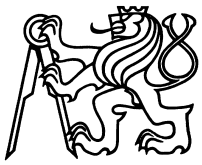
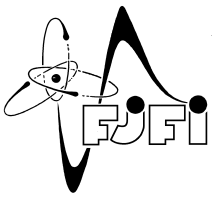


České vysoké učení technické v Praze
Fakulta jaderná a fyzikálně inženýrská



Pokročilé architektury neuronových sítí pro analýzu dat z finančních trhů

Advanced neural network architectures for financial market data analysis

Bakalářská práce

Autor: **Ondřej Šrámek**

Vedoucí práce: **Ing. Pavel Strachota, Ph.D.**

Akademický rok: 2020/2021

České vysoké učení technické v Praze, Fakulta jaderná a fyzikálně inženýrská

Katedra: matematiky

Akademický rok: 2020/2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student:	Ondřej Šrámek
Studijní program:	Aplikace přírodních věd
Studijní obor:	Matematické inženýrství
Studijní zaměření:	Matematické modelování
Název práce (česky):	Pokročilé architektury neuronových sítí pro analýzu dat z finančních trhů
Název práce (anglicky):	Advanced neural network architectures for financial market data analysis

Pokyny pro vypracování:

1) Prostudujte zadanou literaturu a seznamte se s principy hlubokých neuronových sítí, jejich různými typy, algoritmy pro jejich učení a metodikou pro vyhodnocování jejich úspěšnosti.
2) Proveďte rešerši metod a algoritmů používaných pro predikci vývoje časových řad. Soustřeďte se na předzpracování syrových dat a postupy zamezující přetrénování modelů strojového učení (automatický výběr příznaků, testy kauzality).
3) Seznamte se se softwarovými nástroji Keras, Tensorflow a přidruženým ekosystémem jazyka Python pro implementaci hlubokých neuronových sítí.
4) Implementujte některé pokročilé architektury neuronových sítí (LSTM, RCNN, ResNet apod.), experimentujte s jejich nastavením a porovnejte jejich účinnost na predikci časových řad. Využijte datové sady dostupné v komunitě strojového učení (např. na www.kaggle.com) a nebo historická data z kryptoměnové burzy.

Doporučená literatura:

1) C. C. Aggarwal, Neural Networks and Deep Learning. Springer, 2018.
2) F. Chollet, Deep Learning with Python. Manning Publications Co., 2018.
3) G. Zaccone, R. Karim, Deep Learning with TensorFlow. Packt Publishing, 2018.
4) X. Zheng, B. M. Chen, Stock Market Modeling and Forecasting -- A System Adaptation Approach. Springer, 2013.

Jméno a pracoviště vedoucího bakalářské práce:

Ing. Pavel Strachota, Ph.D.
KM FJFI ČVUT v Praze, Trojanova 13, 120 00 Praha 2

Jméno a pracoviště konzultanta:

Datum zadání bakalářské práce: 31.10.2020

Datum odevzdání bakalářské práce: 7.7.2021

Doba platnosti zadání je dva roky od data zadání.

V Praze dne 30.10.2020

.....
garant oboru

.....
vedoucí katedry

.....
děkan

Poděkování:
Chtěl bych zde poděkovat především svému školiteli Ing. Pavlu Strachotovi, Ph.D., za pečlivost, ochotu, vstřícnost a odborné i lidské zázemí při vedení mé bakalářské práce.

Čestné prohlášení:
Prohlašuji, že jsem tuto práci vypracoval samostatně a uvedl jsem všechnu použitou literaturu.

V Praze dne 7. července 2021 Ondřej Šrámek

Název práce:

Pokročilé architektury neuronových sítí pro analýzu dat z finančních trhů

Autor: Ondřej Šrámek

Obor: Matematické inženýrství

Zaměření: Matematické modelování

Druh práce: Bakalářská práce

Vedoucí práce: Ing. Pavel Strachota, Ph.D., KM FJFI ČVUT v Praze, Trojanova 13, 120 00 Praha 2

Abstrakt: V bakalářské práci jsou vysvětleny základní pojmy strojového učení na lineární a logistické regresi, a poté jsou teoreticky uvedeny neuronové sítě. Dále je zkoumán problém přeučení a principy, jak se mu vyhnout. Praktická část je zaměřena na předzpracování časových řad a predikce jejich vývoje pomocí klasifikáčních LSTM neuronových architekturách. Jako časové řady jsou použity vývoje ceny kryptoměnového páru z kryptoměnové burzy Binance. Dále jsou zkoumány úspěšnosti predikce v závislostí na volbě různých hyperparametrů.

Klíčová slova: časové řady, kryptoměnová burza, neuronové sítě, strojové učení

Title:

Advanced neural network architectures for financial market data analysis

Author: Ondřej Šrámek

Abstract: In this bachelor thesis, the basic terminology of machine learning is explained on linear and logistic regression and then the theory of neural networks is laid out. Next the problem of overfitting and principles of how to avoid it are investigated. Practical part is focused on preparation of time series, and prediction of their development with the help of classification LSTM neural architectures. As an example of time series the price evolution of a cryptocurrency pair from the Binance cryptocurrency exchange is used. Next the success rates of prediction with different settings of hyperparameters are investigated.

Key words: time series, cryptocurrency exchange, neural networks, machine learning

Obsah

Úvod

S digitalizací světa a postupným zvyšováním výpočetního výkonu počítačů se zvyšují možnosti využití strojového učení včetně neuronových sítí. Tématem zkoumání práce je aplikace neuronových sítí na kryptoměnové burzy, na kterých se v moderní době začalo úspěšně algoritmicky obchodovat. Rekurentní algoritmy neuronových sítí byly vytvořeny za účelem zkoumání časových řad, což motivovalo jejich využití v této práci pro data z kryptoměnové burzy.

Technická analýza se snaží o predikci vývoje burzy pomocí analyzování údajů tvořené trhem, jako je například cena. Využívá k tomu různých indikátorů (například klouzavé průměry), kterými se řídí obchodníci a podle nich na burzách nakupují a prodávají.

V práci je rozebrán postup a způsob předzpracování dat ve tvaru časových řad. Dále je zkoumáno, zda lze technickou analýzu zobecnit LSTM rekurentní neuronovou sítí, která by dokázala řešit klasifikační úlohu pro pokles a růst ceny na kryptoměnové burze.

V 1. kapitole jsou shrnuty základní pojmy strojového učení na lineární a logistické regresi. Dále jsou teoreticky vysvětleny architektury a principy neuronových sítí, kde se začíná nejjednodušším modelem zvaným perceptron, a poté se přechází ve vícevrstvé dopředné sítě. Na konci kapitoly jsou rozebrány rekurentní neuronové sítě.

