně zmenšuje váhy, avšak v tomto případě snižuje absolutní hodnotou všechny váhy vrstvy, bez ohledu na jejich relativní důležitost (velikost).

Dropout: další technikou, která má za úkol snížit závislost modelu na některých vlastnostech dat, je technika Dropout - vynechání. Jedná se o metodu, která záměrně při trénování některé neurony přeskočí a donutí tak neurony následující nespoléhat na hodnotu přeskočeného neuronu. Tato funkce tedy způsobí neexistenci konkrétní hodnoty na vstupu do další

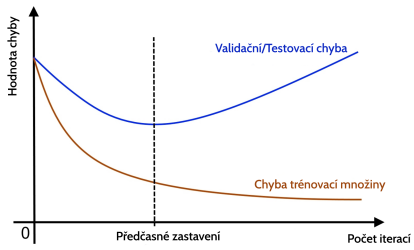


Obrázek 26: Dropout regularizace [20]

následující vrstvy a tedy vynechání informace z přijatých dat. Model se poté musí naučit splnit svůj úkol i bez této informace[19].

Vynechání neuronu je dáno pravděpodobností P, která se určuje pro každou vrstvu jako hyperparametr.

Zpětnovazební techniky: jako další techniky, avšak již s méně regularizačním účinkem, jsou uváděny například:



Obrázek 27: Technika early stopping (přeloženo z: [20])

Early stopping - brzké zastavení, kdy model po dosažení určité úrovně přesnosti již nedokáže nadále zlepšovat výsledky na neznámých datech (testovacích) a zpřesňuje pouze trénovací. To má, především u modelů počítačového vidění, za důsledek větší náchylnost na specifické vlastnosti dat a tedy i přetrénování. Kvůli tomu byla vyvinuta technika brzkého zastavení, která zastaví učení modelu, pokud se několik epoch za sebou nezlepšil výkon modelu[19]. Pro tuto techniku se udává parametr čekání, kdy algoritmus zastaví model až poté, co model nezlepšil chybovost po x epochách - například pokud se od 5 epochy nezlepšil výsledek a model má Early stopping nastavené s na 5, zastaví model na 10 epoše.

Normalizace vstupů je dalším doporučeným postupem v tvorbě modelů, který vychází z toho, že se model lépe učí, pokud neexistují velké rozdíly v hodnotách vah. Rozdíly v hodnotách vah jsou tvořeny zejména rozdílem v hodnotách vstupů, proto je doporučeno vstupy převést na jednotné měřítko pomocí normalizace, která všechny vstupy do neuronové sítě převede do prostoru například <0,1>.

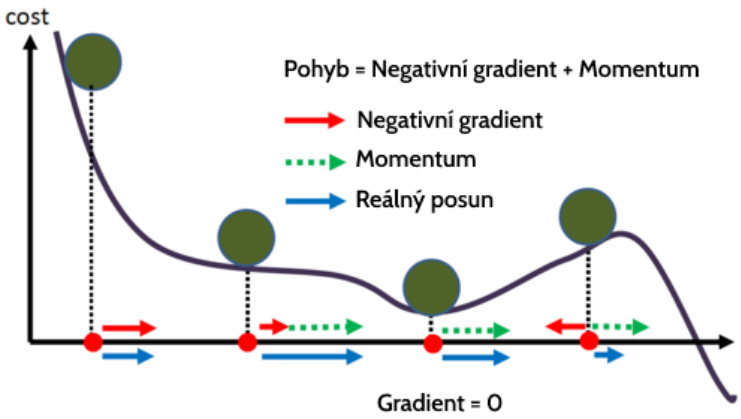
Podobný princip je dále možné uplatnit i uvnitř sítě, kdy je mezi dvě specifické vrstvy, většinou hlouběji v modelu, vložen normalizační článek, který výsledky z jedné vrstvy (respektive vstupy do vrstvy druhé) převzorkuje na stejný interval. To má za následek, stejně jako regularizace, donucení modelu spoléhat na vlastnosti stejnou mírou[19].

7.9.2 Optimizéry

Prozatím byly v práci popsány příklady s technikou gradientního sestupu a jeho rozšíření mini-batch a stochastický gradientní sestup, avšak to nejsou jediné techniky, které dokáží model donutit k aktualizaci vah. Tyto další techniky se obecně nazývají optimizéry, neboť optimalizují chybu modelu k minimu.

Gradientní sestup je velmi mocná a využívaná technika, avšak obsahuje problém sedlového bodu, který nastane, pokud okolí chybové funkce po určitou dobu neklesá. V tu chvíli algoritmus usoudí, že se nachází v minimu - gradient vychází blízký nule a nemá tedy potřebuje dále optimalizovat[19]. Tento stav je též nalezením lokálního minima. Ve vícerozměrných prostorech se však ukázalo, že spíše než lokální minima, obsahují tyto prostory sedlové body, které obdobně gradientní sestup zastaví. Pomocí jiných metod však lze ještě z tohoto sedlového bodu „uniknout“.

Momentum: aby se vyřešil problém se zastavením v sedlovém bodě, vyvinula se takzvaná technika Momentum, neboli hybnosti. Zjednodušeně se jedná o techniku, která ke gradientům násobí hybnost a ta je výsledkem předchozích kroků. Jinými slovy, pokud byly gradienty vysoké v posledním kroku, nyní by měly násobit vyšším číslem, aby došlo k rychlejšímu učení. To též znamená, že pokud optimalizér dojde do sedlového bodu, Momentum donutí gradient překročit tento bod a oscilovat okolo. Tento jev následně pomáhá k odklonu ze sedlového bodu.



Obrázek 28: Optimalizér s Momentum technikou (přeloženo z: [21])

Často se tento algoritmus popisuje na kutálejícím se fotbalovém míči z kopce a chlapci, zatímco chlapec se každým krokem konstantně posouvá z kopce dolů, míč nabírá stále větší rychlost a přesto, že se chlapec bez vynaložení síly zastaví v sedlovém bodě, míč se přes sedlový bod dokáže překutálet[19].

Momentum lze použít i pro zpětné vylepšení algoritmu gradientního sestupu, kdy z původního výpočtu:

$$\theta_j = \theta_j - \epsilon \Delta_{\theta_j} \mathcal{L}(\theta) \tag{35}$$

přidá se rychlost změny z minulých gradientů:

$$v_{t+1} = \rho v_t + \Delta_{\theta_j} \mathcal{L}(\theta) \tag{36}$$

$$\theta_j = \theta_j - \epsilon v_{t+1} \tag{37}$$

, kde ρ je rychlost změny, kterou se původní hodnoty hybnosti propagují.

Adagrad je svým způsobem podobný aplikaci algoritmu Momentum, avšak místo aplikace vypočtené relativní hybnosti, zde minulé gradienty algoritmus sčítá, následně pak fungují jako jmenovatel pro aplikaci gradientu. Tato funkce je snadněji popsána funkcí:

$$g_0 = 0 \tag{38}$$

$$g_{t+1} = g_t + \Delta_{\theta_j}\mathcal{L}(\theta)^2 \tag{39}$$

$$\theta_j = \theta_j - \epsilon\frac{\Delta_{\theta_j}\mathcal{L}}{\sqrt{g_{t+1}} + 1e^{-5}} \tag{40}$$

V prvním výpočtu g_{t+1} je vypočtena kumulace všech předchozích gradientů aplikovaných na druhou mocninu, pro každou dimenzi. V druhé rovnici jsou pro změnu theta aktuální gradienty vyděleny upravenou kumulací. Tento proces má za následek zmenšení gradientů ve směru, ve kterém bývají větší (z kumulace) a naopak zvětšení ve směrech (dimenzích), ve kterých jsou relativně menší. S přibývajícími epochami se však kumulace gradientů stávají většími, a tedy se veškeré změny rapidně zmenšují (dělením kumulací), to způsobuje naprostý útlum optimalizační metody v čase[19].

RMSProp je vylepšení metody AdaGrad. Kumulaci historických gradientů navíc násobí specifickým koeficientem α a aktuální gradient ke kumulaci přidává vážením $1 - \alpha$.

$$g_0 = 0 \tag{41}$$

$$\alpha =\simeq .9 \tag{42}$$

$$g_{t+1} = \alpha g_t + (1-\alpha)\Delta_{\theta_j}\mathcal{L}(\theta)^2 \tag{43}$$

$$\theta_j = \theta_j - \epsilon\frac{\Delta_{\theta_j}\mathcal{L}}{\sqrt{g_{t+1}} + 1e^{-5}} \tag{44}$$

Tato menší úprava odstraňuje problém umírajícího potenciálu Adagrad algoritmu a sice pomaleji, ale velmi přesně postupuje ke chtěnému minimu. Velmi často je tento algoritmus porovnávám s algoritmem SGD upraveným Momentem, který konverguje relativně rychleji, ale s daleko větší cestou, která je potencionálně nebezpečná kvůli výkyvům lokálních optim nebo zmíněných sedlových bodů a ty by optimalizér zpomalily[43].

ADAM zjednodušeně kombinuje jak velké výhody přesnosti RMSProp, tak rychlost algoritmu Momentum. Algoritmicky je to přesná kombinace obou algoritmů:

$$m_0 = 0, v_0 = 0 \tag{45}$$

Momentum:

$$m_{t+1} = \beta_1 m_t + (1 - \beta_1)\Delta_\theta \mathcal{L}(\theta) \tag{46}$$

RMSProp:

$$v_{t+1} = \beta_2 v_t + (1 - \beta_2)\Delta_\theta \mathcal{L}(\theta)^2 \tag{47}$$

Korekce biasů:

$$\hat{m}_{t+1} = \frac{m_{t+1}}{1 - \beta_1^t} \tag{48}$$

$$\hat{v}_{t+1} = \frac{v_{t+1}}{1 - \beta_2^t} \tag{49}$$

Samotná kombinace obou algoritmů:

$$\theta_j = \theta_j - \frac{\epsilon}{\sqrt{\hat{v}_{t+1}} + 1e^{-5}} \hat{m}_{t+1} \tag{50}$$

ADAM do aplikace algoritmu Momentum vkládá nový hyperparametr β_1, díky kterému Momentum zpomalí hybnost v případě opravdu rychlého „spádu“. V případě RMSProp nahrazuje α druhým hyperparametrem β_2, který má stejný princip jako původní alpha[19].

Problém však nastává již při prvních inicializacích, kdy jsou m_0 a v_0 nastaveny na 0. ϵ je tedy při prvních bězích děleno malým číslem a generuje číslo velké, které by způsobilo velkou změnu v neznámém směru, která určitě není chtěná. Kvůli této vlastnosti se do algoritmu přidává korekce biasů, která normalizuje proměnné m a při prvních bězích a tím kompenzuje prvotní absenci historických dat.

Nastavení hyperparametrů se většinou předpokládá kolem hodnot $\beta_1 \approx 0.9$ a $\beta_1 \approx 0.999$, avšak jako ostatní hyperparametry jsou i obě proměnné často objektem dalších experimentů[19].

Nutno zmínit, že ADAM je v dnešní době takzvaným state-of-the-art optimizérem a je používán nejspíše ve většině aplikací klasických neuronových sítí.

Úpadek LR: jak je popsáno výše, velikost proměnné learning rate neboli rychlosti učení je velmi důležitá. Její vysoká hodnota může zapříčinit nedoučení modelu, protože se váhy přenastaví vždy „až moc“ a tedy nikdy nedojde ke konvergenci k minimu. Naopak, příliš malá hodnota této proměnné způsobí to, že se váhy budou aktualizovat pouze o malý koeficient a tedy učení může trvat enormně dlouho oproti vyšší hodnotě LR. Tuto rovnováhu mezi velkou a malou hodnotou není vždy lehké odhadnout a proto vznikla technika Learning rate decay - úpadek rychlosti učení, která s přibývajícími epochami snižuje tuto proměnnou. Tudíž, čím déle se model učí, tím méně se aktualizují váhy a tím více se chybovost dokáže přiblížit chtěnému minimu[45].

Snížení lze vyjádřit například jako:

$$\alpha = \alpha_0 \frac{1}{1 + decay\_rate + epoch\_num} \tag{51}$$

prakticky se však používá několik principů výpočtu tohoto snížení.

Na obrázku je viditelná situace, kdy je vlevo nastavena příliš malá hodnota rychlosti učení, zatímco vpravo je naopak nastavena hodnota velká, která nikdy do minima nedokáže zkonvergovat. Algoritmus LR dacay prakticky použije kombinaci těchto faktorů, kdy nejprve využívá velkou hodnotu k rychlému trénování, následně model již obsahuje nějaké „znalosti“ a je tedy chtěné upravovat jej méně a méně, proto je přínosné proměnnou zmenšovat.

Podobnou funkcionalitu jako tento algoritmus přináší i tzv. LR schedulling, který upravuje proměnnou rychlosti učení postupně po krocích.

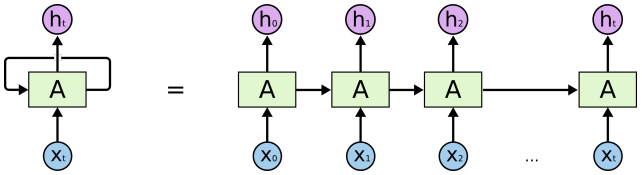
Obrázek 29: Rozdíly nastavení LR (přeloženo z: [22])

7.10 Rekurentní neuronové sítě (RNN)

Potřebu ke vzniku rekurentních neuronových sítí lze velmi snadno přirovnat již ke zmíněným konvolučním sítím, zatímco v nich model zpracovává velké maticové vstupy reprezentující obrázky, v rekurentních sítích je to často informace časová, nebo poziční. Problém je však velmi podobný, pokud by například větu zpracovávala standardní neuronová síť, byla by přesně závislá na konkrétních vstupech. Pokud by se jednalo o jednoduché věty: „Jedl jsem koláč.“, „Koupil jsem kolo.“, byla by s největší pravděpodobností i úspěšná. Potíž by však nastala při přehození slov, například: „Včera jsem navštívil manželku.“ a „Manželku jsem navštívil včera.“, věty sice obsahují stejné informace, avšak na jiných místech (pozicích). Klasická neuronová síť by tedy v druhém případě očekávala na prvním místě informaci „Kdy“, ale získala by „Koho“. Tento rozpor by byl v rámci zpracování srovnatelný s případem, kdy by neuronová síť predikovala cenu domů a vstup by měl prohozené informace o velikosti pozemku a ložnice - model by v tu chvíli nebyl schopný správně vyjádřit, respektive pochopit, co znamená $852m^2$ ložnice, ani $12m^2$ pozemku. Jak již bylo naznačeno výše, rnn sítě se používají zejména pro zpracování přirozeného jazyka (takzvaného NPL) nebo časových řad[45].

7.10.1 Intuice rekurentních sítí

Více technicky rekurentní neuronové sítě zavádí možnost aplikace na sobě závislých vstupů. Z minulých zkušeností si ukládají do paměti vlastnosti s daty a na základě těchto zkušeností rozhodují o konkrétním pozorování. Klasicky je do rekurentní sítě vkládáno několik pozorování za sebou, případně věta, jedná-li se o zpracování jazyka. Z každého pozorování si síť uchová nějakou informaci a vypočítá vlastní výsledek. Jako výstupy této vrstvy lze označit dvě proměnné, prvním a klasickým výstupem je proměnná, vypočítaná v rámci neuronu $\hat{y}$ (označovaná někdy jako h_t). Druhým výstupem je však nová proměnná a_n, která obsahuje matici „vah“, která byla v neuronu aktualizována o informace ze zpracovávaného vzorku. Tato proměnná funguje jako druhý vstup do dalšího neuronu v řadě. Díky tomuto mechanizmu se rekurentní neuronové sítě zobrazují buď za sebou, jako posloupnost neuronů propojených vazbou a předávající si proměnnou a_x, nebo jako jediný neuron se zpětnovazební smyčkou. Toto druhé zobrazení naznačuje jednu vrstvu rekurentní sítě, která obsahuje stejný počet neuronů jako první dimenze vstupů.



Obrázek 30: Způsob fungování rekurentní sítě [23]

Od klasických neuronových sítí se výpočet v rámci jednoho neuronu liší. Prvním rozdílem je, že se v rnn velmi často používá aktivační funkce tanh místo známého relu a to kvůli vlastnosti symetrie kolem osy, která v rekurentní síti lépe přispívá k paměti.

Výpočet v klasickém neuronu lze vyjádřit zkráceně jako:

$$\hat{y} = a = \sigma(W[x] + b) = \sigma(z) \tag{52}$$

v rekurentní síti je přidán vstup z předchozího neuronu a^{t-1} jako:

$$a_k = \sigma_a(W_a[a^{t-1}, x^t] + b_a) = \sigma(z) \tag{53}$$

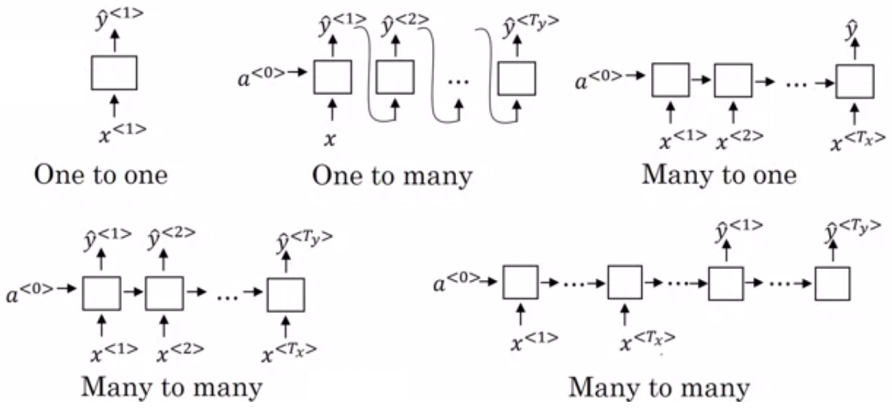
avšak na rozdíl od klasických neuronů se a nerovná finálnímu výsledku$\hat{y}$. V tomto případě se nová hodnota a rovná nové hodnotě paměti, kterou si rekurentní síť udržuje a je možné v tuto chvíli hodnotu odeslat z neuronu do dalšího. Pro výpočet "druhého"výstupu - predikce, je ještě nutné tuto novou hodnotu vynásobit vlastními výstupními vahami a aplikovat druhou aktivační funkci, která může být již jiná, například softmax pro klasifikaci:

$$\hat{y} = \sigma_y(W_y[a^t, x^t] + b_y) \tag{54}$$

, nutno podotknout, že v tomto případě již proměnná a obsahuje hodnotu z tohoto neuronu, jelikož předchozí výpočet obsahoval hodnotou z neuronu minulého (a_{t-1})[19].

Při prvním běhu neuronové sítě se paměť (a_0) nastavuje jako prázdný vektor. Tato paměť neuronu se též nazývá skrytý stav neuronu.