V 2. kapitole je diskutován problém přeučení. Pojem je nejdříve vysvětlen, poté je uveden prahově-rozptylový kompromis, který rozebírá složení chyb v předpovědích neuronových sítí. Zbytek kapitoly je věnován technikám, které přeučení předcházejí.

3. kapitola popisuje implementaci předzpracování dat ve tvaru časové řady a tvorbu modelu neuronové sítě. Jsou zkoumány vlivy nastavení různých hyperparametrů sítě na úspěšnosti predikcí. Na konci kapitoly jsou shrnuty výsledky z těchto experimentů.

Kryptoměnové burzy jsou uzpůsobeny ke snadnému přístupu k datům a obchodování. Od svého vzniku se v kryptoměnách dlouho příliš neprojevovaly vlivy reálného světa, proto panovalo očekávání, že velká část informace pro predikci ceny je ukryta právě v jejím historickém vývoji.

Kapitola 1

Strojové učení

Arthur Samuel, otec umělé inteligence, v roce 1959 [5] nazval strojové učení jako oblast, která dává počítačům schopnost se učit bez toho, aby byly explicitně naprogramovány. Algoritmy se učí pomocí známých dat, která jim jsou předkládána, a upravují si podle nich parametry. Rozlišujeme mezi dvěma základními typy strojového učení – učení s učitelem a učení bez učitele.

Učení s učitelem spočívá v rozdělení dat na trénovací a testovací. Algoritmu jsou předkládána trénovací po tzv. vstupních vektorech (instance). Vstupní vektor (sample, instance) $\boldsymbol{x} = (x_1, x_2, \ldots x_n)$ obsahuje sadu vlastností (features) a je označen informaci y (label, target), kterou tento vstupní vektor charakterizuje. Trénovacím data setem $\mathfrak{X}$ označíme množinu dvojic $(\boldsymbol{x}, y)$, kde y je příslušná informace ke vstupnímu vektoru $\boldsymbol{x}$. V trénovací části nazýváme y pozorovaná hodnota (observed value). Na základě této spojitosti hledá algoritmus vztah mezi daty a jejich pozorovanými hodnotami. Testovací data již jsou bez označení a algoritmus promění své znalosti k předpovědi chybějící informace.

Naproti tomu učení bez učitele nerozděluje data na trénovací a testovací, ale cílem je přijít na nějakou vnitřní strukturu dat. Typickým příkladem je shlukování, při kterém dochází na dělení dat do podmnožin s podobnými vlastnostmi.

V jazyce Python je mnoho algoritmů pro strojové učení implementované v balíčku scikit-learn [7]. Obsahuje přímo modely strojového učení, software pro jejich optimalizaci a umožňuje zpracování dat. Další populární balíček Tensorflow [8], vyvíjený týmem Google Brain [9], nabízí podporu pro GPU procesory, které jsou vhodné pro trénování neuronových sítí. Nachází se v něm rozsáhlá knihovna neuronových sítí Keras, která velmi usnadňuje jejich implementaci. Na tyto balíčky se budu v textu odkazovat. Mezi další hojně používáné balíčky strojového učení patří Pytorch [12], vyvíjený týmem Facebook AI Research [13].

1.1 Klasifikace a regrese

Základními úlohami pro strojové učení je klasifikace a regrese. V případě klasifikačních úloh je pozorovanou hodnotou y reprezentována příslušnost vstupního vektoru $\boldsymbol{x}$ do určité třídy nebo kategorie. Problém lze chápat jako funkci s diskrétním oborem hodnot a úkolem algoritmu je tuto neznámou funkci aproximovat. Klasifikační úlohy se poté dělí na binární, multitřídové apod. podle toho, do kolika tříd vstupní vektor může patřit. Příklad binární klasifikace nalezneme v elektronické poště, kdy se algoritmus rozhoduje, zda příchozí email zařadí do spamu anebo ne. Naproti tomu v regresních úlohách algoritmus aproximuje funkci se spojitým oborem hodnot. Jedná se například o predikci vývoje burzovních kurzů [18].

Značení	Význam
n	Počet vlastností
θ_i	i-tý index váhy
$\hat{y}$	Predikce
x_i	Hodnota i-té vlastnosti
θ_0	Práh (bias)

Tabulka 1.1: Parametry modelu lin. reg.

1.2 Naučené parametry a hyperparametry

Je nutné rozlišovat mezi naučenými parametry modelu a jeho hyperparametry. Naučené parametry jsou nejčastěji k nalezení pod pojmem váhy (weights) a jejich hodnoty si obvykle model určuje sám v průběhu trénování. Po trénování tvoří kostru modelu a jsou používány pro tvorbu předpovědí. Hyperparametry jsou parametry, které musí být manuálně zadány při konstrukci modelu před trénováním. Ovlivňují efektivitu trénování, tvorbu naučených parametrů a mnoho dalšího. Jejich optimalizováním dosahuje model lepších výsledků. Volba správných hodnot vychází z typu úlohy a je do značné míry založena na experimentech a zkušenostech.

1.3 Lineární regrese

Lineární regrese je jednoduchý lineární model, který počítá své predikce pomocí vah jednotlivých vlastností 1.1,

$$\hat{y} = \theta_0 + \theta_1 \cdot x_1 + \ldots + \theta_n \cdot x_n, \tag{1.1}$$

anebo vektorově

$$\hat{y} = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\theta} \cdot \boldsymbol{x}. \tag{1.2}$$

Metoda je velmi jednoduchá, proto může vzniknout vysoká prahová chyba (bias error) vysvětlena v sekci 2.1. Využití metody najdeme od epidemiologie až po ekonomii. Cílem trénování je najít takové hodnoty vah, jež minimalizují ztrátovou funkci (další názvy v literatuře cost, penalty nebo objective function). Tento cíl je příznačný i pro ostatní algoritmy strojového učení. Pro lineární regresi je typickou ztrátovou funkcí střední kvadratická odchylka. Definujme ji pro data set $\mathfrak{X}$ s m vektory funkcí