Typy RNN sítí: jak již vypovídá předchozí kapitola, rekurentní sítě jsou opravdu mocné nástroje a dokáží být aplikovány na spousty různých problémů. Pro každý se však musí nastavit jejich struktura. Nejzákladnější strukturou je one-to-one, která prakticky není rekurentní sítí, ale pouze neuronem s klasickým jedním výstupem, avšak označuje se jako „typ“ rekurentní sítě.



Obrázek 31: Typy rekurentních sítí [24]

Výše popsaná je však tzv. metoda many-to-many, která přijímá x vstupů a stejný počet výstupů. Obecně se používá i jako mezivrstva, kdy další rekurentní vrstva přijímá samotné výstupy z vrstvy předchozí. Takto lze rekurentní sítě vrstvit jako klasické hluboké neuronové sítě.

Dalším velmi používaným typem je vrstva many-to-one, která je využívaná například v predikcích časové řady, kde se jako vstup využívá několik posledních pozorování a jediný výstup predikuje následující hodnotu v řadě.

Třetí, velmi používaná metoda je many-to-many avšak v takzvané roztržené metodě, kterou lze vidět vpravo dole na obrázku 31. Tato struktura vrstvy se nejčastěji používá pro překlad přirozeného jazyka, kdy počet výstupů z překladače není stejný jako počet vstupů - věta v českém jazyce bude například kratší než ta přeložená do angličtiny.

Posledním z využívaných struktur je one-to-many, která dle všeho není používaná tak často jako výše uvedené, ale slouží například pro generování skladeb. Do vrstvy je zadán jediný vstup (často více rozměrný) a vrstva pro jeho hodnoty vygeneruje postupně hodnoty, kde každá z hodnot závisí na několika hodnotách předchozích[19].

Mizející gradient: velkým problémem rekurentních sítí je, stejně jako u řady ostatních metod neuronových sítí, mizející gradient. Díky tomu, že každý neuron ovlivňuje paměť pomocí vah a jiné váhy použije pro výstup, je při zpětné propagaci gradient poměrově štěpen a proto se do prvních neuronů zpropaguje velmi málo. Kvůli této vlastnosti je nutné základní rekurentní síť budovat strukturálně jednoduchou, kde dělení gradientu nevytvoří takový problém[45].

GRU: problém s mizejícím gradientem lze řešit i jiným způsobem než je zjednodušení modelu, a to aplikací GRU modulu (neuronu) místo klasického neuronu rekurentní sítě.

GRU - *Gated Recurrent Units*, neboli uzavírané rekurentní sítě. Tento typ sítě, na rozdíl od klasických rekurentních bloků, podmiňuje uchování historické informace pomocí takzvané aktualizační a resetovací brány. Tyto brány propagují informace abstrahované z aktuálního běhu pouze za podmínky určené naučenými vahami a za jiné podmínky dokáží část historických informací „zapomenout“.

Technicky výpočet probíhá následovně:

Vstupem do modulu jsou dvě proměnné h_{t-1} a x_1. Obě proměnné se v prvním kroku upraví o váhy a vloží do sigmoidní funkce, čímž vzniká nová proměnná r_t. Proměnná obsahuje hodnotu v intervalu <0,1> a reprezentuje tedy pravděpodobnost uchování, nebo zapomenutí historické informace. Tento výpočet je nazýván resetovací branou nebo branou zapomenutí - na obrázku 32 je tento výpočet vyobrazen pod číslem 1. Hodnota se následně sloučí s hodnotou původní h_{t-1} pomocí skalárního součinu (na obrázku pod číslem 2).

$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r) \tag{55}$$

$$\hat{r}_t = r_t \odot h_{t-1} \tag{56}$$

* $\odot$ reprezentuje skalární součin vektorů

V druhém kroku se vypočítává aktualizační brána, u které je výpočet naprosto stejný s branou resetovací, avšak má své vlastní naučené váhy, tedy výsledek bude jiný než v před-

chozím kroku. Tento výsledek se označuje jako z a na obrázku je reprezentován číslem 3.

$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z) \tag{57}$$

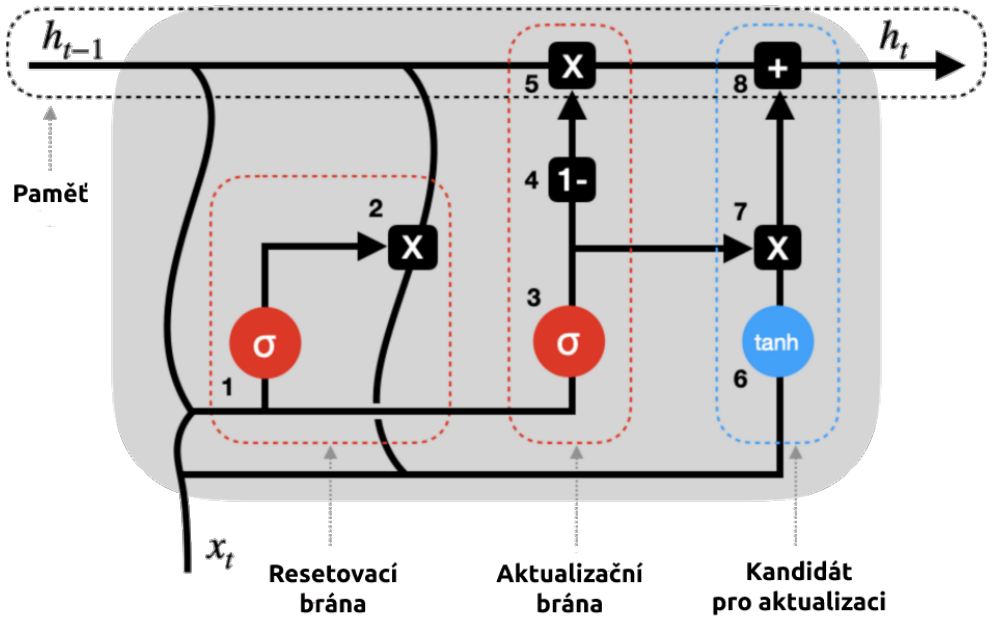
Ve třetím kroku je použita vypočítaná proměnná $\hat{r}_t$, která reprezentuje částečně zapomenutou historickou informaci (h_{t-1}). Pomocí této proměnné a původního vstupu x_t je vypočítán takzvaný kandidát aktualizace, označovaný jako $\hat{h}_t$ aktivační funkcí hyperbolického tangentu (tanh). (obr 32 bod 6)

$$\hat{h}_t = \phi_h(W_h x_t + U_h \hat{r}_t + b_h) \tag{58}$$

V závěru výpočtu je znovu pomocí skalárního součinu vynásoben kandidát aktualizace ($\hat{h}_t$) s výsledkem z aktualizační brány (z_t) do proměnné h_t. Tímto výpočtem je vzniklá hodnota $\hat{h}_t$ (obsahující část původní informace z (h_{t-1}) a část z aktuální informace (x_t)) upravena o její relevanci vůči celé časové řadě - resp. upravena o poměr, jak moc si modul „myslí“, že je aktuální informace důležitá pro budoucí rozhodování.

$$h_t = \hat{h}_t \odot z_t + (1 - z_t) \odot h_{t-1} \tag{59}$$

V rovnici výše je možné vidět i normalizační část $(1 - z_t) \odot h_{t-1}$. Tato část je na obrázku označena čísly 4 a 5, a slouží k normalizaci hodnoty do stejné úrovně, v jaké byla při vstupu. Jinými slovy v kroku tři je ve výpočtu spojena hodnota vstupu x_t s částí historické informace h_{t-1}. Tímto způsobem, při sečtení s původní hodnotou, by se historická informace kumulovala. Pro odstranění tohoto vlivu je z původní paměti buňky odebrán koeficient aktualizace a po sečtení hodnoty původní a vypočtené h_t, hodnota nová nenarůstá[43].



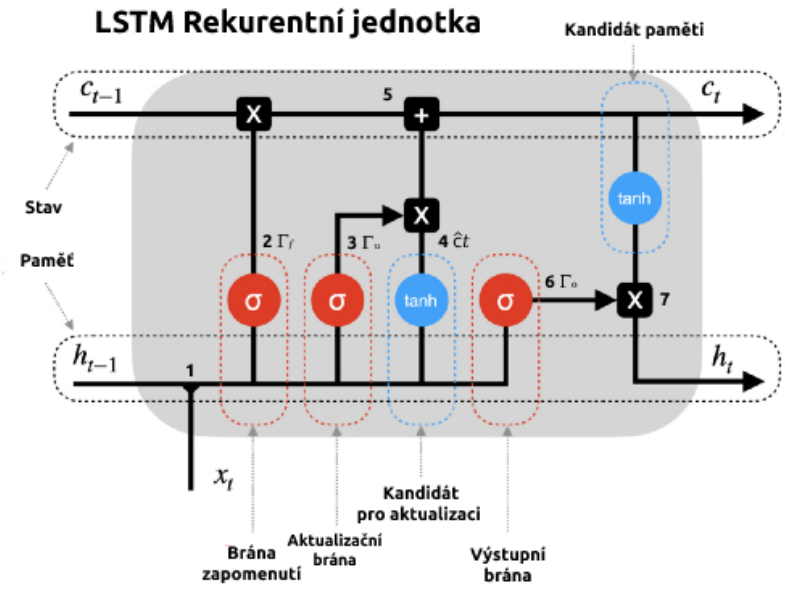
Obrázek 32: GRU modul (přeloženo z: [25])

Výstupem z tohoto modulu je pouze hodnota h_t. Avšak ta může být, například pomocí softmax funkce, převedena na výstupní hodnotu z celé vrstvy, tedy výsledek sítě.

Obrovskou výhodou buňky GRU je její možnost se naučit váhy podle kterých aktualizuje, zapomíná, ale i vypočítává nové hodnoty. Právě i kvůli tomu, že tato buňka koriguje míru aktualizace, je při zpětné propagaci gradient štěpen do jednotlivých buněk pouze relativním dílem a nerozprostírá se do všech buněk jako v případě klasických RNN, kde dochází k mizení gradientu.

7.10.2 LSTM

, neboli *Long-Short Term Memory unit* je velmi podobná, avšak složitější jednotka než GRU.



Obrázek 33: LSTM modul (přeloženo z: [26])

Místo jedné proměnné reprezentující historickou informaci lze u LSTM nalézt proměnné dvě: takzvaný „stav“ (cell state - c) a „paměť“ (hidden state - h). Stav je velmi často přirovnáván k dlouhodobé paměti buňky, tedy uchovává méně informací, ale dlouho. Zatímco paměť buňky je častěji aktualizována a reprezentuje tudíž krátkodobou paměť.

Změnou oproti výpočtu v modulu GRU je již samotný počátek výpočtu, čímž je spojení vstupní hodnoty x_t s pamětí vrstvy h_{t-1} zřetězením hodnot.

Výpočet v LSTM je velmi podobný výpočtu v modulu GRU, avšak aktualizace je počítána oproti stavu buňky (dlouhodobé paměti). Prvním krokem je výpočet koeficientu zapomenutí, často znázorňován Γ_f - na obrazovém schématu 33 pod číslem 2:

$$\Gamma_f = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{60}$$

Následně je, stejně jako u GRU, vypočtena pravděpodobnost/koeficient aktualizace pod značením Γ_u, ve schématu jako číslo 3.

$$\Gamma_u = \sigma(W_u x_t + U_u h_{t-1} + b_u) \tag{61}$$

Ve výpočtu kandidátní hodnoty je však změna, zatímco v GRU byla vypočítávána z kombinace původní hodnoty a vstupu, v případě LSTM je vypočítávána z kombinace

paměti a stavu. Znovu však je nutné připomenout, že tento výpočet je prováděn pro aktualizaci stavu (dlouhodobé paměti - c_t).

$$\hat{c}_t = \phi(W_h x_t + U_h h_{t-1} + b_h) \tag{62}$$

V dalším kroku je výpočet nového, upraveného stavu modulu. Tento stav se vypočítá jako skalární součin původního stavu c_{t-1} a proměnné Γ_f, tedy koeficientu, který udává pravděpodobnost zapomenutí části, nebo celé hodnoty původního stavu. Druhá část výpočtu je skalární součin koeficientu aktualizace (Γ_u) a kandidátu aktualizace ($\hat{c}_t$). Tyto dva výsledky se následně sečtou dohromady. Praktická funkce tohoto výpočtu je nejprve očištění původního stavu od hodnot, které modul vyhodnotí jako nadbytečné a následné přidání hodnot, které považoval (jak ze vstupů, tak i z krátkodobé paměti) za důležité. Nová hodnota c_t se již rovná finální hodnotě dlouhodobé paměti - stavu modulu.

$$c_t = c_{t-1} \odot \Gamma_f + \Gamma_u \odot \hat{c}_t \tag{63}$$

Druhou fází v buňce je výpočet nové hodnoty paměti (krátkodobé paměti). Tento výpočet začíná stejně jako u ostatních, tedy výpočtem výstupní brány. Tato hodnota se znovu vypočítává ze vstupní hodnoty x_t a paměti h_{t-1}.

$$\Gamma_o = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{64}$$

Výpočet konečné hodnoty h_t je následně tvořen aplikací tanh funkce na stav (c_t) a následným skalárním součinem s koeficientem Γ_o. Jak vzorec napovídá, krátkodobou paměť vlastně tvoří pouze částečná informace tvořená aktualizovaným stavem z předchozího pozorování.

$$h_t = \phi(c_t) \odot \Gamma_o \tag{65}$$

Stejně jako u modelu GRU je v případě potřeby výstupu z modulu možnost využít nové hodnoty h_t a pomocí například sofmax nebo sigmoid funkce vypočítat výstupní hodnotu z modelu $\hat{y}$[19].

Pro doplnění, při implementaci se často vstup x_t a h_{t-1} zřetězí, čímž dovolí pouze jediné násobení s váhami pro každou bránu. Váhy se tedy upravují pro obě proměnné zároveň.

Model LSTM je opravdu komplexní model, který lze následně velmi kreativně upravovat. Právě kvůli tomu je však výkonnostně náročný. Byl představen již v roce 1997 Seppem Hochreiterem a Jürgenem Schmidhuberem. I přesto, že byl v této práci nejdříve představen GRU model, LSTM byl vytvořen historicky první. Kvůli komplexitě je nutné s LSTM více experimentovat a ladit hyperparametry, proto byl z LSTM vyvinut právě GRU model, tedy zjednodušená verze. GRU je velmi často aplikován na jednodušší úlohy, které, stejně jako složitější a náročnější model LSTM, přesvědčivě zvládá, ale s mnohem menší náročností[19].

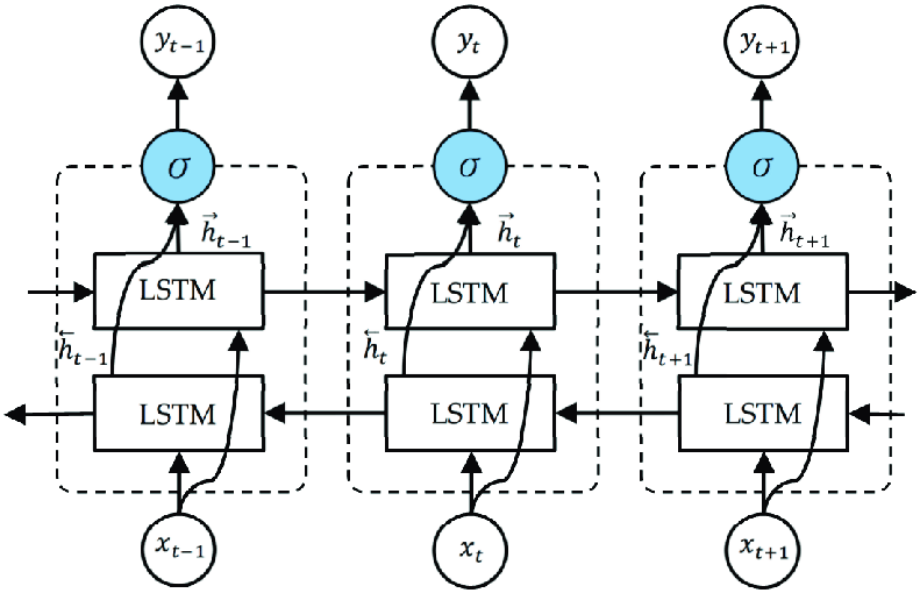
7.10.3 Obousměrné modely RNN:

Při zpracování přirozeného jazyka, je do rekurentních modelu vkládáno jedno slovo po druhém, pokaždé si model vytváří lepší povědomí o sentimentu, nebo o kontextu určité věty. Některé informace dostupné na konci věty by však mohly být prospěšné i na začátku věty. Kvůli tomuto problému se vytvořil takzvaný model obousměrné propagace. Tento model, respektive spíše vrstva, je tvořen páry rekurentních neuronů - jeden, pro klasický dopředný krok, který svůj výstup zasílá dalšímu v pořadí a druhý zpětný, který přijímá výstup posledního dopředného neuronu, provede výpočet a zasílá proti směru původního proudu do

dalšího zpětného neuronu. Takto se zpropagují všechny informace ke každé dvojici rekurentních neuronů. Výstupy z obou neuronů se následně spojí za pomoci doplněných vah i biasů.

$$\hat{y}_t = \sigma(W_y[h_{forward}, h_{backward}] + b_y) \tag{66}$$

Ačkoliv jsou tyto vrstvy využívány hlavně pro zpracování přirozeného jazyka, velmi dobře fungují též pro učení časových řad. Přestože nedokáží profitovat z informací v přesný moment výstupu predikce, jsou obousměrné modely následovány mělkou neuronovou sítí, která díky zpětné informaci ve fázi trénování dokáže lépe upravit své váhy. Obecně lze říci, že použitím obousměrné varianty LSTM se predikce může, ale nemusí zlepšit. Ve většině případů se nezhorší, pouze přidá na komplexnosti.



Obrázek 34: Obousměrný modul LSTM [27]

7.10.4 Další techniky používané v RNN

Rekurentní neuronové sítě jsou kvůli své variabilitě velmi často kombinovány s jinými metodami, ať již z klasických neuronových sítí, ale také ze sítí konvolučních. Díky tomu vzniklo několik, dnes již ustálených technik, které jsou použitelné v široké paletě různých problémů. Tyto techniky jsou aplikovány především na rozpoznávání přirozeného jazyka, mluveného slova, nebo pro generování a transformaci hudby.

Jmenovitě jsou těmito technikami například Attention , Self-Attention a Multi-Head Attention, Transformery a pokročilé spojené encoder-decoder modely vycházející z many-to-many struktury[45].

8 Praktická část

V této části práce jsou vytvořeny modely využívající techniky představené v části teoretické. Každý model je otestován pomocí metrik využívaných ve strojovém učení. Následně jsou vytvořené modely použity pro virtuální obchodování za účelem otestování vhodnosti a ziskovosti těchto modelů na konkrétních instrumentech (Apple Inc., JPMorgan Chase & Co., Amazon.com, Inc., ETF S&P500, All-World ETF, Bitcoin, Ethereum), čímž je možné zhodnotit hypotézy této práce.