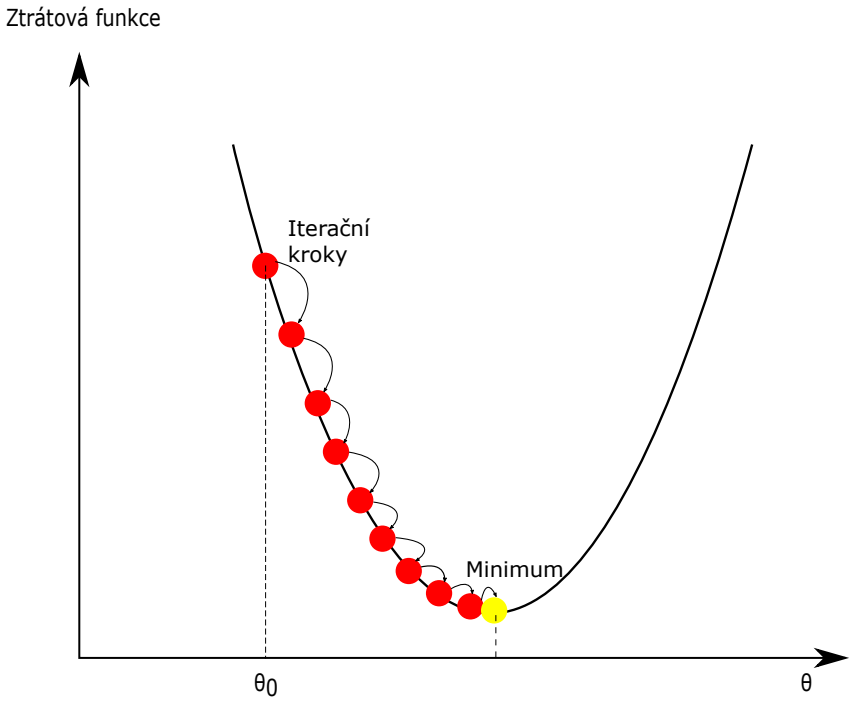
$$\mathrm{MSE}(\mathfrak{X}, h_{\boldsymbol{\theta}}) = \frac{1}{m} \sum_{i=1}^{m} (\boldsymbol{\theta} \cdot \boldsymbol{x}_i - y_i)^2 . \tag{1.3}$$

Obecné řešení je

$$\mathbf{X}^T \mathbf{X} \hat{\boldsymbol{\theta}} = \mathbf{X}^T \boldsymbol{y}, \tag{1.4}$$

kde $\hat{\boldsymbol{\theta}}$ značí vektor vah, který minimalizuje střední kvadratickou odchylku, $\mathbf{X}$ je matice se všemi vstupními vektory a $\boldsymbol{y}$ je vektor obsahující po složkách všechny pozorované hodnoty [2]. Je jednoznačné pouze, když $\mathbf{X}^T\mathbf{X}$ je regulární, jinak řešení nemusí být jednoznačné. V případě regularity je výpočet hodnot minimalizující ztrátovou funkci dán rovnicí

$$\hat{\boldsymbol{\theta}} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1} \mathbf{X}^T \cdot \boldsymbol{y}. \tag{1.5}$$



Obrázek 1.1: Hledání minima ztrátové funkce metodou gradientního sestupu

Jedná se o pouhé invertování matice, takže složitost tohoto algoritmu je nejvýše $O(n^3)$. V modulu `scikit-learn [19]` je výpočet $\hat{\boldsymbol{\theta}}$ implementován pomocí singulárního rozkladu a složitost tímto snížena na $O\left(m \cdot n^2\right)$.

1.4 Gradientní sestup

Dalším způsobem hledaní minima ztrátové funkce je algoritmus gradientní sestup (Gradient Descent), který je hojně používaný i v jiných modelech strojového učení. Myšlenkou tohoto algoritmu je spočítat hodnotu gradientu v náhodném bodě ztrátové funkce vzhledem k váhám a iterativně se posouvat proti směru gradientu do dalšího bodu, dokud gradient nebude rovný nule, jak je znázorněno na obrázku 1.1.

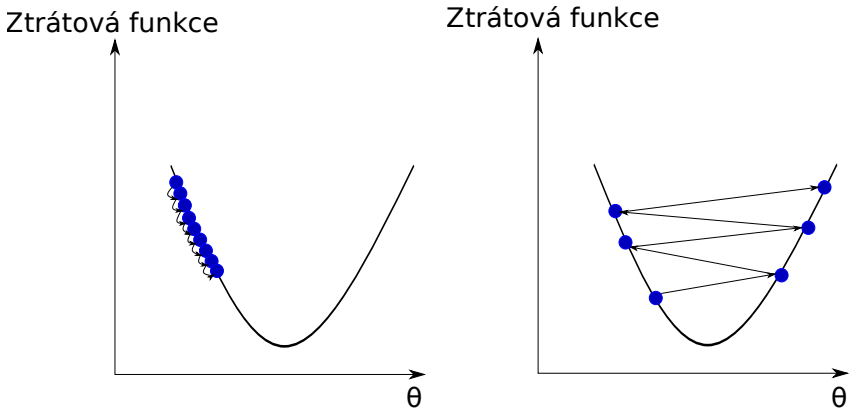
Demonstrujme to na lineární regresi. Pro gradient bude platit

$$\nabla_{\boldsymbol{\theta}} \mathrm{MSE}\left(\boldsymbol{\theta}\right) = \frac{2}{m} \sum_{i=1}^{m} \boldsymbol{x}_i^T \left(\boldsymbol{x}_i \cdot \boldsymbol{\theta} - y_i\right). \tag{1.6}$$

Velikost posunů je určena hyperparametrem zvaným síla učení (learning rate). Váhy $\boldsymbol{\theta}$ se změní iteračně dle

$$\boldsymbol{\theta}^{i+1} = \boldsymbol{\theta}^i - \eta \nabla_{\boldsymbol{\theta}} \mathrm{MSE}\left(\boldsymbol{\theta}\right), \tag{1.7}$$

kde η je síla učení. Když je jeho hodnota nízká, potom jsou jeho posuny příliš malé a trvá déle, než algoritmus zkonverguje. Naopak když je hodnota síly učení příliš vysoká, mohou posuny kolem minima oscilovat nebo se dokonce vzdalovat a algoritmus bude divergovat. Oba problémy jsou vizualizovány na obrázku 1.2. Dalším problémem gradientního sestupu mohou být sedlové body a lokální minima. Tento problém se řeší například pomocí metod založených na setrvačnosti nebo optimalizací různé síly učení [1]. V případě lineární regrese je tento problém zažehnán konvexností ztrátové funkce.



(a) Nízká hodnota síly učení (b) Příliš vysoká hodnota síly učení

Obrázek 1.2: Grafy chodu gradientního sestupu se špatně zvolenou silou učení

Způsob (1.7), při kterém se používají všechna trénovací data, se nazývá jednodávkový (batch) gradientní sestup. Dávkou (batch) rozumíme podmnožinu trénovacích dat, po které dojde k aktualizaci parametrů [14]. Od toho je odvozen důležitý hyperparametr zvaný velikost dávek (batch size), která říká, kolik jedna dávka obsahuje vstupních vektorů. V případě dávkového gradientního sestupu je velikost dávek rovna velikosti trénovacího data setu. Trénovací data mohou být v průběhu učení algoritmu předložena vícekrát. Jeden cyklus, při kterém jsou modelu předložena všechna trénovací data, se nazývá epocha a počet epoch je důležitým hyperparametrem. V tomto případě tedy dojde k úpravě vah právě za jednu epochu. Nevýhodou dávkového gradientního sestupu je, že při velmi velkém počtu vstupních dat zabere jeho výpočet mnoho času.

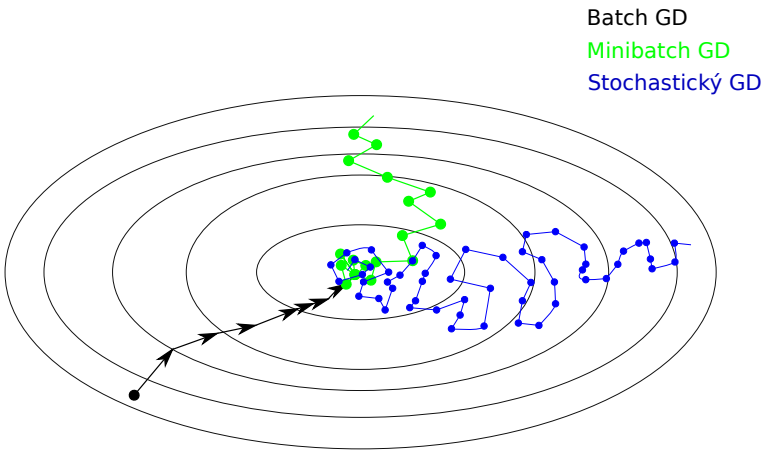
Další variantou je stochastický gradientní sestup, jenž funguje tak, že při každém kroku spočítá gradient na základě pouze jednoho náhodně vybraného vstupního vektoru (velikost dávky je rovna 1). Stochastický gradientní sestup má díky náhodnosti větší šanci neskončit v lokálním minimu. Tento způsob implementace klesá v průměru mnohem rychleji než dávkový gradientní sestup, ale od určitého blízkého okolí minima již nedokáže sestoupit níže a pohybuje se v tomto okolí. Optimalizuje se funkcí, která se nazývá učící časový plán (learning schedule). Ta má za úkol postupně vhodně snižovat učící čas. Tím se ze začátku zvýší šance, že algoritmus neskončí v lokálním minimu a pozdější nízká hodnota učícího času zaručí, že minimum nalezneme nebo budeme velmi blízko.

Kompromisem mezi stochastickým a dávkovým gradientním sestupem je mini-dávkový (mini-batch) gradientní sestup, při kterém se v každé iteraci vybere malá náhodná podmnožina trénovacích dat. Rozdíl mezi těmito variantami je vizualizován na obrázku 1.3 V případě neuronových sítí se používají další účinější optimalizace uvedených v podsekci (1.6.6).

1.5 Logistická regrese

Logistická regrese je složitější model a již dokáže zachytit v datech i jinou než lineární strukturu. Slouží na řešení klasifikačních i regresních úloh a je určena pro binární rozhodování, které probíhá pomocí rovnice

$$\hat{p} = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(X^T \Theta). \tag{1.8}$$



Obrázek 1.3: Porovnání variant gradientního sestupu

Symbol	Význam
$\boldsymbol{\theta}^{(k)}$	vektor vah třídy k
s_k	softmax skóre vstupního vektoru $\boldsymbol{x}$ pro třídu k
σ_k	pravděpodobnost příslušností k třídě k
$\boldsymbol{s}(\boldsymbol{x})$	vektor obsahující softmax skóre vstupního vektoru $\boldsymbol{x}$
n	počet tříd

Tabulka 1.2: Parametry modelu logistické regrese

Sigmoid funkce značena v (1.8) je normalizovaná exponenciela (1.9) a její obor hodnot je na celé reálné ose v intervalu $(0, 1)$.

$$\sigma(t) = \frac{1}{1 + \exp(-t)} \tag{1.9}$$

V modelu vyhodnocuje pravděpodobnost, s jakou vstupní vektor patří do třídy. Pro klasifikaci se jednoduše pokládáme předpovídanou třídu

$$\hat{y} = \begin{cases} 0 & \hat{p} < 0.5 \\ 1 & \hat{p} \geq 0.5 \end{cases}.$$

Ztrátová funkce je nazývaná binární křížová entropie (log loss) (1.10) a řeší se opět algoritmy gradientního sestupu, které díky konvexnosti funkce dosahují správných výsledků.

$$L_{\log} = -\frac{1}{m} \sum_{i=1}^{m} [y_i \log(\hat{p}_i) + (1 - y_i) \log(1 - \hat{p}_i) \tag{1.10}$$

Zobecnění logistcké regrese na multitřídovou klasifikaci softmax regrese. Je definovaná funkcí

$$\hat{p}_k = \sigma_k(s(\boldsymbol{x})) = \frac{\exp(s_k(\boldsymbol{x}))}{\sum_{i=1}^{n} \exp(s_j(\boldsymbol{x}))} \qquad \text{kde } s_k(\boldsymbol{x}) = \boldsymbol{x}^T \cdot \boldsymbol{\theta}_k \tag{1.11}$$

nazývanou softmax. V balíčku `scikit-learn` se mezi těmito regresemi přepíná hyperparametr `multi_class`, který se ve výchozím nastavení rozhoduje automaticky.

Ztrátovou funkcí je křížová entropie (1.10).

1.6 Umělé neuronové sítě

Jak název napovídá, umělé neuronové sítě mají hlubokou motivaci ve svých biologických protějšcích [1, 2]. Lidský nervový systém je tvořen neurony s výběžky, které jsou spojeny synapsemi. Stejně tak základní stavební jednotkou umělých neuronových sítích jsou neurony (node, perceptron, cell, unit). Ty jsou navzájem spojeny váhami a uloženy ve vrstvách (layers), které si navzájem předávají signál. První vrstva, do které data pouze vstupují a předá tento signál dál, se nazývá vstupní vrstva. Výstupní vrstvou je pojmenovaná poslední vrstva, ze které proudí výstup. Všechny ostatní vrstvy nacházející se mezi vstupní a výstupní vrstvou jsou nazvány skryté vrstvy. Jsou pojmenovány takto z faktu, že jejich výpočty zůstávají pro uživatele skryty a odehrávají se pouze uvnitř sítě.. Existuje mnoho architektur neuronových sítí, které mohou obsahovat více vrstev nebo pouze vrstvy vstupní a výstupní (nazývané jednovrstvé - single-layer, protože obsahují pouze jednu výpočetní vrstvu). Model neuronu ilustrujme na příkladě v následující podsekci 1.6.1.

Výhodou neuronových sítí oproti obvyklých algoritmů strojového učení je, že přesnost jejich předpovědí se vzrůstajícím počtem trénovacích dat roste rychleji [1]. Naproti tomu nevýhodou je náročnost na výpočetní výkon, která se však s moderními technologiemi snižuje.