8.1 Technické prostředí

Pro veškeré výpočty byl použit notebook Lenovo ThinkPad P15 gen 2i obsahující jako procesor Intel i7-11800H (2.3GHzx16), 82 GB pamětí RAM a grafickou kartu NVIDIA Quadro T1200 s 4GB vlastní paměti. Trénování modelů probíhalo převážně na grafické kartě (kromě modelu ARIMA - viz níže). Jiné prostředky než grafická karta však vytíženy při trénování nebyly, tedy je možné pro připravený notebook využít i prostředí Google Colaboratory[46], či Deepnote[47].

8.1.1 Použité knihovny a prostředí

Celá aplikační část práce je psaná v jazyce Python[48], který je nejspíše nejrozšířenějším jazykem pro práci s neuronovými sítěmi, následovaný například jazykem R[49].

K implementaci řešení bylo použito několik knihoven.

Pro operativu s nastavením a ukládáním dat byla užita knihovna ConfigParser[50]. Pro vizualizace dat a křivky učení matplotlib[51] a livelossplot[52]. Dále pro stažení potřebných datasetů knihovny binance[53] a yfinance[54]. A pro samotné výpočty - v případě modelu ARIMA statsmodels(v. 0.13.2)[55] a pro neuronové sítě Tensorflow[56], respektive rozšíření Keras[57] ve verzích 2.8.0. Kvůli snadnějšímu hledání hyperparametrů byla použita i knihovna Optuna[58].

8.2 Struktura praktické aplikace

Praktická aplikace této práce je rozdělena do dvou částí, první je Jupyter notebook, ve kterém jsou s popisem představeny jednotlivé kroky stažení dat, vytváření modelů i část práce s daty. V tomto notebooku jsou však použity vlastní python funkce, které model naplňují, trénují a dovolují udržet notebook spíše informační než technický. Druhá část práce je v python skript, který využívá podobné funkce jako notebook, ale testuje všechny datové sady, na rozdíl od notebooku, který je prováděn pouze na vybraném instrumentu. Toto rozdělení je provedeno hlavně pro možnost experimentace s modely v Jupyter notebooku, aniž by se musely upravovat a spouštět celé testovací skripty, které trvají velmi dlouho (kolem 20 hodin).

8.3 Data

Data byla použita ze tří zdrojů: Yahoo Finance[59], Binance[60] a Alpha Vantage[61].

První zmíněný zdroj byl využit pro data denní, protože Yahoo Finance (na rozdíl od Alpha Vantage) poskytuje data akcií očištěná od dělení (stock split), tedy jsou použitelná pro učení bez dalších úprav. Yahoo Finance však poskytuje nejkratší datové úseky jednoho

dne, proto byla v praktické části použita i data ze zdrojů Alpha Vantage a Binance, která poskytují data hodinová, třicetiminutová i minutová a to několik měsíců zpětně.

8.4 Testovací metody

Testování učených modelů bylo v první řadě prováděno pomocí střední kvadratické chyby 8.4.3, kterou měl každý model za úkol minimalizovat. Po učení, byla pro testovací data zároveň změřena i hodnota chyb RMSE, MAE a R^2 viz níže, které vyjadřují správnost modelu a schopnost vysvětlení časové řady modelem.

Druhá fáze testování porovnává výkonnost obchodování modelu s celkovým růstem nástroje. Jinými slovy, zda se vyplatí použití modelu a aktivní obchodování, nebo naopak, tedy že model nemá dostačující predikční schopnosti a je lepší pouze nakoupit a nechat instrument sám stoupat, či klesat bez zásahu. Toto porovnání bylo použito především dle reálných pozorování výkonnosti modelů, které budou dále představeny. Pokud by na trhu modely performovaly lépe, bylo by také možné využít porovnání se speciálními obchodovacími technikami, které však v této práci není třeba představovat.

8.4.1 Bodová předpověď

Bodová předpověď, která je v praktické části sestrojena, představuje odhad hodnoty v časové řadě v určitém budoucím okamžiku. Chyba předpovědi e_t v čase t je definována funkcí[43]:

$$e_t = Y_t - \check{Y}(t-1) \tag{67}$$

, kde

- Y označuje skutečnou hodnotu v čase t
- $\check{Y}(t-1)$ označuje predikovanou hodnotu v čase bezprostředně předcházejícím t

8.4.2 MAE

MAE – Mean Absolute Error neboli střední absolutní chyba je nejjednodušší metrika chyby predikce, která vypočítává absolutní chybu v předpovězené hodnotě vzhledem k reálné hodnotě a je definována následujícím vzorcem[43]:

$$MAE = \frac{1}{N}\sum_{t=1}^{n}|e_t| \tag{68}$$

8.4.3 MSE

Alternativou k metrice MAE je metrika MSE (mean squared error - střední kvadratická chyba). MSE počítá odchylku predikce a reálné hodnoty v druhé mocnině, čímž zvětší velmi vzdálené hodnoty predikce, ale naruší škálu hodnot a proto výsledky z této metriky nejsou jednoduše reprezentovatelné. Tato metrika je velmi často používána pro rekurentní sítě jako hodnota, kterou optimizér optimalizuje[43].

$$MSE = \frac{1}{N}\sum_{t=1}^{n}e_t^2 \tag{69}$$

8.4.4 RMSE

Nadstavbou metriky MSE je alternativa RMSE, která výpočet obohacuje o odmocninu. Tato metrika znovu vrací hodnotu do škály původních dat a je tedy reprezentovatelná vůči samotným hodnotám dat[43].

$$RMSE = \sqrt{\frac{1}{N}\sum_{t=1}^{n} e_t^2} \tag{70}$$

8.4.5 Koeficient determinace

Vyjadřuje, jak velkou část variability proměnných se modelu podařilo vysvětlit. Nejčastěji se uvádí v procentech.

Pro výpočet je nejprve nutné vypočítat rozptyl chyby, tedy rozptyl rozdílů mezi předpovězenými hodnotami a skutečností[34]:

$$S_r = \sum_{i=1}^{n} e^2 \tag{71}$$

Dále je nutné vypočítat rozptyl vzorků v celé testovací množině.

$$S_e = \sum_{i=1}^{n} (Y_i - \bar{Y}_t)^2 \tag{72}$$

Z těchto dvou proměnných se následně vytvoří koeficient:

$$R^2 = 1 - \frac{S_e}{S_r} \tag{73}$$

8.4.6 Porovnání výdělečnosti

Jak bylo zmíněno výše, samotné metriky chybovosti hodnot nezaručují modelu vlastnost predikovat pohyby. Je proto možné, že model i přes skvělé hodnoty metrik nemusí v obchodním světě uspět, protože absolutní chyba mezi hodnotami predikovanými a reálnými sice může být malá, ale obchodování je závislé na stejném směru růstu / poklesu. Proto tedy model i přes malou chybovost nemusí předpovídat tyto změny a bude při obchodování nepoužitelný.

Výsledek hospodaření modelu

Pro porovnání výdělečnosti byly zaimplementovány dva mechanizmy. Prvním z nich je výsledek hospodaření daného modelu. Tento mechanizmus předpokládá prvotní kapitál jednoho milionu korun. Model rozhoduje o dalším kroku instrumentu (v denních grafech den, v hodinových hodina, ...) tak, že: mechanizmus registruje „nákup“ pokud model předpověděl kladný výsledek intervalu, nebo „prodej“ (shortování) pokud záporný. Na konci daného intervalu (například dne) mechanizmus pozici zruší, tedy instrument prodá, respektive nakoupí zpět. V tomto mechanizmu je abstrahováno od spreadu i poplatků burz, pro zjednodušení a možnosti porovnávání.

Porovnání následně probíhá mezi výsledkem hospodaření na testovacích datech oproti případu, kdy by investor na začátku testovacího období nakoupil a na konci prodal, tedy včetně různých pohybů instrumentu.

Hodnoty výsledku hospodaření je možné porovnávat pouze mezi stejnými typy instrumentů a stejným modelem (pouze kryptoměny BTC, ETH a pouze modely základního LSTM například). Je tomu tak z toho důvodu, protože jsou instrumenty různě obchodované - kryptoměny 24 hodin denně, akcie pouze část dne a též, jak je prezentováno v dalších kapitolách, nelze do všech modelů z různých důvodů vkládat plný datový set. Data jsou tedy při každém testování různého modelu jiná - neporovnatelná.

Teoretická ziskovost modelu Aby byly výsledky modelů porovnatelné mezi datovými množinami i mezi modely, bylo nutné výsledky z výše uvedeného mechanizmu abstrahovat do výpočtu teoretické ziskovosti za jednotné období.

Tato ziskovost se vypočítává jako profit každého kroku modelu, který se násobí hodnotou odpovídající předem stanovenému intervalu (jednomu týdnu). Jinými slovy, výsledek předešlého hospodaření byl vyčíslen na jeden milion korun plus to, co model vydělal, případně prodělal. Tento výsledek byl následně zprůměrován na jednu předpověď/jeden krok (den, hodinu, minutu).

$$profit\_per\_step = \frac{balance - 1.000.000}{length\_of\_test} \tag{74}$$

Pro každou datovou sadu, která obsahovala denní, hodinová, půlhodinová nebo minutová data byla vypočtena délka kroků vedoucích do univerzálního intervalu jednoho týdne. V tomto výpočtu teoretické hodnoty je předpokládán 24 hodinový obchodní den, sedm dnů v týdnu, tedy pro denní data byla hodnota násobena sedmi, pro hodinová 168 (7*24), pro půlhodinová 336 (7*24*2), pro minutová 10080 (7*24*60). Tímto výpočtem lze dosáhnout teoretické porovnatelné hodnoty, kterou by model vydělal za toto období.

8.5 Trénování modelu ARIMA

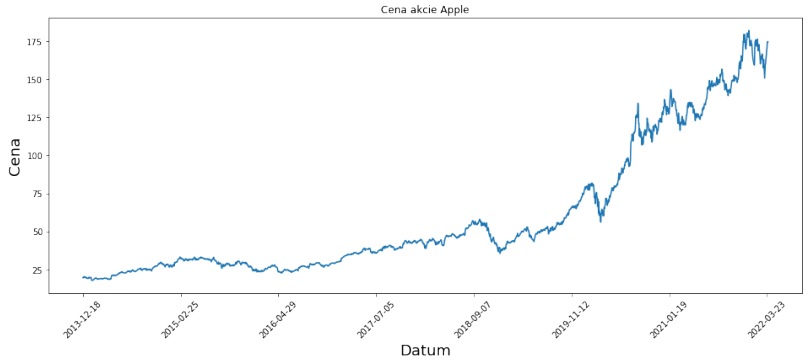
Model ARIMA, jak je již naznačeno v teoretické části práce, je velmi odlišný od modelů neuronových sítí. Nejen, že staví na naprosto odlišných způsobech výpočtu, ale též potřebuje pro své výpočty jiný formát dat, proto je tento model od modelů neuronových sítí oddělen metodami a výpočty. Avšak metriky jsou vytvořeny stejné jako pro neuronové sítě, tedy výsledky jsou naprosto porovnatelné.

Normalizace a příprava dat: Jedním z velkých rozdílů mezi modely neuronových sítí a modelem ARIMA je náročnost na data. Zatímco neuronové sítě potřebují ke konvergenci opravdu velké množství dat, u modelu ARIMA tomu tak převážně není. Při experimentech na denních datech společnosti Apple dosáhl model podobných výsledků při evaluaci na 20 i 40 % celého datasetu (u některých datasetů i lepších - přetrénování), proto bylo v dalším zkoumání použito pouze 20 % datových bodů (2083 dnů). Hlavním důvodem tohoto snížení je výpočetní náročnost. Protože model ARIMA (respektive jeho implementace v knihovně statsmodel) využívá pouze procesor, evaluace je výrazně časově náročnější než u neuronových sítí i na menším datovém setu. Například v případě čtyř tisíc vzorků je model naučen s výše uvedeným HW za necelých pět minut, zatímco u dvanácti tisíc vzorků je to již přes pětadvacet minut na jedné datové množině.

Po omezení množiny dat je nutné data takzvaně normalizovat. Jak již bylo popsáno v kapitole 7.9.1, je velmi rozšířenou praxí data přetransformovat do stejného rozsahu, pro ulehčení trénování modelu. Pro tuto normalizaci se v implementaci použil MinMaxScaler z knihovny sklearn.

```
scaler = MinMaxScaler(feature_range=(-1, 1))
df_close = scaler.fit_transform(np.array(df_close).reshape(-1, 1))
```

Určení parametrů: firma Apple je jedna z firem označovaných jako růstové, díky stále zvyšující se hodnotě akcie, tím pádem je její cena více volatilní a zároveň za poslední roky stoupá. Díky tomuto faktu a pohledem na graf (obr.6 (1)) lze bezpečně prohlásit, že časová řada stacionární není - zvětšování středních hodnot a viditelný trend.

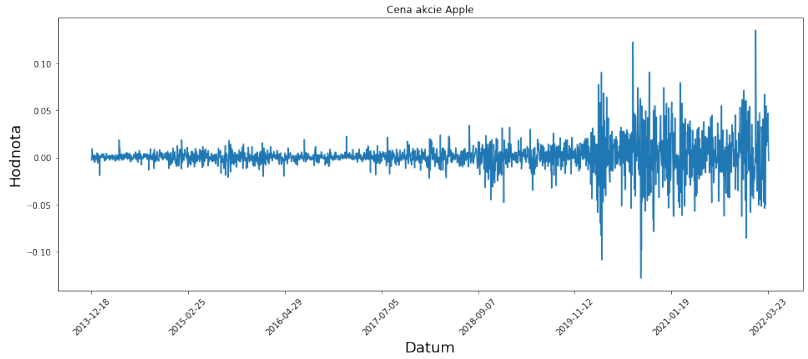


Obrázek 35: Graf akcie Apple [vlastní zpracování]

Pro odstranění stacionarity lze použít knihovnu NumPy a jednoduchou diferenci datasetu pomocí funkce diff():

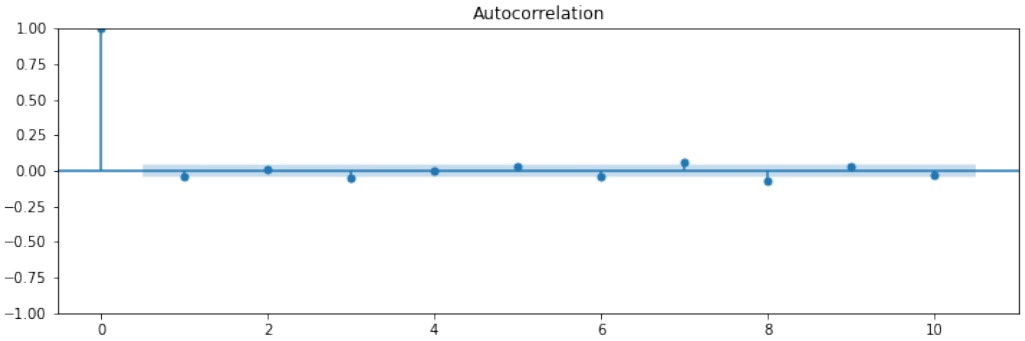
```
import numpy as np
ts = np.diff(np.array(df_close[:, 0]))
```

Dataset po exekuci tohoto kódu je zobrazen na obrázku 36.

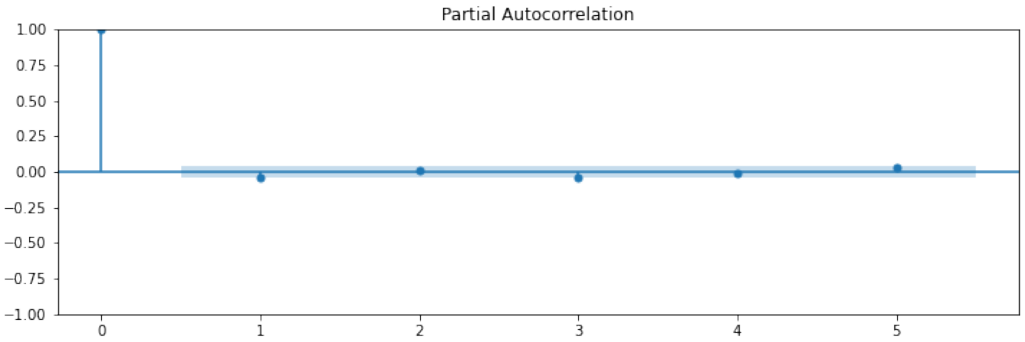


Obrázek 36: Datová množina po aplikaci diferenční transformace [vlastní zpracování]

Výsledná predikce bude záviset na zvolených parametrech p, d, q. Pro jejich zvolení je nutné zkonstruovat autokorelační funkci (ACF) a funkci parciálně-autokorelační (PACF) 5.4.1. Hodnota parametru se musí rovnat hodnotě funkce, kdy poprvé protne horní hranici intervalu spolehlivosti.



Obrázek 37: Graf ACF zvolené datové množiny [vlastní zpracování]



Obrázek 38: Graf PACF zvolené datové množiny [vlastní zpracování]

V tomto případě z obou funkcí vyšla hodnota jedna.

Testovací množina: pro další operace s daty bylo nutné dataset rozdělit na dvě skupiny, jednu trénovací, na které se model bude učit, resp. skupinu, kterou bude model analyzovat. Na druhé skupině (testovací) se model po implementaci otestuje. Více o tomto rozdělení je popsáno v kapitole 6.4.

```
train_size = int(len(df_close) * 0.80)
train, test = df_close[0:train_size], df_close[train_size:len(df_close)]
```

8.5.1 Vytvoření ARIMA modelu a predikce

Vytvoření po přípravě dat a zjištění parametrů je již velmi jednoduché. Pro každé testovací pozorování je vytvořen vlastní model s historií do dané chvíle predikce. Z modelu je následně vyžádána následující predikce, ta je uložena do seznamu predikcí a reálná hodnota vložena do seznamu historie pro predikci dalšího pozorování.

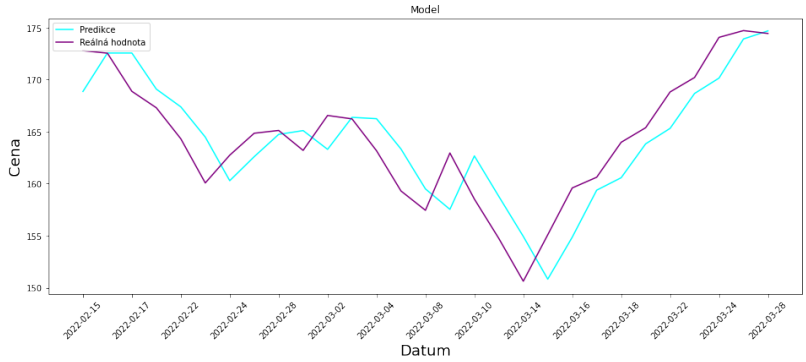
```
for t in range(len(test)):
    model = ARIMA(train, order=(1, 1, 0))
    model_fit = model.fit()
    predictions.append(model_fit.forecast()[0])
    train.append(test[t])
```

8.5.2 Evaluace a výsledek modelu

Po celém predikčním procesu jsou v proměnné *predictions* hodnoty v rozsahu <0,1>, protože byla použita normalizace. Hodnoty lze jednoduše převést na původní interval zavoláním metody *inverse_transform* na scaler objektu, který data původně normoval:

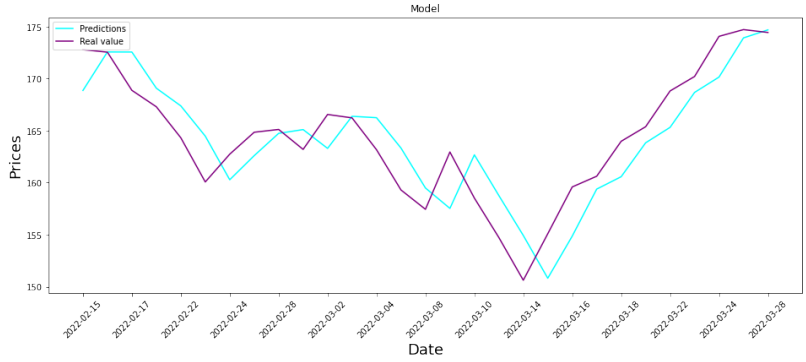
```
predictions_denormalized = scaler.inverse_transform(predictions)
```

Po denormalizaci lze data například vyjádřit grafem:



Obrázek 39: Predikce vs. reálné hodnoty akcie [vlastní zpracování]

Pro porovnání s trénováním na 40% množině:



Obrázek 40: Predikce vs. reálné hodnoty akcie se 40% dat [vlastní zpracování]

Jak je možné vidět, grafy se liší skoro nerozeznatelně. Zároveň je však vidět, že model hodnoty příliš neupravuje a používá převážně hodnotu předešlého pozorování (určitá změna oproti předešlému pozorování však je).

Hodnoty metrik následně vychází:

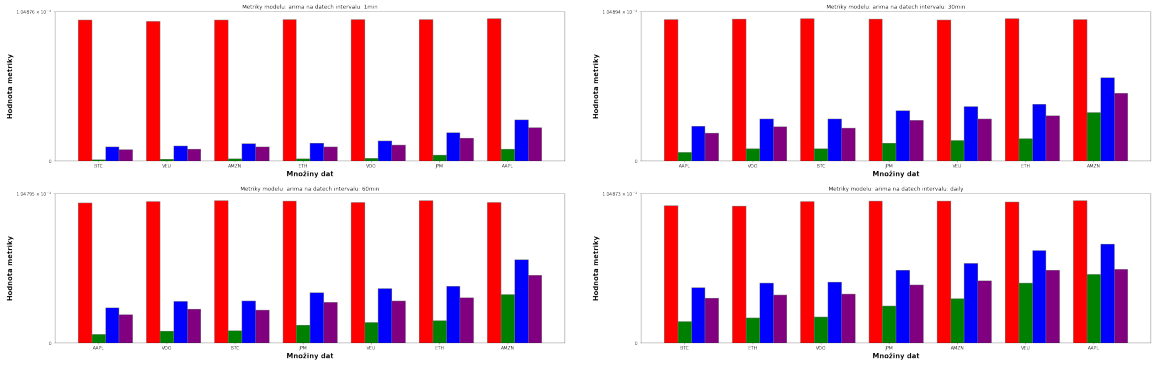
Metrika	Hodnota
R^2	0.982
MSE	0.001102
MAE	0.002669
RMSE	0.003319

Tabulka 1: Hodnoty metrik

Z hodnot výše lze vyčíst, že si model nevedl vůbec špatně, skóre 0.982 na metrice determinačního koeficientu značí, že dokázal vysvětlit velkou část časové řady. V tomto případě, kdy má časová řada velmi malé výkyvy, je však tento údaj velmi klamavý, protože i když ARIMA zvládla svou práci velmi dobře, v tomto měřítku to není dostatečné.

8.5.3 Výsledky na dalších datových množinách

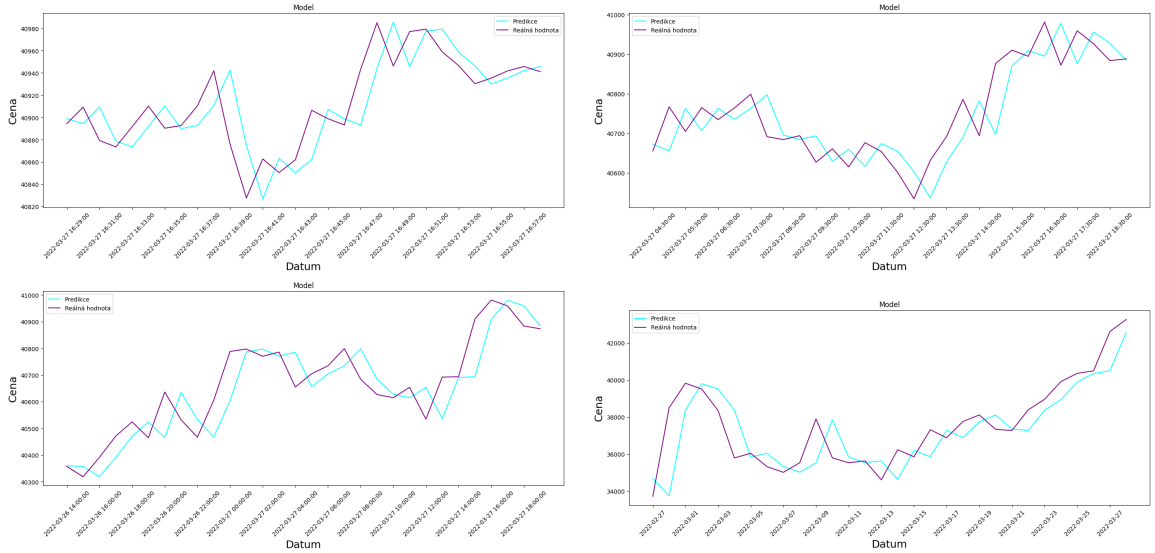
Výsledky výše jsou uvedeny pouze pro jednu datovou sadu. Tyto výsledky však nezaručují účinnost modelu na jiných datových sadách a už vůbec ne na jiném typu dat (například z denních grafů na minutové). Proto byl tento model testován na všech dostupných datových sadách s následujícími výsledky:



Obrázek 41: Metriky modelu ARIMA na minutových, 30 minutových (vpravo nahoře), hodinových (vlevo dole) a denních (vpravo dole) datových sadách [vlastní zpracování]

Z těchto grafů je jasně viditelné, že model byl schopný všechny testované datové sady z větší části vysvětlit a jeho chyby nejsou nijak velké.

Ovšem, jak bylo zmíněno výše, tyto metriky nemusí být vypovídající. Je nutný dále pohled na rozdíly predikcí a reálných hodnot například posledních 30 pozorování v testovacích datech:



Obrázek 42: Rozdíl predikcí a reálných hodnot na minutových, půlhodinových (vpravo nahoře), hodinových (vlevo dole) a denních (vpravo dole) datových sadách ARIMA modelu [vlastní zpracování]

Jak je z porovnání hodnot viditelné, u těchto předpovědí, stejně jako u vytváření modelu, lze rozeznat takzvané lagování hodnoty, kdy model velkou měrou kopíruje data z předchozího pozorování.

Pomocí postupu fiktivního obchodování popsaného v kapitole 8.4.6 lze tento model zhodnotit i po stránce použitelnosti v obchodním prostředí. Z tohoto testování vychází modely na konkrétních datových množinách dle tabulek níže. Sloupce model a instrument ukazují závěrečný stav účtu po obchodování modelu a instrumentu, respektive situaci, kdy by investor na konci období instrument prodal. Další dva sloupce ukazují průměrný zisk na jeden obchod a normalizovaný zisk, který by teoreticky nastal po 7 dnech obchodování na pokyn daným modelem.

datový set	model	instrument	průměrný obchod	normalizovaný profit
AAPL	999885	1000589	-0.0353	-356.24
AMZN	999871	1000249	-0.0677	-682.58
BTC	999908	1000082	-0.0393	-396.14
ETH	999536	1000200	-0.198	-1997.06
JPM	999238	1000234	-0.380	-3830.9
VEU	1000061	999955	0.05	402.67
VOO	1000164	1000090	0.086	871.44

Tabulka 2: Bilance obchodování pomocí predikcí modelu ARIMA - minutová data

datový set	model	instrument	průměrný obchod	normalizovaný profit
AAPL	997132	1000504	-1.127	-378.79
AMZN	998148	999640	-0.848	-284.92
BTC	1000556	998126	0.089	29.92
ETH	1003111	998565	0.498	167.4
JPM	998467	998919	-0.681	-228.72
VEU	998888	999702	-0.975	-327.75
VOO	1000785	999981	0.338	113.64

Tabulka 3: Bilance obchodování pomocí predikcí modelu - půlhodinová data

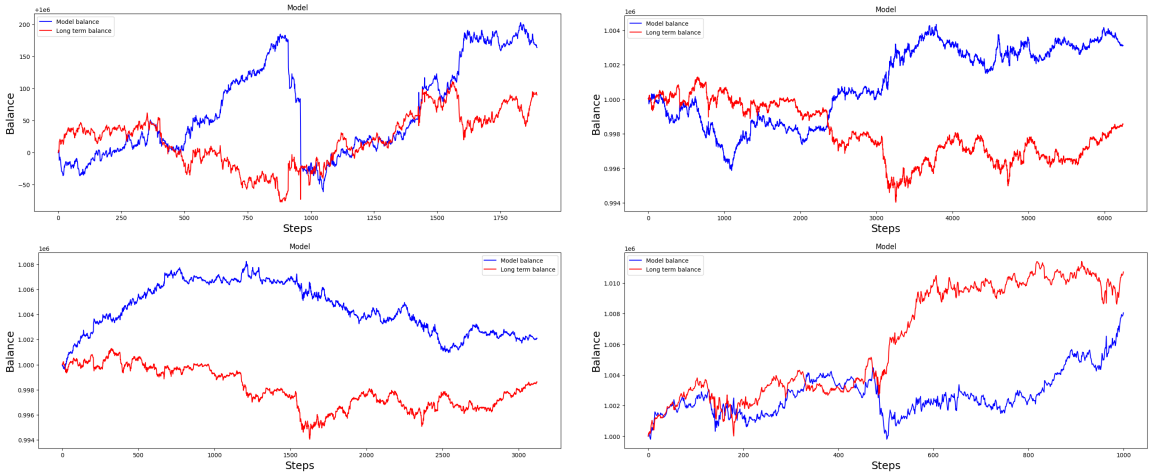
datový set	model	instrument	průměrný obchod	normalizovaný profit
AAPL	998954	1000526	-0.821681	-138.04
AMZN	996013	999687	-3.413527	-573.47
BTC	1001816	998142	0.581492	97.69
ETH	1002102	998618	0.673071	113.08
JPM	996191	998893	-3.195470	-536.84
VEU	999342	999710	-1.061290	-178.3
VOO	997503	1000039	-2.035045	-341.89

Tabulka 4: Bilance obchodování pomocí predikcí modelu - hodinová data

datový set	model	instrument	průměrný obchod	normalizovaný profit
AAPL	998474	1020656	-0.915966	-6.41
AMZN	1008056	1010728	8.039920	56.28
BTC	986412	1006924	-30.881818	-216.17
ETH	999275	1008725	-2.832031	-19.82
JPM	992715	1010117	-4.295401	-30.07
VEU	996235	1001653	-6.202636	-43.42
VOO	1000245	1004467	0.525751	3.68

Tabulka 5: Bilance obchodování pomocí predikcí modelu - denní data

Nejlépe vychází obchodování modelu instrumentu S&P500 (VOO) u minutových dat, kryptoměny ETH na půlhodinových i hodinových grafech a akcie Amazon u dat denních. Toto obchodování lze zobrazit grafy:



Obrázek 43: Obchodování dle predikcí modelu u nejlepších výkonností - vlevo nahoře 1minutová data VOO, vpravo nahoře půlhodinová data ETH, vlevo dole hodinová data ETH a vpravo dole denní data AMZN [vlastní zpracování]

Nutno dodat, že denní data akcií Amazon, by bez obchodování za pomoci modelu, vydělala více než za použití modelu. Tedy i přes kladný výsledek hospodaření je model vlastně špatný.

Z grafů je též možné vyčíst přesné vzory, které model dokázal předpovědět a na kterých tudíž „vydělal“ největší zisk. Obecně však, vzhledem k výsledkům, není možné s jistotou model prohlásit za použitelný.

8.6 Obecný postup pro aplikaci neuronových sítí

Pro neuronové sítě lze obecně specifikovat jednotný, či velmi podobný způsob přípravy dat. Prvním krokem je, stejně jako u modelu ARIMA, transformace dat do stejného rozsahu od 0 do 1 pomocí MinMaxScaler funkce (popsáno výše 8.5). Následně je však nutné tyto data připravit odlišným způsobem než pro ARIMU a to tak, že každé jedno pozorování bude obsahovat skupinu pozorování předešlých, které to aktuální mají popsat. Základní počet těchto stínových pozorování je nastaven na 60 a uložen v proměnné *length_of_sample*. Transformace dat na tento tvar probíhá následovně:

```
x = []
y = []
for i in range(length_of_sample, len(data)):
    x.append(data[i - length_of_sample:i])
    y.append(close_data[i])
```

V tento moment by již data měla být připravena a lze přejít k přípravě samotného modelu. Aniž by nyní bylo nutné znát strukturu jednotlivých modelů, či o jaký model se jedná, obecně je zde proces vždy stejný. Nejprve se model zkompiluje. Kompilace modelu znamená, že mu je nastavena chybová funkce, optimizér a metriky.

V případě této práce se tedy dle výše uvedených kritérií kompiluje následujícím kódem:

```
model.compile(
    optimizer=opt,
```

```
    loss='mean_squared_error',
    metrics=[
        RootMeanSquaredError(),
        MeanAbsoluteError(),
        MeanSquaredError()])
```

Jak je možno vidět, proměnná *opt* obsahuje název optimizéru použitého pro daný model. Ten je možné pro experimenty měnit, avšak ve většině případ je to výše zmíněný optimizér ADAM (7.9.2).

Po zkompilování je možná model konečně „učit“. To se provádí pomocí metody fit. Tato metoda přijímá hned několik parametrů, prvním a druhým z nich jsou trénovací data, respektive data x_train obsahující historii pozorování, jak bylo zmíněno výše a data y_train obsahující reálnou hodnotu ceny, které se má predikce blížit. Dalším z parametrů je $batch\_size$, $epoch$, $shuffle$ a $callbacks$. První dva jsou již vysvětlené v kapitole 7.7.3 a jsou objektem experimentů pro každou datovou množinu. Parametr shuffle značí možnost promíchávání trénovacích dat před trénováním modelu. Tato možnost je některými zdroji nedoporučována pro časové řady, avšak dle experimentů autora práce, zamíchávání ve všech experimentálních případech zlepšilo trénování modelu, protože model bez zamíchání průměroval trend - váhy se nejspíše postupně učily trendovou složku a výsledek byl tedy vždy hodnotami nízko pod hodnotou reálnou. Zamícháním se naopak váhy nepřizpůsobovaly trendu, ale braly v potaz pouze relevance historických hodnot a modely netrpěly problémem podhodnocení. Posledním parametrem jsou callback funkce, které dovolují do trénování modelu přiložit nějakou vlastní logiku. V případě této práce byly například využity funkce $PlotLossesKeras$ a $EarlyStopping$, které jsou zmíněny v sekci hledání parametrů 8.8.1.

```
model.fit(
    x_train,
    y_train,
    batch_size=bs,
    epochs=epochs,
    shuffle=shuffle,
    callbacks=fit_callbacks)
```

Po naučení modelu je již možné predikovat nové hodnoty. Pro tuto funkci je na modelu dostupná metoda predict, která jako vstupní parametr vyžaduje testovací data. Tyto data musí být ve stejném formátu jako jsou výše popsaná trénovací data, tedy každé pozorování musí mít přidělenou historii několika kroků.

```
predictions = model.predict(x_test_values)
```

Stejně jako u modelu ARIMA je následně nutné tyto predikce přetransformovat do původního měřítka pomocí $scaler.inverse\_transform()$.

U objektu modelu je jako poslední volání prováděno $evaluate$. Tato metoda znovu požaduje testovací data, avšak včetně reálných hodnot y. Sama následně provádí predikce a vyhodnocuje metriky, které byly uvedeny při kompilaci modelu. Výstupem z této metody je list výsledků všech metrik.

```
evaluations = model.evaluate(x_test_values, y_test_values)
```

Metriku R^2 však současná verze keras knihovny nepodporuje, proto je, za pomocí poskytnutých predikcí, vypočítávána vlastní funkcí.