1.6.1 Perceptron

Perceptron je nejjednodušší neuronovou sítí a slouží k binární klasifikaci. Pozorovaná hodnota y v tomto případě nabývá hodnot -1 a 1. Je tvořen pouze vstupní vrstvou a výstupním neuronem. Vstupní vrstva je tvořena neurony, které odpovídají počtu složek vstupního vektoru $\boldsymbol{x} = (x_1, x_2, \dots x_n)$ reprezentující určitou vlastnost. Každý neuron je spojen s výstupním neuronem a tyto spoje mají váhu w_i, která odpovídá důležitosti i-tévlastnosti. Všechny váhy, které do neuronu vstupují, označíme vektorem vah $\boldsymbol{w} = (w_1, \dots, w_n)$. V průběhu chodu algoritmu se vlastnosti vynásobí s odpovídající váhou, tyto součiny se sečtou a vstoupí do funkce nazývané aktivační funkce, která je nedílnou součástí neuronu. Detailněji rozebrané v podsekci 1.6.2. V případě perceptronu se jako aktivační funkce používá funkce signum (1.12), kde $\hat{y}$ je výsledná předpovídaná hodnota. Váha w_0 je práh a lze ji chápat jako váhu spoje neuronu s konstantní hodnotou $x_0 = -1$.

$$\hat{y} = \operatorname{sign}\left(\sum_{i=1}^{n} w_i \cdot x_i - w_0\right) \tag{1.12}$$

Vizualizaci nalezneme na obrázku 1.1.

K aktualizaci vah dochází, když se pozorovaná hodnota y neshoduje s predikcí $\hat{y}$ nebo-li když dojde ke špatné předpovědi. Trénování funguje po epochách. Při $i-$té neshodě, kdy prošel algoritmem vstupní vektor $\boldsymbol{x}_k$, se aktualizují vztahem

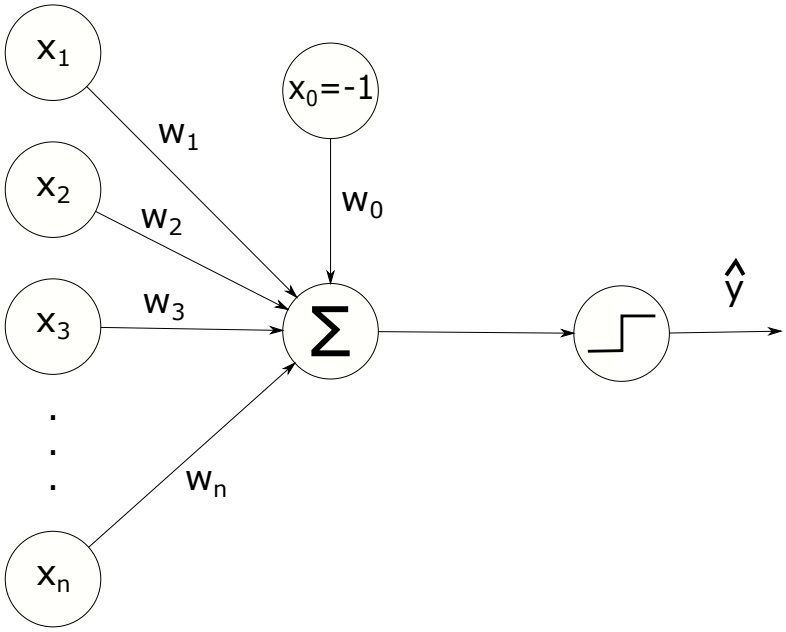
$$\boldsymbol{w}^{i+1} = \boldsymbol{w}^{i} - y_k \boldsymbol{x}_k. \tag{1.13}$$

$\sum_{i=0}^{n} w_i \cdot x_i$ definuje v prostoru $\mathbb{R}^{n+1}$ nadrovinu, která rozděluje tento prostor na množinu vstupních vektorů vyhodnocených v klasifikaci pozitivně ($y = 1$) a množinu vstupních vektorů vyhodnocených v klasifikaci negativně ($y = -1$). Za předpokladu lineárně separovatelných dat perceptron v konečně mnoha krocích konverguje. To lze potvrdit následující větou 1 uvedené v [6].

Věta 1. *Mějme data set s m dvojicemi* $(\boldsymbol{x}_k, y_k)$, $k \in \{1, \dots m\}$. *Nechť existuje vektor vah* $\boldsymbol{w}$ *takový, že* $\|\boldsymbol{w}\|_2 = 1$ *(euklidovská metrika), a nějaké* $\gamma > 0$ *takové, že pro všechny k platí*

$$y_k \boldsymbol{x}_k \cdot \boldsymbol{w} \geq \gamma. \tag{1.14}$$

Navíc nechť pro všechny $k = 1, 2, \dots m$ *platí* $\|\boldsymbol{x}_k\|_2 \leq R$ *. Potom perceptron udělá při trénování maximálně* $\frac{R^2}{\gamma^2}$ *špatných předpovědí.*



Obrázek 1.4: Perceptron

Důkaz. Označme $\boldsymbol{w}^i$ hodnotu váhy, která je aktuální před tím, než algoritmus udělá $i-$tou chybu. Položme $\boldsymbol{w}^1 = 0$. Dále předpokládejme, že ke $i-$té chybě dojde u prvku k.

$$\boldsymbol{w}^{i+1} \cdot \boldsymbol{w} = \left(\boldsymbol{w}^i + y_k \boldsymbol{x}_k\right) \cdot \boldsymbol{w} = \boldsymbol{w}^i \cdot \boldsymbol{w} + y_k \boldsymbol{x}_k \cdot \boldsymbol{w} \geq \boldsymbol{w}^i \cdot \boldsymbol{w} + \gamma \tag{1.15}$$

V rovnici (1.15) jsme využili předpokladu (1.14). Algoritmus funguje po epochách, proto vektor $\boldsymbol{x}_k$ mohl nastavováním vah projít vícekrát. Když budeme aplikací pro $k = 1, 2, \ldots, l$ 1.15 pokračovat dále v odhadech dostaneme

$$\boldsymbol{w}^{l+1} \cdot \boldsymbol{w} \geq l\gamma. \tag{1.16}$$

Dále proveďme odhad, kde využijeme, že velikosti vstupních vektorů jsou omezeny konstantou R, a také toho že při $\boldsymbol{w}^i$ došlo k chybě, a proto součin $y_k \boldsymbol{x}_k \cdot \boldsymbol{w}^i$ je menší než 0.