8.7 Vytvoření ANN modelu

Samotný model je následně možné vytvořit velmi jednoduše. Jediné, co je v této chvíli již potřeba udělat, je definovat samotnou strukturu modelu. U modelu klasické neuronové sítě je to v případě úplného základu například takto:

```
model = Sequential()
model.add(Dense(units=128, activation='relu',
    input_dim=length_of_sample))
model.add(Dense(units=1, activation='linear'))
```

Tento kód definuje model, který obsahuje pouze jednu vstupní vrstvu používající ReLU aktivační funkci a mající 128 neuronů. Dále už pouze jednu lineární výstupní vrstvu, pro výstup jednoho čísla - predikce. Základní hodnotou mse evaluace trénovacích dat tohoto modelu na BTC denních datech je $1.4 * 10^{-5}$. To jsou velmi pěkná čísla. Takto mělká síť však například u BTC grafů nemusí pojmout všechny vzory v grafu. Jak tedy udělat síť hlubší? Lehce, přidáním dalších vrstev:

```
model = Sequential()
model.add(Dense(units=128, activation='relu',
    input_dim=length_of_sample))
model.add(Dense(units=256, activation='relu'))
model.add(Dense(units=1, activation='linear'))
```

Po trénování tohoto modelu je možné zaznamenat naprosto podobnou hodnotu mse, tedy velmi lehké zlepšení. Z těchto výsledků lze usuzovat, že trénovací data model popisuje správně a nedochází tedy k podtrénování modelu, protože se jedná o velmi malé hodnoty mse a minimální změny po zesložitění modelu. Po natrénování modelu a následné predikci je však možné zaznamenat řádový rozdíl mezi mse chybou na trénovacích a testovacích datech (testovací data - $1.2*10^{-4}$). To by v případě hlubší neuronové sítě mohlo znamenat přeučení modelu. Dle kapitoly o tomto stavu (6.4.2) je hlavním řešením aplikace regularizace. Model by po této aplikaci mohl vypadat například takto:

```
model = Sequential()
model.add(Dense(units=128,
    kernel_initializer='lecun_uniform',
    kernel_regularizer=regularizers.l1_l2(l1=1e-5, l2=1e-4),
    activation='relu', input_dim=length_of_sample))
model.add(Dropout(0.2))
model.add(Dense(units=256,
    kernel_regularizer=regularizers.l1_l2(l1=1e-5, l2=1e-4),
        activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(units=1, activation='linear'))
```

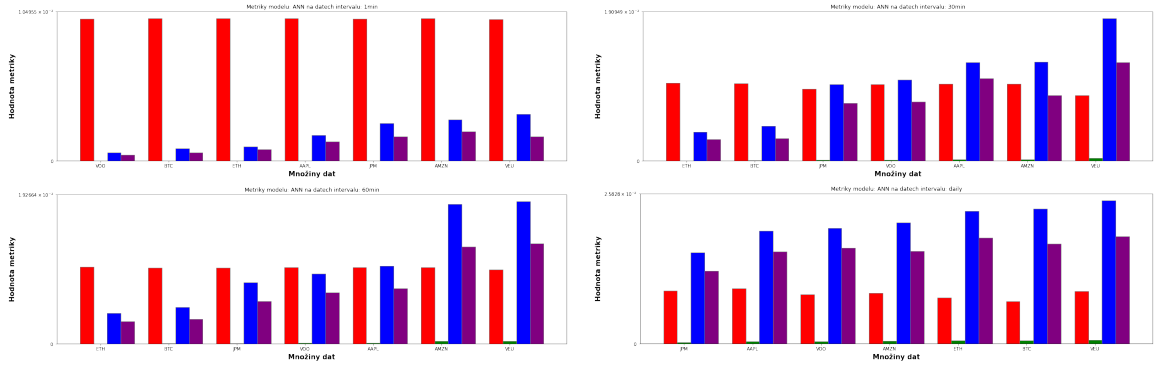
V tomto modelu je aplikována technika L1 a L2 regularizace současně s technikou Dropout. Po této aplikaci však významně vzrostla chybovost mse skóre jak na trénovacích, tak i testovacích datech z $1.4 * 10^{-5}$ na $4 * 10^{-2}$. Ani při aplikaci menších regularizací, nebo jen jedné z technik nevznikla zlepšení na testovacích datech, proto regularizace v testování nebyly použity - k přetrénování nedochází. Obecně je jeden řád ($1.4 * 10^{-5}$ -> $1.4 * 10^{-4}$)

velmi malá odchylka mezi testovacími a trénovacími daty, proto regularizace nebyla nejspíše potřeba a je zde hlavně pro demonstraci.

Pro výše uvedené příklady byly využity parametry $batch\_size = 200$, $batch\_size = 30$, optimizér ADAM a délka historie jednotlivých pozorování 5 (počet historických pozorování, dle kterých se model rozhoduje $length\_of\_sample$). Všechny tyto parametry je možné dále testovat a rozvíjet - viz kapitola 8.8.1.

8.7.1 Výsledky modelu na testovacích datech:

Pro samotné testování byl po několika experimentech použit výše uvedený model s první vrstvou obsahující 128 neuronů a druhou vrstvou s 256 neurony bez aplikace regularizace. Opět, stejně jako u modelu ARIMA, byl model testován pro všechny datové sady s následujícími výsledky:



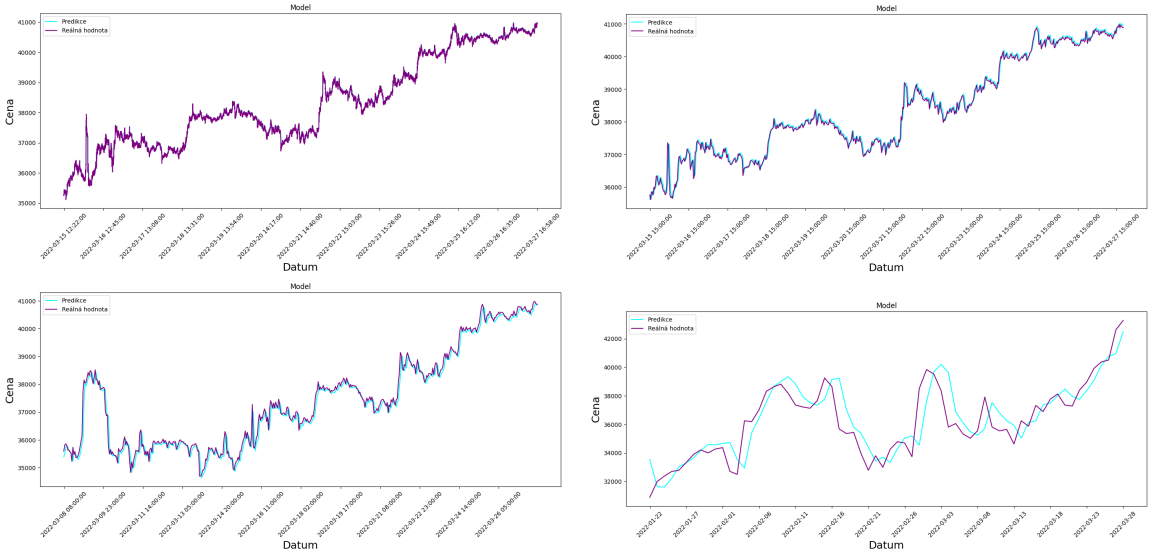
Obrázek 44: Metriky klasické neuronové sítě na minutových, 30 minutových (vpravo nahoře), hodinových (vlevo dole) a denních (vpravo dole) datových sadách [vlastní zpracování]

Již v této chvíli je však znatelný markantní rozdíl oproti modelu ARIMA. Tento model, zdá se, velmi dobře zafungoval na datech minutových, kde se determinační koeficient přibližuje 100 %, avšak na ostatních datech je chybovost vysoká - na datech denních dokonce velmi vysoká. Pro porovnání lze uvést alespoň rozdíl hodnot metrik minutových a denních dat například na ceně kryptoměny BTC:

Inteval setu	R^2	RMSE	MAE	MSE
1min	0.999576	0.000848	0.000561	0.000001
30min	0.988758	0.004409	0.002870	0.000019
60min	0.980284	0.004705	0.003160	0.000022
denní	0.728470	0.023206	0.017142	0.000539

Tabulka 6: Hodnoty metrik predikce ceny kryptoměny BTC ANN modelem

Dle těchto hodnot by neměly predikce modelu vůbec odpovídat skutečným hodnotám, to lze ověřit pohledem na graf pro tyto čtyři datové sady:



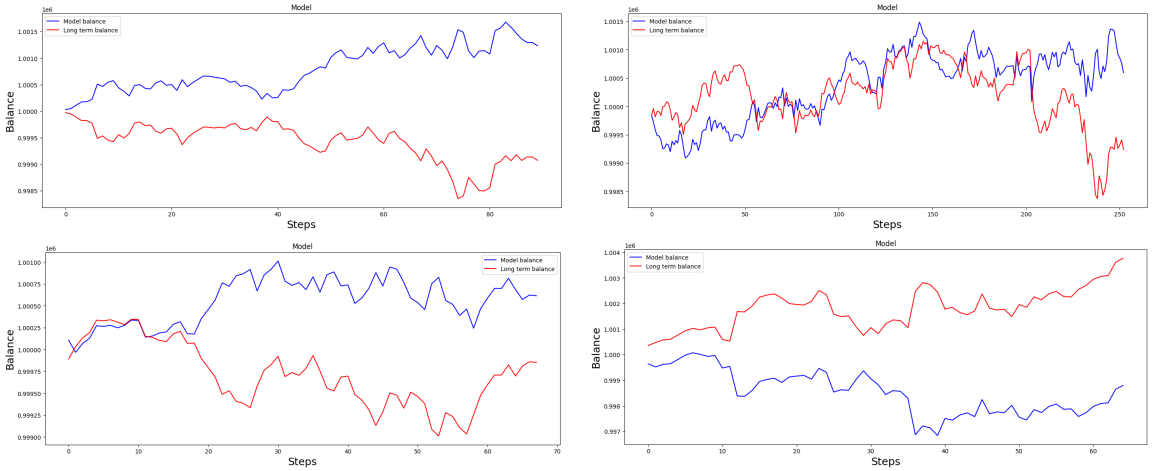
Obrázek 45: Rozdíl predikcí a reálných hodnot na minutových, půlhodinových (vpravo nahoře), hodinových (vlevo dole) a denních (vpravo dole) datových sadách ANN modelu [vlastní zpracování]

Dle tohoto pohledu se však o takovou chybovost samotného modelu nejspíše nejedná. Daný velký pokles na metrikách denních dat je s největší pravděpodobností způsoben velikostí změn ceny BTC. Ve sledovaném období totiž cena tohoto instrumentu kolísá mezi 32 tisíci až 42 tisíci. Druhý faktor, který tento problém mohl zajistit, je nesouměrná velikost dat těchto setů, zatímco minutová data BTC mají přes milion a sto tisíc řádků, denních záznamů je pouze kolem tří tisíc (2751 - 7.5 roku). Tento fakt může též znamenat neschopnost obecné aplikace neuronových sítí na denní data, protože takto malá množina dat nemusí plně stačit modelu ke konvergenci. Důležitějším faktorem, než výsledky metrik, budou však samotné zisky, které by model dle fiktivního obchodování realizoval. Pro zkrácení jsou v tabulce níže uvedeny pouze tři nejlepší a nejhorší výsledky.

datový set	model	instrument	průměrný obchod	normalizovaný profit
denní VEU	1001234	999074	13.560440	94.92
denní JPM	1000595	999239	2.342520	16.4
denní VOO	1000615	999853	8.913043	62.39
denní AAPL	1000790	1003542	3.172691	22.21
60min AMZN	998071	1001047	-11.022857	-1851.84
denní BTC	998800	1003770	-18.181818	-127.27

Tabulka 7: Bilance obchodování pomocí predikcí modelu - minutová data

Pro náhled na možné problémy lze znovu zobrazit grafy vývoje zůstatku modelu.



Obrázek 46: Obchodování dle predikcí modelu u nejlepších výkonností a naopak nejhorší - vlevo nahoře VEU, vpravo nahoře JPM, vlevo dole VOO a vpravo dole BTC [vlastní zpracování]

Zatímco na grafu VEU je znatelná určitá forma systematického odhadu, zda půjde cena nahoru, či dolů, u grafů obchodování s JPM a VOO lze pozorovat několik velmi dobrých predikcí, které dovolily modelu vydělat na poklesu aktiva. Naopak u BTC jsou viditelné oblasti špatných predikcí, kde model i přes ziskovost instrumentu snižoval hodnotu svého „portfolia".

Avšak nutno zmínit, že alespoň dle názoru autora, je i přes některé horší výsledky výkon klasické neuronové sítě velmi dobrý.

8.8 Vytvoření LSTM modelu

Tvorba modelů v knihovně keras, respektive minimálně tvorba jejich struktur, je velmi jednoduchá a tedy ani LSTM nebude po vzoru ANN žádnou velkou změnou. LSTM model prakticky vždy obsahuje vrstvy LSTM neuronů na začátku řetězce, či ve skrytých vrstvách. Poslední vrstvy jsou však, stejně jako u klasických sítí, věnovány klasickým neuronům, které tvoří samotný výsledek z modelu:

```
model = Sequential()
model.add(LSTM(128, return_sequences=True,
    input_shape=(length_of_sample, 1)))
model.add(LSTM(64, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))
```

V tomto modelu je rovnou vytvořena i druhá, skrytá vrstva obsahující polovinu počtu neuronů vrstvy předchozí. Rozdíl v těchto vrstvách je v parametru *return_sequences*. Tento parametr udává, zda vrstva odesílá do další vrstvy pouze jednu hodnotu - hodnota predikce posledního LSTM neuronu, nebo celou sekvenci hodnot ze všech neuronů. Výstup jako sekvence se používá právě pro napojení druhé vrstvy LSTM, která pomocí této sekvence dostává více informací, než když by obdržela pouze poslední výstup.

8.8.1 Pokročilé hledání parametrů

LSTM model, stejně jako model předchozí, obsahuje velmi mnoho hyper-parametrů. Volba těchto parametrů je možná pomocí empirického testování, avšak pokud trénování modelu netrvá extrémně dlouhou dobu, je možné toto vybírání určitým způsobem optimalizovat. První možností je vytvořit několik cyklů, které každou z možností ověří. To je však velmi zdlouhavé i přes krátkou dobu učení. Druhou, sofistikovanější možností je využití knihoven navržených pro tento účel, jednou z nich je například optuna.

Pro použití této knihovny je nutné vytvořit metodu objective s parametrem trial. Tato metoda bude volána několikrát za sebou a pokaždé je nutné, aby obsahovala trénink modelu při jiných parametrech. Tyto parametry mění pomocí právě parametru trial, pomocí kterého knihovna volí aktuální kombinaci. Lépe je tento princip možné popsat pomocí kódu:

```
def objective(trial):
    bs = trial.suggest_int('batch size', 2, 128)
    epochs = trial.suggest_int('epochs', 10, 50)
    opt = trial.suggest_categorical('optimizer',
        ['Adam', 'Adadelta', 'Adagrad', 'SGD'])
    length_of_sample = trial.suggest_int('length_of_sample', 20, 100)
    first_l_size = trial.suggest_int('first_l_size', 96, 256, step=32)
    second_l_size = trial.suggest_int('second_l_size', 16, 128, step=32)
```

Parametr trial, jak je viditelné z kódu, generuje aktuální kombinaci parametrů z výčtu všech možných. Tyto parametry je nutné po vygenerování následně použít v modelu. V případě aplikace z této práce je možné pouze nadefinovat strukturu modelu a zavolat při testování vytvořenou metodu:

```
c_stop = cb.EarlyStopping(
    monitor='root_mean_squared_error',
    patience=7)
_, _, _, _, _, local_evaluations = eval_model_with_results(
    ds_dict,
    ds_name=first_ds,
    ds_split_ratio=0.0,
    scaler=MinMaxScaler(feature_range=(0, 1)),
    train_ratio=0.95,
    length_of_sample=inner_length_of_sample,
    model=inner_model,
    opt=opt,
    bs=bs,
    epochs=epochs,
    fit_callbacks=[PlotLossesKeras(), c_stop])
return local_evaluations[0]
```

V kódu je též viditelné použití techniky EarlyStopping, která zastaví model, pokud se již několik epoch nezlepšil jeho výsledek. Jako výstup z této metody objective je číslo identifikující jednu z metrik. V tomto případě je výstupem metrika mse.

Spuštění samotného testování je následně pouze o vydefinování počtu pokusů (n_trials):

```
import optuna
study = optuna.create_study(
```

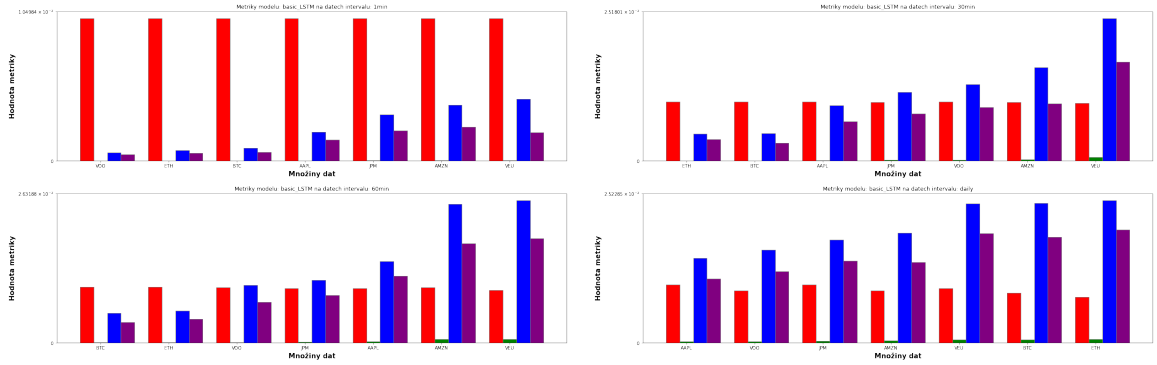
```
    study_name='ananke',
    direction='minimize',
    pruner=optuna.pruners.MedianPruner())
study.optimize(objective, n_trials=40,show_progress_bar=True)
```

Jméno studie není zde příliš podstatné. Direction uvádí, zda má být metrika, která je výstupem z modelu objective minimalizována, či maximalizována (pokud by byla výstupem například hodnota portfolia). Takzvaný pruner dovoluje některé běhy zastavit v případě, že se nejeví jako úspěšné, tato technika však není v tomto případě důležitá, protože se obvykle používá u modelů, které mají velmi dlouhé běhy. Naposledy se nastavuje n_trials, které reprezentují počet pokusů na běh metody.

Po běhu této „studie“ je možné nejlepší parametry následně získat pomocí volání *study.best_params* a *study.best_value*, ta uvádí nejlepší výsledek, kterého model s parametry dosáhl.

8.8.2 Výsledky modelu na testovacích datech:

LSTM model je všeobecně vnímán jako hlavní prostředek pro predikce časových řad a určitým způsobem náhrada modelu ARIMA. Díky tomu by jeho výsledky měly být s metodou ARIMA porovnatelné a předčit model klasické perceptronové sítě (ANN model). Prvními zkoumanými veličinami jsou znovu výsledky metrik z tohoto modelu:



Obrázek 47: Metriky LSTM modelu na minutových, 30 minutových (vpravo nahoře), hodinových (vlevo dole) a denních (vpravo dole) datových sadách [vlastní zpracování]

Z grafů jsou viditelné velmi dobré výsledky pro minutové grafy, avšak stejně jako u klasické neuronové sítě výkon klesá s délkou intervalu časové řady. V tuto chvíli by mohlo být užitečné porovnání právě s ANN modelem:

Metrika	1min	30min	60min	denní
R^2	0.999560	0.979460	0.987671	0.951298
RMSE	0.001803	0.012591	0.010019	0.019365
MAE	0.001332	0.010529	0.007144	0.015841
MSE	0.000003	0.000159	0.000100	0.000375

Tabulka 8: Hodnoty metrik predikce akcie firmy Apple ANN modelem

Metrika	1min	30min	60min	denní
R^2	0.999671	0.993342	0.961260	0.981635
RMSE	0.002005	0.009324	0.014373	0.014332
MAE	0.001485	0.006660	0.011729	0.010825
MSE	0.000004	0.000087	0.000207	0.000205

Tabulka 9: Hodnoty metrik predikce ceny akcie firmy Apple LSTM modelem

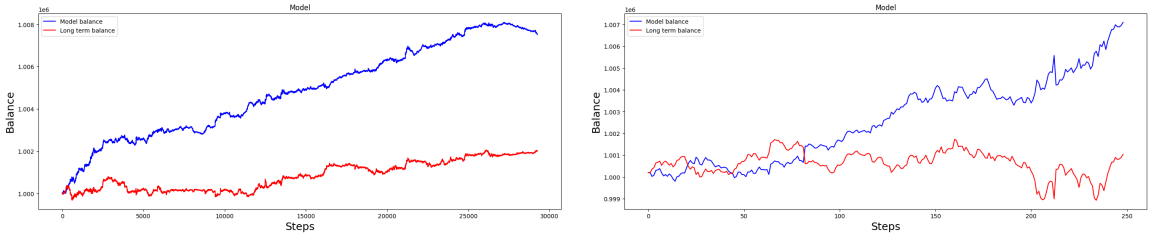
Toto porovnání výsledků v tabulkách 8 a 9 je nejspíše méně průkazné, než by se z definic modelů mohlo zdát, LSTM model má sice výsledky metrik lepší, avšak například u 60 minutových dat je naopak lepší klasická neuronová síť. Ve všech případech však nelze konstatovat velký rozdíl. Rozhodovat proto musí znovu porovnání teoretické ziskovosti. Nejlepší ziskovosti v tomto ohledu dosáhl model u minutových dat kryptoměn ETH a BTC, současně s denními daty akcie Amazon a indexu, respektive ETF - VEU.

datový set	model	instrument	průměrný obchod	normalizovaný profit
1min - ETH	1007516	1002006	0.188298818	2589.07
denní - AMZN	1007098	1001038	24.240000000	198.74
1min - BTC	1005567	1001417	0.141822158	1917.69
denní - VEU	1000962	999240	11.403973510	44.6

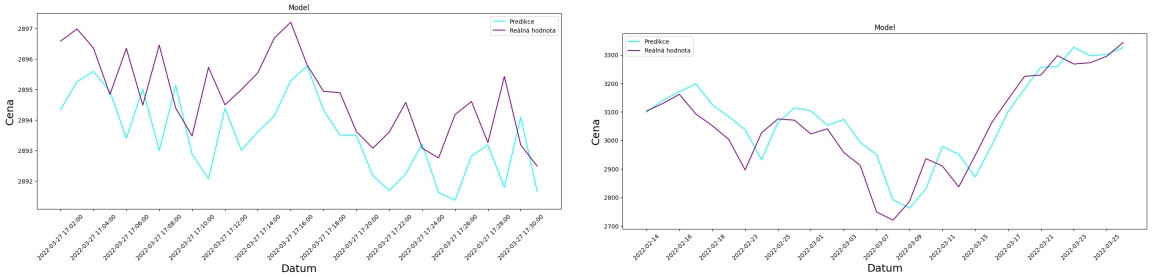
Tabulka 10: Bilance obchodování pomocí predikcí LSTM modelu - minutová data

V tabulce 10 jsou již data oproti ANN modelu odlišná, na rozdíl od něj totiž LSTM model nejspíše dokázal s určitou spolehlivostí vzory v časové řadě predikovat.

Historie bilancí virtuálního obchodování vypadala pro tento model následovně:



Obrázek 48: Obchodování dle predikcí modelu LSTM - vlevo minutová data ETH, vpravo denní AMZN [vlastní zpracování]



Obrázek 49: Predikce a reálné hodnoty minutových dat ETH a denních dat AMZN [vlastní zpracování]

Jak je možné vidět v grafu porovnávajícím predikce s reálnou hodnotu, vypočítané hodnoty se stále nedají vysvětlit pouhým pozorováním.

8.9 Obousměrný LSTM model

Dalším z testovaných modelů je (7.10.3), obousměrný model LSTM. Tato architektura je nadstavbou rekurentních sítí, takže kromě využití pro model LSTM lze tento způsob uplatnit též například pro model GRU. Způsob vytvoření takovéhoto modelu se trochu liší od modelů dřívějších:

```
model = Sequential()
model.add(
    Bidirectional(
        LSTM(128, return_sequences=False),
        input_shape=(length_of_sample, 1)))
model.add(Dense(16))
model.add(Dense(1))
```

V tomto modelu je nutné inicializovat vrstvu *Bidirectional* přímo s LSTM jednotkou v parametru, kterou Bidirectional objekt „obalí“.

8.9.1 Výsledky modelu na testovacích datech:

Obousměrný LSTM model by měl teoreticky výsledky standardního modelu překonat. Již pohledem na tabulku níže, zobrazující porovnání metrik na datasetech cen akcie firmy Apple mezi klasickým LSTM modelem a jeho obousměrnou variantou, je jasné, že v tomto případě model nedosáhl kýženého efektu.

I když jsou například hodnoty metrik zobrazené v tabulce 11 velmi podobné, ani na jednom ze všech datasetů obousměrný model metrikami nepřekonal klasický model.

Zajímavé je však též porovnání bilancí nejziskovějších datových setů, kde jsou oba modely v nejlepších výsledcích téměř shodné.

datový set	model	instrument	průměrný obchod	norm. profit
1min - ETH - LSTM	1007516	1002006	0.188298818	2589
1min - ETH - Bi-LSTM	1007656	1002006	0.193083180	2637.3
denní - VEU - LSTM	1000962	999240	11.403973510	44.6
denní - VEU - Bi-LSTM	1001162	999240	12.728476821	53.9

Tabulka 12: Bilance obchodování pomocí predikcí modelu - minutová data

Inteval setu a model	R^2	RMSE	MAE	MSE
1min - LSTM	0.999671	0.002005	0.001485	0.000004
1min - Bi-LSTM	0.999144	0.003234	0.002815	0.000010
30min - LSTM	0.993342	0.009324	0.006660	0.000087
30min - Bi-LSTM	0.985857	0.013589	0.009757	0.000185
60min - LSTM	0.961260	0.014373	0.011729	0.000207
60min - Bi-LSTM	0.944572	0.017192	0.013735	0.000296
denní - LSTM	0.981635	0.014332	0.010825	0.000205
denní - Bi-LSTM	0.921214	0.029686	0.024843	0.000881

Tabulka 11: Hodnoty metrik predikce ceny společnosti Apple Inc. modely LSTM a Bi-LSTM

Kromě těchto dvou příkladů, je však v porovnání s klasickou variantou LSTM, tento model více ztrátový.

8.10 Komplexní model

Již o něco složitější je model nazvaný komplexní, někdy též nazývaný ensamble - seskupený. Tyto architektury se pokouší pomocí spojení dvou modelů zlepšit výsledek, který generují samostatně. Jinými slovy, pokud se dva modely naučí na stejných datech, nemusí být jejich vnitřní stavy stejné a tedy jejich zkombinováním může nějaká nesdílená informace přispět k dalšímu zlepšení modelu. Tvorba tohoto modelu je možná přes speciální metodu tensorflow concatenate, která spojuje výstupy dvou vrstev a předává do jediné další.

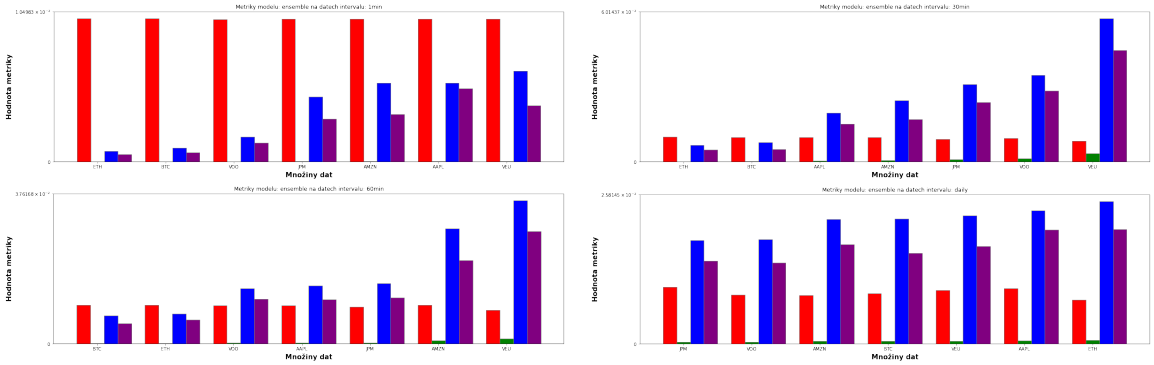
Složitější strukturu modelu však nebylo možné definovat stejným způsobem jako modely předchozí, proto byl pro tento model využit takzvaný funkcionální přístup, který místo do jedná Sequential proměnné, přidává vrstvy do dalších v řadě. První musí být vždy definován vstup (Input), který je zároveň přidělen k oběma spojovaným modelům. Tímto způsobem se docílí toho, že do modelu není nutné přikládat dva vstupy, ale oba modely se dělí o jeden.

```
input1 = Input(shape=(length_of_sample, 1))

first = Bidirectional(LSTM(128, return_sequences=False),
    input_shape=(length_of_sample, 1))(input1)
first = Dense(16)(first)

second = Bidirectional(LSTM(128, return_sequences=False),
    input_shape=(length_of_sample, 1))(input1)
second = Dense(16)(second)

adding_model = layers.concatenate([first, second], axis=1)
adding_model = Dense(32)(adding_model)
adding_model = Dense(1)(adding_model)
```

Jak je možné vidět v kódu, oba modely mají stejnou strukturu, jejich vnitřní stavy budou však jiné, kvůli náhodnosti inicializace vah při začátku trénování. Každý bude mít v moment predikování jiný výsledek a je pouze na vrstvách klasické neuronové sítě (na vrstvě předposlední - o 32 neuronech), aby rozhodla, který z výsledků je lepší, případně je mezi sebou jistým způsobem upravila.

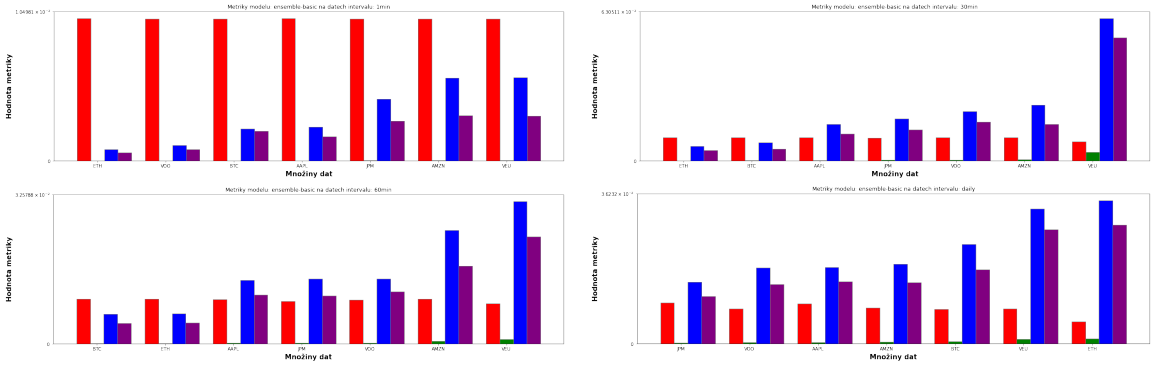
8.10.1 Výsledky modelu na testovacích datech:

Tento model je o něco specifičtější, než byly modely předchozí. Kvůli napojení dvou samostatných LSTM modulů je možné, že model nebude vůbec schopný zkonvergovat. Po zkontrolování tabulky výsledků však tato situace po nastavení modelu nenastala:



Obrázek 50: Metriky komplexního modelu na minutových, 30 minutových (vpravo nahoře), hodinových (vlevo dole) a denních (vpravo dole) datových sadách [vlastní zpracování]

Tento model lze jednoduše porovnat s modelem o stejné struktuře, avšak obsahující pouze LSTM modul místo obousměrného LSTM.



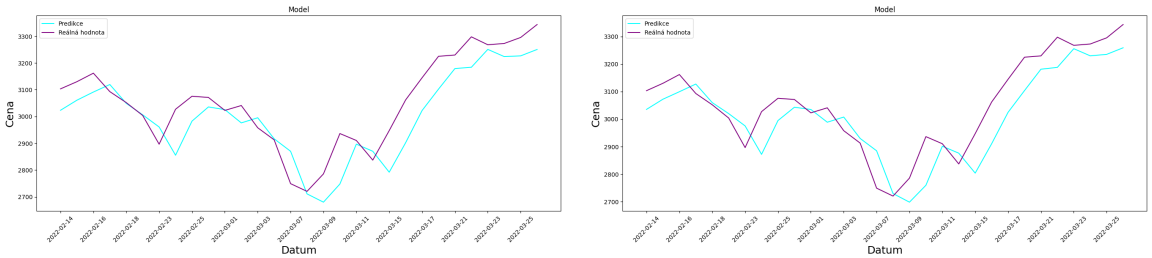
Obrázek 51: Metriky komplexního modelu bez obousměrné vrstvy na minutových, 30 minutových (vpravo nahoře), hodinových (vlevo dole) a denních (vpravo dole) datových sadách [vlastní zpracování]

Oba modely jsou dle tohoto porovnání a i dle porovnání konkrétních výsledků přibližně totožné. Největší fiktivní zisky těchto modelů jsou následující:

datový set	model	instrument	průměrný obchod	norm. profit
denní - AMZN - LSTM	1006856	1001038	23.272	191.97
denní - AMZN - Bi-LSTM	1006544	1001038	22.024	183.23
1min - ETH - LSTM	1004478	1002006	0.0845	1542.56
1min - ETH - Bi-LSTM	1004442	1002006	0.08325	1530.15
denní - VEU - LSTM	1001014	999240	11.7483	47.0
denní - VEU - Bi-LSTM	1000886	999240	10.9	41.07
60min - BTC Bi-LSTM	1002451	1002005	0.5718	527.9
60min - BTC - LSTM	1002199	1002005	0.2487	473.63

Tabulka 13: Ziskovost setů s nejlepšími metrikami komplexního modelu v eurech.

Jak je možné vidět z tabulky výsledků ziskovosti 13, modely jsou si velmi podobné, pro zakončení analýzy tohoto modelu je tedy potřeba ještě porovnat samotné predikce:



Obrázek 52: Porovnání predikcí s reálnými pozorováními v obousměrné variantě (vlevo) a základní LSTM variantě (vpravo) [vlastní zpracování]

Z tohoto posledního porovnání je možné vidět menší rozdíl mezi křivkami predikcí, avšak obecně jsou stále velmi podobné. Tento model lze výkonností porovnávat například s modelem základního LSTM, avšak žádné extrémní zlepšení není zjistitelné, spíše trochu naopak.

8.11 Hybridní model s více vstupy

Posledním z vytvořených modelů je model kombinující několik vstupů. Prozatím existovalo pro každou předpovídanou hodnotu několik historických hodnot (většinou šedesát). V tomto modelu se však kombinuje dalších jedenáct doplňkových informací, které mohou modelu pomoci zlepšit predikci řady. Tyto informace by mohly být jakékoliv - novinky o firmě, reálné hodnoty prodejů firmy apod. Pro účely této práce je však přidáno jedenáct informací, které jsou vytvořeny z hodnot indikátorů v moment predikce. Podrobně to jsou indikátory klouzavého průměru, MACD, Bollingerovi pásma, exponenciální klouzavý průměr a indikátor Momentum. Tyto hodnoty byly připojeny k ceně akcie pro každý z pozorování (funkcí tests.prepare_dataset_with_indicators).

Pro samotný model byl následně použit jednoduchý obousměrný model LSTM s přidáním správného rozměru druhé dimenze vstupních dat (input_shape):

```
model = Sequential()
model.add(Bidirectional(LSTM(128, return_sequences=False),
    input_shape=(length_of_sample, 11)))
model.add(Dense(16))
model.add(Dense(1))
```

8.11.1 Výsledky modelu na testovacích datech:

Stejně jako minulý, kombinovaný model, je ten hybridní částečně velmi experimentální. I přesto, že model popsaný výše byl tvořen a testován na 30 minutových datech cen akcií firmy Apple, jeho výsledky na dalším intervalu jsou následující:

Inteval setu	R^2	RMSE	MAE	MSE
30min	0.987573	0.014771	0.011895	0.000218
denní	-3.054586	0.265407	0.237732	0.070441
60min	0.856902	0.021949	0.019405	0.000482
1min	0.995979	0.008353	0.006970	0.000070

Tabulka 14: Hodnoty metrik predikce ceny akcie AAPL hybridním modelem

Denní ceny však tento model vůbec nedokázal popsat. Při dalším testování a aplikaci na ceny kryptoměny BTC, byly vyhodnoceny následující údaje:

Inteval setu	R^2	RMSE	MAE	MSE
30min	-1.162334	0.072857	0.063363	0.005308
daily	-44.364044	0.517184	0.511452	0.267479
1min	-0.558283	0.060329	0.052529	0.003640
60min	-12.630908	0.147563	0.142047	0.021775

Tabulka 15: Hodnoty metrik predikce ceny kryptoměny BTC hybridního modelem

Tato tabulka potvrzuje, že tento model sice celkem dobře popisuje minutová i půlhodinová data akcie firmy Apple, větší generalizaci však nezvládne i přes další experimenty.

Nejlepších výsledků model dosahuje na následujících datových množinách:

Inteval setu	R^2	RMSE	MAE	MSE
1min - JPM	0.989227	0.009882	0.009086	0.000098
1min - VEU	0.987418	0.014077	0.012954	0.000198
1min - AAPL	0.985318	0.015962	0.015695	0.000255
30min - AAPL	0.971528	0.022357	0.018671	0.000500

Tabulka 16: Hodnoty nejlepších metrik hybridního modelu

Teoretická ziskovost zobrazena v tabulce 17 však ani na těchto setech není nijak dobrá:

datový set	model	instrument	průměrný obchod	normalizovaný profit
1min - JPM	997514	1000770	-0.649770505	-5000.77
1min - VEU	1000023	1000481	-0.119989521	60.74
1min - AAPL	999640	1001214	-0.193508729	-446.13
30min - AAPL	999253	1001199	-6.138801262	-791.77

Tabulka 17: Teoretická ziskovost setů s nejlepšími metrikami hybridního modelu v eurech.

V dalším pokusu byl použit model s jednoduchým LSTM modelem místo obousměrného, avšak bez většího účinku. Stejně tak dopadlo experimentování s různými hodnotami hyperparametru určující počet historických záznamů, který by mohl toto chování ovlivnit.