$$||\boldsymbol{w}^{i+1}||^2 = ||\boldsymbol{w}^i + y_k \cdot \boldsymbol{x}_k||^2 = ||\boldsymbol{w}^i||^2 + y_k^2 ||\boldsymbol{x}_k||^2 + 2 y_k \boldsymbol{x}_k \cdot \boldsymbol{w}^i \leq ||\boldsymbol{w}^i||^2 + R^2 \tag{1.17}$$

Když budeme opět pokračovat v odhadech, dojdeme k nerovnici

$$||\boldsymbol{w}^{i+1}||^2 \leq iR^2. \tag{1.18}$$

Kombinací odhadů (1.16) a (1.18) dojdeme k

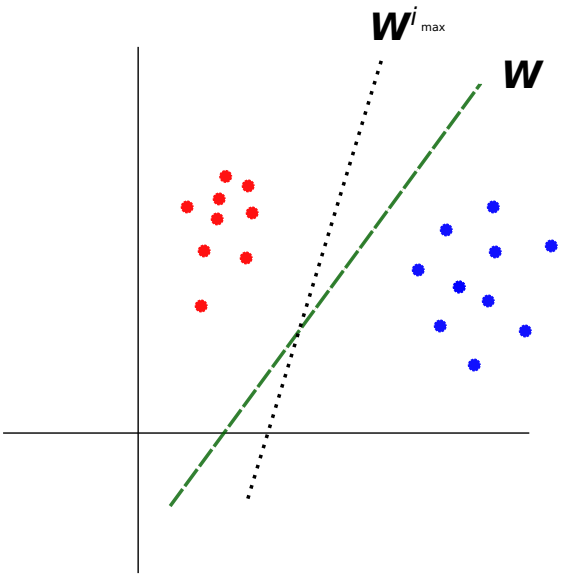
$$i^2\gamma^2 \leq ||\boldsymbol{w}^{i+1}||^2 \leq iR^2. \tag{1.19}$$

Úpravou plyne vztah

$$i \leq \frac{R^2}{\gamma^2}. \tag{1.20}$$

To dokazuje, že počet chyb nepřesáhne konstantu $\frac{R^2}{\gamma^2}$ ze znění věty. Při trénování dojde ke konečně mnoho chybám a po určitém počtu epoch jsou všechna trénovací data klasifikována správně. Výsledná váha, ke které algoritmus dospěje, je znázorněna na obrázku 1.5. □

Popsaný algoritmus vede k 100% úspěšnosti modelu na trénovacích datech. Perceptron tedy funguje pro binární problémy na lineárně separovatelných datech, ale na složitější problémy je nutné použít vícevrstvé sítě se složitějšími aktivačními a ztrátovými funkcemi.



Obrázek 1.5: Lineárně separovaná data rozdělená nadrovinami tvořenými vahou $\boldsymbol{w}$ z předpokladu věty a vahou $\boldsymbol{w}^{i_{max}}$, ke které algoritmus dokonvergoval.

1.6.2 Aktivační funkce

Jedná se o transformační funkci, kterou neuron zpracuje skalární součin vah se s vstupujícím signálem a předá výstupní signál dál. Obecně ji budeme dále značit řeckým písmenem Φ. Volbami této funkce lze napodobit jiné modely strojového učení jednoduchou neuronovou sítí [1]. Každá vrstva neuronové sítě může obsahovat jiný typ aktivační funkce. Jedná se o volbu velmi důležitou pro návrh neuronové sítě a zvláště pro výstupní vrstvu je její volba ovlivněna typem úlohy. Volbami této funkce lze napodobit jiné modely strojového učení jednoduchou neuronovou sítí. V průběhu bakalářská práce bylo již uvedeno několik příkladů aktivačních funkcí, jejichž seznam se s neuronovými sítěmi rozšíří:

- Identita

  $$\Phi(t) = t$$

  Je de facto použita u lineární regrese v sekci 1.3. Není v neuronových sítích nebývá ideální volbou viz věta 2.

- Sigmoid

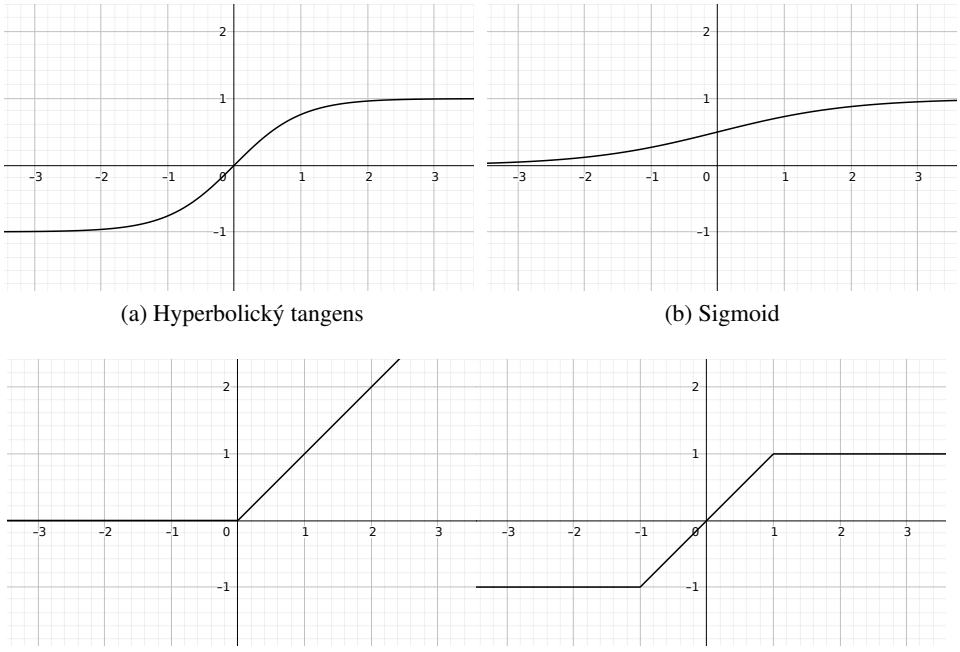
  Je znázorněna na obrázku 1.6b. Použita byla v sekci logistické regrese 1.5. V případě regresních úloh se její uplatnění ve výstupní vrstvě nachází, když je požadováno výstup udržet v omezeném intervalu. Ovšem ve výstupní vrstvě se nejčastěji objeví při klasifikačních úlohách a stejně jako v případě logistické regrese je při více třídových klasifikacích nahrazena funkcí softmax.

- Signum

  Použita u perceptronu v podsekci 1.6.1.