9 Shrnutí výsledků

V minulé kapitole bylo představeno sedm modelů, které slouží pro predikci časových řad. V rámci implementace byly zavedeny metriky, kterými je možné modely zkoumat a také způsoby fiktivního obchodování, kterými lze zjistit použitelnost určitého modelu při investování. Fiktivní obchodování je představeno v kapitole 8.4.6. Jedná se o umělé obchodování pomocí obchodní strategie, která „nakoupí“ instrument na začátku obchodního dne, pokud model předpovídá kladnou změnu ceny instrumentu, nebo instrument prodá (shortuje) v případě opačné předpovědi. Pokaždé vkládá do nákupu 10 tisíc eur a hodnota počátečního kapitálu je jeden milion eur. Fiktivní obchodování abstrahuje od poplatků a spreadu, avšak to by nemělo ovlivnit porovnatelnost ziskovosti modelů a pasivního investování. V této kapitole budou srovnány všechny modely po jednotlivých intervalech různých datových sad. Jak již bylo zmíněno, je velmi pravděpodobné, že modely budou mít různou schopnost predikovat data na odlišných datových souborech a zvláště pokud se jedná o množiny obsahující jiný typ dat (minutová data / hodinová data).

9.1 Denní data

Prvním exemplárním typem dat jsou denní data, obsahující cenu, za kterou bylo aktivum prodáváno v daný den. Datové sady s tímto typem dat mají relativně málo záznamů oproti sadám s jiným typem. Testování bylo provedeno na již zmíněných 7 instrumentech (Apple Inc., JPMorgan Chase & Co., Amazon.com, Inc., ETF S&P500, All-World ETF, Bitcoin, Ethereum) pro 7 modelů - ARIMA, klasická neuronová síť, model LSTM, obousměrný model LSTM (Bi-LSTM), model komplexní (Ensemble) obsahující obousměrný LSTM, model komplexní obsahující pouze jednostranný LSTM modul a modul Hybridní, který kombinuje několik vstupů. Jako hlavní metriku lze zvolit koeficient determinace, protože zjednodušeně vyjadřuje vysvětlitelnost časové řady modelem 8.4.5. Po celé sadě testování vznikne následující tabulka:

	ARIMA	ANN	LSTM	Bi-LSTM	Ensemble	Ens.-lstm	Hybrid
AAPL	0,9988	0,9513	0,9816	0,9212	0,9525	0,9696	-3,0546
AMZN	0,9952	0,8701	0,8792	0,7758	0,8365	0,8696	-106,3830
BTC	0,9625	0,7285	0,8396	0,7182	0,8652	0,8350	-44,3640
ETH	0,9602	0,7940	0,7712	0,2344	0,5280	0,7604	-4,3499
JPM	0,9961	0,9101	0,9846	0,9817	0,9837	0,9888	0,4232
VEU	0,9892	0,8978	0,9170	0,8295	0,9261	0,8400	0,7222
VOO	0,9937	0,8435	0,8837	0,6734	0,8466	0,8409	-3,0870
ø	0,9851	0,8565	0,8938	0,7335	0,8484	0,8720	-22,8704

Tabulka 18: R^2 pro jednotlivé modely testované na denních datových sadách

Nejlépe ve srovnání dopadl klasický model ARIMA s hodnotou koeficientu přes 98 %. Naopak, například hybridní model nedokázal přesvědčivě popsat řadu ani v jednom z výše uvedených setů. Velmi dobře si nadále vedly i ostatní modely, kromě obousměrného LSTM modelu, který se 73 % je spíše za hranou akceptace. Vítězství ARIMA modelu může mít hned několik příčin. Jednou z nich může být například velmi malý počet pozorování, která data obsahují - tato skutečnost by se moha vylepšit i na dalších datových typech.

Diskriminace řady však není zárukou úspěchu při obchodování, proto bylo aplikováno i testování fiktivním modelem obchodování 8.4.6. Ziskovost jednotlivých modelů je vyjádřena znovu tabulkou:

	ARIMA	ANN	LSTM	Bi-LSTM	Ensemble	Ens.-lstm	Hybrid
AAPL	998474	1000790	1000663	1000145	999555	997499	999118
AMZN	1008056	999630	1007098	999720	1006544	1006856	999376
BTC	986412	998800	1000118	994084	999310	999466	1000196
ETH	999275	1002003	1001455	999389	1000475	1001949	998944
JPM	992715	1000595	999271	1002787	1000801	1000709	995927
VEU	996235	1001234	1000962	1001162	1000886	1001014	999936
VOO	1000245	1000615	999584	1000494	1000002	1000104	1001522
ø	997345	1000524	1001307	999683	1001082	1001249	999288

Tabulka 19: Teoretická ziskovost modelů na testovaných denních datových setech v eurech.

Z výše uvedeného výčtu v tabulce 19 je jasně vidět paradox předpovědi vzhledem ke změnám ceny. Model ARIMA, který nejlépe vysvětloval testovanou řadu nejspíše nedokázal správně předpovídat změny v cenách, které nastávaly a právě z toho důvodu v prvních metrikách chybovosti jasně vyhrál, avšak v následném fiktivním obchodování skončil až na posledním místě. Vítězem je model LSTM, který nejlépe dokázal predikovat směr ceny v dalším kroku řady.

9.2 Hodinová data

Stejným způsobem byla testována i hodinová data s výsledky metriky koeficientu determinace uvedenými v následující tabulce:

	ARIMA	ANN	LSTM	Bi-LSTM	Ensemble	Ens.-lstm	Hybrid
AAPL	0,9838	0,9877	0,9613	0,9446	0,9606	0,9639	0,8569
AMZN	0,9868	0,9856	0,9769	0,9738	0,9678	0,9763	-1,5647
BTC	0,9974	0,9803	0,9828	0,9480	0,9696	0,9737	-12,6309
ETH	0,9981	0,9929	0,9837	0,9739	0,9709	0,9775	-0,1362
JPM	0,9947	0,9782	0,9557	0,8859	0,9184	0,9280	0,8897
VEU	0,9852	0,9563	0,9229	0,8377	0,8425	0,8818	0,9109
VOO	0,9910	0,9841	0,9763	0,9443	0,9561	0,9539	-0,1133
ø	0,9910	0,9807	0,9656	0,9297	0,9408	0,9507	-1,6839

Tabulka 20: R^2 pro jednotlivé modely testované na hodinových datových sadách

I v tomto případě vítězí model ARIMA, avšak je nutné podotknout, že ostatní modely neuronových sítí, kromě hybridního modelu, mají rozdíl do sedmi procent od nejlepšího modelu.

Níže v tabulce 21 je znovu uvedena ziskovost při aplikaci virtuálního obchodování pomocí testovaných modelů:

	ARIMA	ANN	LSTM	Bi-LSTM	Ensemble	Ens.-lstm	Hybrid
AAPL	998954	1000481	999537	998137	998485	998427	999451
AMZN	996013	998071	997866	998280	998198	998466	998838
BTC	1001816	999985	999779	1003143	1002451	1002199	998411
ETH	1002102	999678	998877	1000529	998331	1002011	998204
JPM	996191	1000348	1000526	1000438	999380	999300	1000794
VEU	999342	999897	1000384	1000498	1000018	999566	1000041
VOO	997503	1000545	1000308	999190	998900	999356	999813
ø	998846	999858	999611	1000031	999395	999995	999365

Tabulka 21: Teoretická ziskovost modelů na hodinových datových setech v eurech.

V případě hodinových dat je však situace naprosto jiná než u dat denních. V tomto případě kromě komplexního modelu nedokázal žádný vykázat zisk a i predikce změn takovýchto modelů tedy nebyla dostatečná. Za zmínku však stojí především model Bi-LSTM, který dokázal vygenerovat zisk ve čtyřech ze sedmi případů. I přesto je však nutné konstatovat nepoužitelnost modelů na těchto datech.

9.3 Půlhodinová data

Další ze vzorů použitých pro testování jsou data půlhodinová. Tato data již neobsahují převážně rysy hlavního trhu, ale převládají vlastnosti tvořené samotným obchodováním a samotnými tradery, proto je možné, že modely změní chování, které bylo možné abstrahovat z minulých datových setů. Úspěšnost a ziskovost těchto modelů je tedy následující:

	ARIMA	ANN	LSTM	Bi-LSTM	Ensemble	Ens.-lstm	Hybrid
AAPL	0,9916	0,9795	0,9933	0,9859	0,9710	0,9817	0,9876
AMZN	0,9933	0,9803	0,9897	0,9793	0,9749	0,9769	-3,1127
BTC	0,9986	0,9888	0,9913	0,9879	0,9763	0,9761	-1,1623
ETH	0,9990	0,9933	0,9943	0,9942	0,9878	0,9895	-0,2987
JPM	0,9973	0,9190	0,9870	0,9708	0,9069	0,9694	0,7439
VEU	0,9913	0,8356	0,9697	0,9329	0,8274	0,8103	0,8659
VOO	0,9956	0,9771	0,9923	0,9856	0,9444	0,9800	-0,0888
ø	0,9953	0,9534	0,9883	0,9766	0,9412	0,9548	-0,2950

Tabulka 22: R^2 pro jednotlivé modely testované na půlhodinových datových sadách

	ARIMA	ANN	LSTM	Bi-LSTM	Ensemble	Ens.-lstm	Hybrid
AAPL	997132	999928	1000097	1000083	1000111	1000335	999253
AMZN	998148	1000073	998728	999710	999300	999260	998925
BTC	1000556	999873	1000650	999912	999936	999964	998651
ETH	1003111	1000127	1000263	999725	999521	998801	998015
JPM	998467	1000071	1000696	1000204	999292	1000146	999868
VEU	998888	999709	1000124	999614	999940	998940	999352
VOO	1000785	999612	999809	999807	999793	1000019	999271
ø	999584	999913	1000052	999865	999699	999574	999048

Tabulka 23: Teoretická ziskovost modelů na půlhodinových datových setech v eurech.

Bohužel ani z těchto datových sad není možné konstatovat úspěch modelů, protože i přes perfektní popsání dat modely, ať již ARIMA modelem, nebo i LSTM, Bi-LSTM a relativně i ostatními, žádný z nich nedokázal v souhrnu vygenerovat zisk. Pokud se ovšem porovnají tyto ziskovosti s předešlými případy jsou vidět opakující se vzory. Například prakticky všechny modely velmi špatně predikují změny ceny akcie firmy Amazon.com. Pokud se tato firma z virtuálního portfolia odstraní, stane se model LSTM ziskový a vykáže ztrátu pouze u jednoho aktiva z šesti. Odstraněním firmy Amazon i z hodinových dat, se stane obdobný efekt také u modelu Bi-LSTM.

9.4 Minutová data

Posledním testovaným typem dat jsou minutová data:

	ARIMA	ANN	LSTM	Bi-LSTM	Ensemble	Ens.-lstm	Hybrid
AAPL	0,9988	0,9996	0,9997	0,9991	0,9975	0,9995	0,9960
AMZN	0,9902	0,9990	0,9994	0,9990	0,9987	0,9986	-1,3767
BTC	0,9882	0,9996	0,9997	0,9997	0,9996	0,9979	-0,5583
ETH	0,9925	0,9995	0,9998	0,9996	0,9998	0,9998	-0,7596
JPM	0,9925	0,9958	0,9989	0,9986	0,9979	0,9980	0,9973
VEU	0,9792	0,9935	0,9990	0,9984	0,9979	0,9982	0,9976
VOO	0,9915	0,9973	0,9994	0,9930	0,9949	0,9980	-1,7702
ø	0,9904	0,9978	0,9994	0,9982	0,9981	0,9986	-0,2106

Tabulka 24: R^2 pro jednotlivé modely testované na minutových datových sadách

	ARIMA	ANN	LSTM	Bi-LSTM	Ensemble	Ens.-lstm	Hybrid
AAPL	999885	999313	998945	999057	999521	999299	999640
AMZN	999871	998498	996436	997312	997362	997782	998941
BTC	999908	999905	1005567	1004849	1000705	1002883	998676
ETH	999536	1001456	1007516	1007656	1004442	1004478	998092
JPM	999238	998277	995654	995518	996354	996524	997514
VEU	1000061	999977	998989	999807	999807	999793	1000023
VOO	1000164	999792	999877	1000205	1000197	1000435	999292
ø	999809	999603	1000426	1000629	999770	1000171	998883

Tabulka 25: Teoretická ziskovost modelů na minutových datových setech v eurech.

Tato sada dat je již z hlediska modelů neuronových sítí daleko zajímavější než sady předešlé. Nejenže poprvé model LSTM porazil model ARIMA v chybovostní metrice, ale také společně s Bi-LSTM modelem zakončil testovací období ziskem převyšujícím ztráty ostatních modelů. Druhým pohledem je však viditelné, že veškerý zisk těchto modelů byl vygenerován pouze dvěma tituly kryptoměn BTC a ETH. Na titulech akcií a ETF oba modely vykázaly, i přes skvělé metriky, nakonec ztrátu.

9.5 Struktura aplikační části práce

Samotná aplikační část je dostupná na verzovacím systému gitlab [1]. Tento repozitář obsahuje již zmíněný hlavní Jupyter notebook - **work.ipynb**, kde lze nalézt exemplární případy užití všech výše zmíněných modelů na datech cen akcií Apple. Dále je však možné zobrazit soubor tests.py, který na rozdíl od Jupyter notebooku obsahuje systematické testování všech modelů na všech dostupných datových množinách. Při testování ukládá jednotlivé výsledky do složky *data-results* a natrénované modely do *models* (tento repozitář je verzován pomocí git funkce LFS). Při ukládání výsledků do json souboru je následně zobrazen a uložen i graf posledních 30 predikovaných testovacích údajů a ziskovost na testovacích datech. Tyto grafy jsou uloženy ve složce *plots*. Veškeré pomocné funkce, jak pro notebook, tak pro testovací soubor jsou dostupné ve složce/package *util*. Posledním důležitým souborem je druhý Jupyter notebook *result_data.ipynb*, který obsahuje zpracování výsledných json souborů, jejich zobrazení do grafů a základní filtrace výsledků a bilancí.

Dataset	Interval	Model	Ziskovost v EUR
ETH	1min	Bi-LSTM	1007656
BTC	1min	LSTM	1005567
VEU	1min	ARIMA	1000061
ETH	30min	ARIMA	1003111
VOO	30min	ARIMA	1000785
JPM	30min	LSTM	1000696
ETH	60min	ARIMA	1002102
BTC	60min	Bi-LSTM	1003143
VEU	60min	Bi-LSTM	1000498
JPM	60min	Hybridní	1000794
BTC	denní	Hybridní	1000196
VOO	denní	Hybridní	1001522
VEU	denní	ANN	1001234
AAPL	denní	ANN	1000790

Tabulka 26: Shrnutí nejlepších modelů, které překonaly ziskovost samotného instrumentu na jednotlivých variantách datových setů.

[1] Zdrojový kód aplikace: `https://gitlab.com/kamest/Ananke`

10 Závěry a doporučení

Cílem této diplomové práce byl návrh a implementace několika modelů, které dokáží samostatně obchodovat na burze a generovat stabilní zisk, který by byl vyšší než zisk po prodeji pouze drženého aktiva (v případě nákupu na začátku investování a prodeje na konci horizontu).

Pro tyto účely bylo zaimplementováno sedm modelů, z toho jeden statistický (ARIMA) a zbývající, které používají neuronové sítě. Po otestování všech těchto vytvořených modelů na všech testovacích datech je nutné konstatovat, že modely navržené v této práci nemají dostatečnou přesnost na to, aby byly použitelné jako automatický nástroj pro robotické investování. Nejlepších výsledků však dosáhly modely LSTM a Bi-LSTM, naproti tomu, že složitější modely by teoreticky tyto základní LSTM i Bi-LSTM měly porazit. Tento fakt podporuje hlavně velká ztrátovost modelů při testování pomocí fiktivního obchodování. Toto obchodování bylo oproštěno od poplatků, spreadů a dalších možných negativních vlivů na výsledek testování, proto by reálná ztráta modelů byla ještě o něco vyšší.

Důvodů, proč se těmto modelům nepodařilo naplnit zamýšlený cíl je nejspíše hned několik. Hlavním důvodem je zřejmě komplexita a náhodnost samotných časových řad cen. Například pro předpověď objemu objednávek v dalším týdnu je model nucen naučit se vzory vycházející z principů střídajících se víkendů/svátků a otevíracích hodin obchodu. V případě cen akcií je však těchto řídících faktorů nespočitatelné množství. Dalším faktorem je obrovský zájem investorů předpovědět tuto cenu, pokud by cenu predikovalo velké množství a všichni se společně chovali na doporučení takovéhoto modelu, okno příležitosti, kterého by tento model využíval, by se tím automaticky též zavřelo, protože by poptávka ceně přidala další nevysvětlenou proměnnou, s kterou by model nepočítal[45].

Jako druhý možný problém modelů využitých a testovaných v této práci je omezenost jejich aplikace. Existuje velmi málo portálů, které poskytují objemná historická data pro konkrétní tituly aktiv a povětšinou jsou tyto sady dat ještě placené. Byly proto v této práci použity, mimo denních dat, pouze data za období posledních dvou let. V době psaní této práce však probíhá válečný konflikt na nedaleké Ukrajině současně se stále nekončící krizí kolem nemoci Covid-19. Oba tyto veskrze globální problémy vedly k velkému ovlivnění cen akcií, ať již po omezené období, nebo i částečným ovlivněním mechanizmu cenotvorby. Proto je též možné, že modely neměly možnost být ziskovější, protože tyto faktory nelze za žádných okolností predikovat. Touto argumentací však nelze vysvětlovat obecnou neefektivitu modelů, je to pouze jeden z možných negativních faktorů, který při testování modely ovlivnil.

Výsledky v praktické části ukazují, že některé modely sice předčí výnosnost samotného aktiva v daném období, avšak tuto situaci není možné žádným způsobem zaručit, protože modely nebyly ziskové na všech testovaných aktivech, ani na všech intervalech dat. Kvůli tomu je nutné konstatovat, že hypotéza H1(3.1), která předpokládá zaručení zisku, musí být zamítnuta. V praktické části byl dále použit takzvaný hybridní model, který kombinuje metody strojového učení a hodnoty indikátorů z technické analýzy. Tento model se však již při implementaci stával nestabilní a i přesto, že některé datové sady vysvětluje s velmi dobrým výsledkem, na většině ostatních datových sadách nedokáže zkonvergovat do optimálního bodu a tedy nedokáže ani zpracovávat výsledky. Kvůli tomu je nutné zamítnout i hypotézu č.2(3.1).

Oproti tomu, z dat se však prokázala vlastnost implementovaných sítí daleko přesněji předpovídat časové řady menších intervalů než například denní časovou řadu, kde většina modelů strojového učení neměla uspokojivý výsledek. Tyto výsledky se však často lišily

i v rámci jednoto typu časové řady a lze tedy konstatovat, že nynější implementace modelů jsou přímo závislé jak na typu časové řady, která může velmi negativně ovlivnit celkový výsledek modelu, tak i na samotných vzorech v časové řadě. Z tohoto důvodu je tedy nutné potvrdit hypotézu H3(3.1).