- ReLU

  Je znázorněn na obrázku 1.6c. Jedná se o upravenou lineární funkci vztahem



(a) Hyperbolický tangens (b) Sigmoid

(c) ReLU (d) Linearizovaný hyperbolický tangens

Obrázek 1.6: Aktivační funkce

$$\Phi\,(\mathrm{t}) = \mathrm{ReLU}\,(t) := \begin{cases} 0 & t < 0, \\ t & t \geq 0. \end{cases} \tag{1.21}$$

Velmi často se vyskytuje ve skrytých vrstvách a to z důvodu mizejícího gradientu viz podsekce (1.6.5). Ve výstupní vrstvě se nachází pouze v případě regresních úloh, kdy požadujeme pouze kladné hodnoty.

- Hyperbolický tangens

  Je znázorněn na obrázku 1.6a. A jeho rovnice je

$$\Phi\,(t) = \tanh\,(t)\,. \tag{1.22}$$

- Linearizovaný hyperbolický tangens

  Je znázorněn na obrázku 1.6d a je definovaný rovnicí

$$\Phi\,(t) = \mathrm{htanh}\,(t) = \begin{cases} -1 & t < -1, \\ t & -1 \leq t \leq 1, \\ 1 & t > 1. \end{cases} \tag{1.23}$$

1.6.3 Vícevrstvé (hluboké) dopředné (feed-forward) sítě

Vícevrstvé neuronové sítě obsahují více než jednu výpočetní vrstvu. V základním architektuře je každý neuron jedné vrstvy spojen se všemi neurony následující vrstvy (plně propojená neuronová síť)

Značení	Význam
$\boldsymbol{x}$	Vstupní vektor
x_i	i-tá složka vstupního vektoru
h_{ij}	Výstup j-tého neuronu v i-té skryté vrstvě
$\boldsymbol{h}_i$	Výstup i-té skryté vrstvy
$\boldsymbol{o}$	Výstup z výstupní vrstvy
k	Počet skrytých vrstev
$\mathbf{W}_i$	Matice obsahující váhy mezi i-tou a (i+1)-ní vrstvou
p_i	dimenze i-té vrstvy

Tabulka 1.3: Přehled značení ve vícevrstvé dopředné neuronové síti

[1]. Dopředné jsou navíc ty sítě, ve kterých data proudí v jednom směru od vstupní vrstvy až po výstupní vrstvu. Dalšími důležitými hyperparametry neuronové sítě tedy jsou počet vrstev a počet neuronů v každé vrstvě. K výstupní vrstvě je dále přidružena ztrátová funkce, její volba rovněž jako aktivační funkce závisí na typu úlohy. Pro předpovídání spojitých hodnot se hojně používá střední kvadratická odchylka, zatímco pro diskrétní předpovědi je preferovaná křížová entropie.

Proces, jakým způsobem ve vícevrstvé dopředné neuronové síti proudí data, znázorňují rovnice

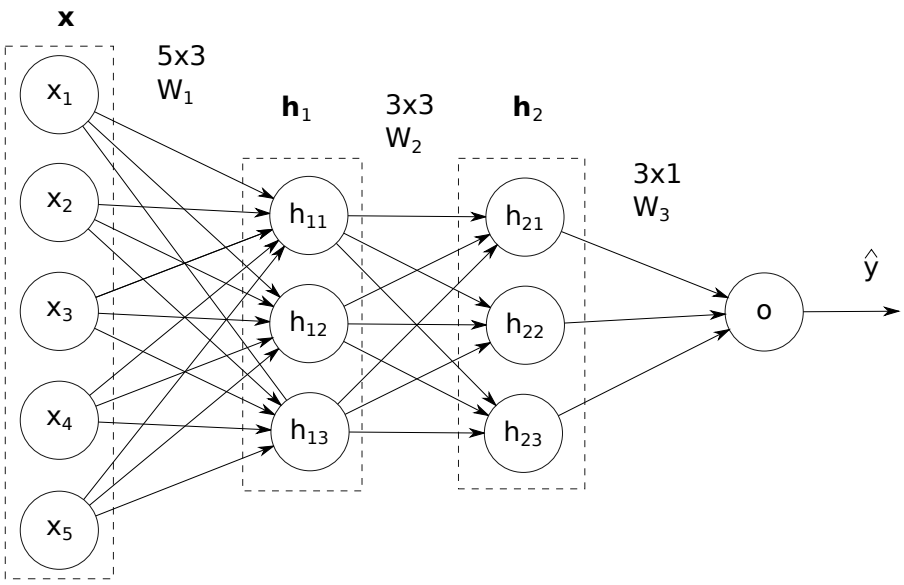
$$\begin{aligned} \boldsymbol{h}_1 &= \boldsymbol{\Phi}\left(\mathbf{W}_1^T \cdot \boldsymbol{x}\right) \\ \boldsymbol{h}_{s+1} &= \boldsymbol{\Phi}\left(\mathbf{W}_{s+1}^T \cdot \boldsymbol{h}_s\right) = \mathbf{W}_{s+1}^T \cdot \boldsymbol{h}_s \qquad \forall s \in \{1 \ldots k-1\} \\ \boldsymbol{o} &= \boldsymbol{\Phi}\left(\mathbf{W}_{k+1}^T \boldsymbol{h}_k\right), \end{aligned} \tag{1.24}$$

přičemž výrazem $\boldsymbol{\Phi}$ v (1.24) rozumíme

$$\boldsymbol{\Phi}(\boldsymbol{x}) = \begin{pmatrix} \Phi(x_0) \\ \vdots \\ \Phi(x_n) \end{pmatrix},$$

tj. aktivační funkce působí po složkách. Průběh rovnic (1.24)je graficky ilustrován na obrázku 1.7. Značení je vysvětleno v tabulce 1.3. Jak již bylo uvedeno v podsekci o perceptronu 1.6.1, spojení mezi neurony je popsáno vahou (pro jednoduchost značení nebudeme brát v potaz práh). Všechny spoje, které vedou do jednoho neuronu z neuronů předchozí vrstvy, jsou popsány vektorem vah. V případě neuronových sítí reprezentuje přechod mezi vrstvami matice $\mathbf{W}_i$ tvořena právě všemi vektory vah, které se nachází mezi i-tou a $(i+1)$-ní vrstvou. Její dimenze je rovna $p_i \times p_{i+1}$, kde p_i odpovídá počtu neuronů v i-té výpočetní vrstvě. O $p_i$se hovoří jako o dimenzi i-té vrstvy. Na obrázku 1.7 se vstupní vektor dimenze 5 vynásobí s maticí vah $\mathbf{W}_1^T$ dimenze 5×3, výsledný vektor vstoupí do aktivační funkce, ze které vyjde výstupní vektor $\boldsymbol{h}_1$ 1. skryté vrstvy dimenze 3. V dalších vrstvách se opakuje stejný postup s výstupními vektory skrytých vrstev a s maticemi $\mathbf{W}_2^T$ dimenze 3×3 a $\mathbf{W}_3^T$dimenze 3×1 s konečným výstupem o.