Obecně v oboru predikcí časových řad finančních trhů existuje, dle názoru autora, velmi málo článků a literatury oproti jiným aplikacím machine learning algoritmů. Avšak i přesto je spousta dalších typů modelů a metod, které lze dále použít pro rozšíření modelů. Prvním takovým vylepšením, které by nebylo těžké aplikovat, jsou konvoluční vrstvy místo klasických rekurentních. Tento princip aplikují ve své publikaci například Sheng Chen a Hongxiang He [62], kteří používají pouze vrstvy konvoluční a perceptronové a proklamují zajímavé výsledky. Velmi slibným se však zdá princip vytvoření vstupních konvolučních vrstev, následovaný LSTM moduly, za kterými jsou napojeny již pouze výstupní lineární neurony. Tento princip využívá například Wenjie Lu a kolektiv [63]. Mezi komplikovanější principy používané v posledních pár letech se zdají být modely, využívající moduly ESN - echo state network. Aplikace tohoto přístupu je například v publikaci od Huaguang Zhanga[64] a kolektivu, popisující aplikaci a zajímavé kladné výsledky.

Techniky zmíněné výše používají, stejně jako modely v této práci, pouze agregované informace z ceny aktiva. Stejně jako u hybridního modelu v praktické části této práce, se v publikacích využívajících konvoluční sítě používají hodnoty převážně technických indikátorů. Existují však i daleko pokročilejší metody, které se snaží dodat do modelu informace z externích zdrojů. Od informací z určených sociálních sítí přes dodatečné informace o finančním trhu až po fiktivně vytvořené časové řady, které mají za úkol napodobit testovanou řadu a uměle tak dodat modelu ceněná data.

Vyvinuté modely jsou však daleko jednodušší než ty prezentované výše. I přesto však nelze jednoznačně tvrdit, že žádná aplikace (například LSTM modelu) nebude zisková při predikci ceny některé z kryptoměn. Na základě informací a prezentovaných výsledků v této práci, je možné vytvořené modely nadále obohatit a nalézt řešení, která opravdu splní požadavky spolehlivých nástrojů pro investování.

Literatura

[1] ig. Beginners' guide to fundamental analysis. Dostupné z: https://www.ig.com/en/trading-strategies/beginners-guide-to-fundamental-analysis-190503

[2] Novotný, R. Východiska diverzifikace rodinného jmění. Aug. 2013. Dostupné z: https://www.investujeme.cz/clanky/vychodiska-diverzifikace-rodinneho-jmeni/

[3] tradingView, t. tradingView. Dostupné z: https://www.tradingview.com/symbols/NASDAQ-AAPL/

[4] Vencálek, O. Časové řady I. Mar. 2015. Dostupné z: https://info.sso.vsb.cz/cz.vsb.edison.info.web/attachment/CasoveRady1_1.pdf?attachmentId=26708

[5] Knutsen, P. Time series - Analyse ACF and PACF plots. Dostupné z: https://stats.stackexchange.com/questions/134487/analyse-acf-and-pacf-plots

[6] Shin, T. All Machine Learning Models Explained in 6 Minutes. Nov. 2021. Dostupné z: https://towardsdatascience.com/all-machine-learning-models-explained-in-6-minutes-9fe30ff6776a

[7] javatpoint. Machine Learning Models - Javatpoint. Dostupné z: https://www.javatpoint.com/machine-learning-models

[8] Lessner, D. Soubor:Rozhodovaci strom priklad jazyky.svg – Základy informatiky pro střední školy. Dostupné z: https://popelka.ms.mff.cuni.cz/~lessner/mw/index.php/Soubor:Rozhodovaci_strom_priklad_jazyky.svg

[9] Navlani, A. Python Logistic Regression Tutorial with Sklearn & Scikit. Dec. 2019. Dostupné z: https://www.datacamp.com/community/tutorials/understanding-logistic-regression-python

[10] Dasila, M. S. Basics K-Means Clustering algorithm. July 2019. Dostupné z: https://medium.com/@msdasila90/basics-k-means-clustering-algorithm-a77c539c9e00

[11] Quora. What exactly is overfitting and why do we prefer models that aren't overfitted even when results are better? Dostupné z: https://www.quora.com/What-exactly-is-overfitting-and-why-do-we-prefer-models-that-arent-overfitted-even-when-results-are-better

[12] Nagyfi, R. The differences between Artificial and Biological Neural Networks. Sept. 2018. Dostupné z: https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7

[13] v7labs. 12 Types of Neural Networks Activation Functions: How to Choose? Dostupné z: https://www.v7labs.com/blog/neural-networks-activation-functions

[14] KNIME. A Friendly Introduction to [Deep] Neural Networks. Dostupné z: https://www.knime.com/blog/a-friendly-introduction-to-deep-neural-networks

[15] Ademola, O. A. Figure 3. A visual representation of a deep neural network (right) and... Dostupné z: https://www.researchgate.net/figure/A-visual-representation-of-a-deep-neural-network-right-and-a-shallow-neural-network_fig3_356214624

[16] Sweta. Batch , Mini Batch and Stochastic gradient descent. Aug. 2020. Dostupné z: https://sweta-nit.medium.com/batch-mini-batch-and-stochastic-gradient-descent-e9bc4cacd461

[17] Frajberg, D. Introduction to the Artificial Intelligence and Computer Vision revol.... Dostupné z: https://www.slideshare.net/darian_f/introduction-to-the-artificial-intelligence-and-computer-vision-revolution

[18] into Deep, D. 6.3. Padding and Stride — Dive into Deep Learning 0.17.5 documentation. Dostupné z: http://d2l.ai/chapter_convolutional-neural-networks/padding-and-strides.html

[19] Ng., A. Neural Networks and Deep Learning - Course materials. Dostupné z: https://www.coursera.org/learn/neural-networks-deep-learning/home/welcome

[20] Oppermann, A. Regularization in Deep Learning — L1, L2, and Dropout. Aug. 2020. Dostupné z: https://towardsdatascience.com/regularization-in-deep-learning-l1-l2-and-dropout-377e75acc036

[21] Khandewal, H. Momentum ,RMSprop And Adam Optimizer. Aug. 2020. Dostupné z: https://medium.com/analytics-vidhya/momentum-rmsprop-and-adam-optimizer-5769721b4b19

[22] Jordan, J. Setting the learning rate of your neural network. Mar. 2018. Dostupné z: https://www.jeremyjordan.me/nn-learning-rate/

[23] Colah. Understanding LSTM Networks – colahův blog. Dostupné z: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[24] Prasad, Y. Types of RNN (Recurrent Neural Network). Feb. 2020. Dostupné z: https://iq.opengenus.org/types-of-rnn/

[25] Dobilas, S. GRU Recurrent Neural Networks — A Smart Way to Predict Sequences in Python. Feb. 2022. Dostupné z: https://towardsdatascience.com/gru-recurrent-neural-networks-a-smart-way-to-predict-sequences-in-python-80864e4fe9f6

[26] Dobilas, S. LSTM Recurrent Neural Networks — How to Teach a Network to Remember the Past. Mar. 2022. Dostupné z: https://towardsdatascience.com/lstm-recurrent-neural-networks-how-to-teach-a-network-to-remember-the-past-55e54c2ff22e

[27] Harfiya, L. N. Figure 4. The unfolded architecture of Bidirectional LSTM (BiLSTM) with... Dostupné z: https://www.researchgate.net/figure/The-unfolded-architecture-of-Bidirectional-LSTM-BiLSTM-with-three-consecutive-steps_fig2_344751031

[28] Jílek, J. *Akciové trhy a investování*. Praha: Grada, 2009, ISBN 978-80-247-2963-3.

[29] Wang, J.; Deng, H.; Liu, B.; et al. Systematic Evaluation of Research Progress on Natural Language Processing in Medicine Over the Past 20 Years: Bibliometric Study on PubMed. *Journal of medical Internet research*, volume 22, Jan. 2020, doi:10.2196/16816.

[30] B, B. Using the latest advancements in deep learning to predict stock price movements. Jan. 2019. Dostupné z: https://towardsdatascience.com/aifortrading-2edd6fac689d

[31] Bansal, K. You will Lose Money if you Don't Know These Top 7 Golden Rules of Investing. June 2021. Dostupné z: https://medium.datadriveninvestor.com/you-will-lose-money-if-you-dont-know-these-top-7-golden-rules-of-investing-94ef7b0eaf17

[32] Group, C. Time, Not Timing, Is What Matters. Dostupné z: https://www.capitalgroup.com/individual/planning/investing-fundamentals/time-not-timing-is-what-matters.html

[33] Warren, G. *Investment Risk for Long-Term Investors*. ResearchGate, Apr. 2021.

[34] Arlt, J.; Arltová, M.; Rublíková, E. *Analýza ekonomických časových řad s příklady*. Vysoká škola ekonomická Praha, 2002, ISBN 978-80-245-0307-3. Dostupné z: https://is.muni.cz/publication/486204/cs/Analyza-ekonomickych-casovych-rad-s-priklady/Arlt-Arltova-Rublikova

[35] Rejnus, O.; Fio banka. *Finanční trhy*. Praha: Grada, 2014, ISBN 978-80-247-3671-6, oCLC: 897868952.

[36] Geambasu, C.; Sova, R.; Jianu, I.; et al. Risk measurement in post-modern portfolio theory: Differences from modern portfolio theory. *Economic Computation and Economic Cybernetics Studies and Research*, volume 47, Jan. 2013: pp. 113–132.

[37] Graham, B.; Zweig, J. *Inteligentní investor*. Praha: Grada, 2007, ISBN 978-80-247-1792-0, oCLC: 188560654.

[38] Kumar, R. How to Build Stock Technical Indicators with Python. June 2021. Dostupné z: https://medium.com/analytics-vidhya/how-to-build-stock-technical-indicators-with-python-7a0c5b665285

[39] Cory, M. Sentiment Indicator Definition and Example. Dostupné z: https://www.investopedia.com/terms/s/sentimentindicator.asp

[40] Peixeiro, M. The Complete Guide to Time Series Analysis and Forecasting. Apr. 2022. Dostupné z: https://towardsdatascience.com/the-complete-guide-to-time-series-analysis-and-forecasting-70d476bfe775

[41] University, T. P. S. 2.2 Partial Autocorrelation Function (PACF) | STAT 510. Dostupné z: https://online.stat.psu.edu/stat510/lesson/2/2.2

[42] Samuel, A. L. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, volume 3, no. 3, July 1959: pp. 210–229, ISSN 0018-8646, doi:10.1147/rd.33.0210, conference Name: IBM Journal of Research and Development.

[43] Geron, A. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems*. Beijing [China] ; Sebastopol, CA: O'Reilly Media, Inc, second edition edition, 2019, ISBN 978-1-4920-3264-9.

[44] Muller, A. C.; Guido, S. *Introduction to machine learning with Python: a guide for data scientists*. Sebastopol, CA: O'Reilly Media, Inc, first edition edition, 2016, ISBN 978-1-4493-6941-5, oCLC: ocn895728667.

[45] Jansen, S. *Machine Learning for Algorithmic Trading: Predictive models to extract signals from market and alternative data for systematic trading strategies with Python, 2nd Edition*. Packt Publishing, July 2020, ISBN 978-1-83921-771-5.

[46] Colaboratory, G. Google Colaboratory. Dostupné z: `https://colab.research.google.com/`

[47] Deepnote. Deepnote - Data science notebook for teams. Dostupné z: `https://deepnote.com`

[48] Python.org. Welcome to Python.org. Dostupné z: `https://www.python.org/`

[49] R. R: The R Project for Statistical Computing. Dostupné z: `https://www.r-project.org/`

[50] Coombs, J. R. jaraco/configparser. Mar. 2022, original-date: 2019-01-23T15:18:11Z. Dostupné z: `https://github.com/jaraco/configparser`

[51] Matplotlib. Matplotlib — Visualization with Python. Dostupné z: `https://matplotlib.org/`

[52] Migdał, P. livelossplot. Apr. 2022, original-date: 2018-03-10T17:51:43Z. Dostupné z: `https://github.com/stared/livelossplot`

[53] McHardy, S. Welcome to python-binance v1.0.15. Apr. 2022, original-date: 2017-08-14T12:09:45Z. Dostupné z: `https://github.com/sammchardy/python-binance`

[54] Aroussi, R. Download market data from Yahoo! Finance's API. Apr. 2022, original-date: 2017-05-21T10:16:15Z. Dostupné z: `https://github.com/ranaroussi/yfinance`

[55] statsmodels. Introduction — statsmodels. Dostupné z: `https://www.statsmodels.org/stable/`

[56] TensorFlow. TensorFlow. Dostupné z: `https://www.tensorflow.org/`

[57] Keras. Keras: the Python deep learning API. Dostupné z: `https://keras.io/`

[58] Optuna. Optuna - A hyperparameter optimization framework. Dostupné z: `https://optuna.org/`

[59] Finance, Y. Yahoo Finance - Stock Market Live, Quotes, Business & Finance News. Dostupné z: `https://finance.yahoo.com/`

[60] Binance. Nakupovat/prodávat bitcoiny, ethery a altcoiny | Kryptoměnová burza | Binance. Dostupné z: https://www.binance.com/cs

[61] Vantage, A. Free Stock APIs in JSON & Excel | Alpha Vantage. Dostupné z: https://www.alphavantage.co/

[62] Chen, S.; He, H. Stock Prediction Using Convolutional Neural Network. *IOP Conference Series: Materials Science and Engineering*, volume 435, Nov. 2018: p. 012026, doi:10.1088/1757-899X/435/1/012026.

[63] Lu, W.; Li, J.; Li, Y.; et al. A CNN-LSTM-based model to forecast stock prices. *Complexity*, volume 2020, Nov. 2020: pp. 1–10, doi:10.1155/2020/6622927.

[64] Zhang, H.; Liang, J.; Chai, Z. Stock Prediction Based on Phase Space Reconstruction and Echo State Networks. *Journal of Algorithms & Computational Technology*, volume 7, Mar. 2013, doi:10.1260/1748-3018.7.1.87.

Přílohy

Součástí práce je zip archiv, který obsahuje kód aplikace popsané v praktické části. Tento kód je možné též nalézt ve veřejném git repozitáři[2].

Hlavní kód práce je obsažen v souboru *work.ipynb*, ve kterém je možné ke kódu nalézt i stručnou dokumentaci. Zmíněný kód využívá funkce ze složky util, které zjednodušují práci v souboru. Jedná se především o funkce stahující data transformační, evaluační (testovací) a vizualizační (zobrazující grafy). Data jsou stahována pomocí služeb popsaných v kapitole 8.3. Tyto služby vyžadují přístupové api klíče, proto je nutné ve složce config tyto klíče uvést. Data se následně stahují do složky *data*.

Hlavní soubor jupyter notebooku dovoluje vytvořit a testovat jednotlivé modely, avšak otestovat všechny datasety na všech modelech by bylo v notebooku nepraktické (kvůli ukládanému výstupu). Proto je v kořenové složce umístěn i soubor *tests.py*, který za pomocí stejných funkcí (ze složky util), automatizovaně vytváří všechny představené modely a testuje na všech stažených datových sadách. Výsledky se následně ukládají do několika složek pro jednotlivé další observace: *data-results* obsahuje metrické výsledky modelů na jednotlivých množinách dat včetně výsledků fiktivního hospodaření v separátním souboru *json*. Dalším výstupem je složka *plots* obsahující grafy jednotlivých výsledků. Pro možnost opakování měření je při prvním spuštění skriptu vytvořena i složka *models*, která obsahuje všechny naučené modely, při dalším spuštění tak není nutné proces učení znovu absolvovat a čas testování je snížen na jednotky minut (místo jednotek hodin v případě nutnosti učení). Funkci přepoužití uložených modelů je možné vypnout přidáním parametru *train_new_model=True* na funkci *eval_model*.

Posledním z využitelných skriptových souborů je rovněž jupyter notebook - *result_data.ipynb*. Tento notebook obsahuje načtení všech json souborů s výsledky a experimenty s nimi. Obsahuje několik připravených funkcí pro zobrazení obecných výsledků a tvorbu grafů k těmto výsledkům.

Samotné výsledky nejsou z kapacitních důvodů uloženy v archivu, který je součástí práce, ale výše popsané skripty (po dodání klíčů api) veškerá data vygenerují. Git repozitář, na rozdíl od odevzdávaného archivu, obsahuje vytvořené a naučené *h5* modely, json soubory a obrázky s grafy. Tyto dodatečné soubory jsou uloženy v lfs úložišti git repozitáře. Samotná data z licenčních důvodů nejsou umístěna ani v repozitáři git a z uvedených služeb se musí generovat (provádí skript v notebooku *work.ipynb*).

[2] `https://gitlab.com/kamest/Ananke`

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání diplomové práce

Autor:	**Bc. Štěpán Kameník**
Studium:	I2000309
Studijní program:	N1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název diplomové práce:	**Robotické investování**
Název diplomové práce AJ:	Robotic Investing

Cíl, metody, literatura, předpoklady:

Cílem diplomové práce je vyvinout algoritmus robotického investování a za pomocí srovnání určit vhodnost a ziskovost algoritmu oproti ostatním ne-automatizovaným metodám investování.

Základní osnova:

1. Úvod\par
2. Rešerše vybraných použitelných investiční nástrojů (stocks, crypto, forex)\par
3. Teoretická část\par
    1. Základy technické analýzy\par
    2. Styly obchodování (intradenní/swing/long-term)\par
    3. Obchodní taktiky využitelné k algoritmizaci (DCA,RRR,Trendování)\par
    4. Management risku\par
    5. Rešerše nástrojů umožňujících robotické investování\par
    6. Pokročilé algoritmické možnosti (využití machine learningu, použití rozšířeného kontextu)\par
4. Praktická část\par
    1. Implementace algoritmu v konkrétním nástroji / vývoj vlastního řešení pro obchodování\par
    2. Testování analytické i praktické chybovosti (včetně backtestů)\par
    3. Použití některé z pokročilých technik\par
    4. Rešerše návratnosti ne-automatizovaných nástrojů (ETF, dluhopisy)\par
    5. Srovnání teoretických návratností a shrnutí výsledků praktické části\par
5. Závěr\par

1. BACIDORE, Jeffrey M., 2020. Algorithmic Trading: A Practitioner's Guide. New York: TBG

Press. ISBN 978-0-578-71523-0. \par

2. HILPISCH, Yves. Python for Algorithmic Trading: From Idea to Cloud Deployment. B.m.: O'Reilly Media. ISBN 978-1-4920-5335-4. \par

3. CHAKRAVARTY, Satya a Palash SARKAR. Algorithmic Trading Strategies. Bingley: Emerald Publishing Limited, s. 79-89. ISBN 978-1-78973-894-0.\par

4. CHAN, Ernie. Algorithmic Trading: Winning Strategies and Their Rationale. New Jersey: Wiley. ISBN 978-1-118-46014-6. \par

5. NAPATE, Sachin, Mukul THAKUR a D B-SCHOOL. Algorithmic Trading and Strategies. \par

Garantující pracoviště:	Katedra ekonomie, Fakulta informatiky a managementu
Vedoucí práce:	Ing. Jan Mačí, Ph.D.
Datum zadání závěrečné práce:	9.9.2021