Úspěšnost neuronových sítí zvyšuje použití nelineární aktivační funkce. Kdyby všechny vrstvy používaly jako aktivační funkci identitu, potom i s přihlédnutím k faktu, že skládání lineárních transformací je stále lineární transformace, se neuronová síť zjednoduší na lineární regresi [1].



Obrázek 1.7: Třívrstvá dopředná plně propojená neuronová síť demonstrovaná s výstupní vrstvou o délce $d = 1$

Věta 2. *Vícevrstvá neuronová síť, která používá jako aktivační funkce ve všech svých vrstvách pouze identity, funguje jako jednovrstvá síť vykonávající lineární regresi.*

Důkaz. Nechť má neuronová síť k skrytých vrstev. Mějme matice vah $\mathbf{W}_1, \ldots, \mathbf{W}_{k+1}$spojující po sobě jdoucí vrstvy. Buď $\boldsymbol{x}$ vstupní vektor, $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_k$ vektory odpovídající výstupům ze skrytých vrstev a $\boldsymbol{o}$ výstupní vektor dimenze m odpovídající počtu výstupů. Ve vztahu (1.24) máme podle předpokladu všechny aktivační funkce identity. To znamená, že

$$\boldsymbol{h}_1 = \boldsymbol{\Phi}\left(\mathbf{W}_1^T \cdot \boldsymbol{x}\right) = \mathbf{W}_1^T \cdot \boldsymbol{x}, \tag{1.25}$$

$$\boldsymbol{h}_{s+1} = \boldsymbol{\Phi}\left(\mathbf{W}_{s+1}^T \cdot \boldsymbol{h}_s\right) = \mathbf{W}_{s+1}^T \cdot \boldsymbol{h}_s \qquad \forall s \in \{1 \ldots k-1\}, \tag{1.26}$$

$$\boldsymbol{o} = \boldsymbol{\Phi}\left(\mathbf{W}_{k+1}^T \cdot \boldsymbol{h}_k\right) = \mathbf{W}_{k+1}^T \cdot \boldsymbol{h}_k. \tag{1.27}$$

Po vyjádření (1.27) pomocí rekurentního vztahu (1.26) získáme

$$\boldsymbol{o} = \mathbf{W}_{k+1}^T \ldots \mathbf{W}_1^T \cdot \boldsymbol{x} = (\mathbf{W}_1 \ldots \mathbf{W}_{k+1})^T \cdot \boldsymbol{x}. \tag{1.28}$$

Součin matic $\mathbf{W}_1 \ldots \mathbf{W}_{k+1}$ je $m \times d$ rozměrná matice, kterou označíme $\mathbf{W}_{xo}$. Tím můžeme vzorec pro výstup nahradit $\boldsymbol{o} = \mathbf{W}_{xo}^T \cdot \boldsymbol{x}$ a to je po složkách přímo vzorec pro lineární regresi (viz sekce 1.3), která má navíc mnohem méně učících parametrů než původní model. □

Naopak při použití nelineárních aktivačních funkcí stačí pouze jedna skrytá vrstva k tomu, aby neuronová síť dokázala aproximovat jakoukoli měřitelnou funkci libovolně přesně. Důkaz je poněkud náročnější a je uveden v [15].

Hlubším neuronovým sítím postačí méně neuronů a s tím spojených parametrů nutných pro aproximaci komplexnějších funkcí. Počet parametrů k tomu potřebných dokonce klesá exponenciálně s počtem skrytých vrstev. To je zdůvodněno počtem cest, které se objeví v grafu ve vícevrstvých sítích [1]. Hlubší neuronové sítě dokáží lépe nalézt nějaké opakující se vzory v trénovacích datech na různých úrovních abstrakce, které pak uplatní na datech testovacích. V prvních vrstvách jsou schopny odhalit detailnější

vzory, v případě úlohy s rozpoznáváním obrazů to jsou například hrany, v pozdějších vrstvách již dokáží odhalit daleko vyšší vzory, jako jsou pozice objektů apod. Díky snížení počtu parametrů je menší riziko, že síť si data jen zapamatuje (přeučení 2.1) a dokáže je lépe generalizovat. Mezi nevýhody hlubších neuronových sítí naopak patří problém mizejícího a vybuchujícího gradient v podsekci 1.6.5.

1.6.4 Algoritmus zpětného šíření (Backpropagation algorithm)

Jako ve většině algoritmů strojového učení i v neuronových sítí se při učení používají metody gradientního sestupu. Ve vícevrstvých sítí je výpočet gradientu ztrátové funkce složitější. Pro jeho řešení v dopředných sítí se používá dynamicky implementovaný algoritmus zpětného šíření. Skládá se ze dvou fází - dopředné a zpětné. V dopředné fázi se přivádí do neuronové sítě vektory vstupních dat, které projdou jednotlivými výpočty vrstev s aktuálními váhami. Za daného vstupu se spočítají hodnoty všech výstupů skrytých vrstev a poslední vrstvy. Pro úpravu vah se musí vypočítat gradient ztrátové funkce podle všech vah, a tomu se děje ve zpětné fázi.

Mějme $k+1$ vrstev, $0-$tá vrstva je vstupní, $k-$tá vrstva je výstupní. Rovnice

$$h_{i_r}^{(r)} = \overbrace{\phi^{(r)}\left(\sum_{i_{r-1}} w_{i_r i_{r-1}}^{(r)} h_{i_{r-1}}^{(r-1)}\right)}^{\phi_{i_r}^{(r)}} \tag{1.29}$$

vyjadřuje výstup i_r-tého neuronu v $r-$té vrstvě. Rovnici zpětného šíření zkonstruujeme nejdříve pro neuronovou síť, která má 4 vrstvy. Na $\frac{\partial L}{\partial w_{ij}^{(r)}}$, kde $w_{ij}^{(r)}$ značí člen v r-té matici vah, aplikujme řetězové pravidlo. Nechť $r \in \{1,\ 2\}$. Potom

$$\frac{\partial L}{\partial w_{i_r i_{r-1}}^{(r)}} = \sum_{i_3} \frac{\partial L}{\partial h_{i_3}^{(3)}} \cdot \frac{\partial h_{i_3}^{(3)}}{\partial w_{i_r i_{r-1}}^{(r)}} = \sum_{i_3} \frac{\partial L}{\partial h_{i_3}^{(3)}} \cdot \phi'^{(3)}_{i_3} \sum_{i_2} w_{i_3 i_2}^{(3)} \frac{\partial h_{i_2}^{(2)}}{\partial w_{i_r i_{r-1}}^{(r)}}. \tag{1.30}$$

Pro $r = 2$ jsme u konce, pro $r = 1$ lze analogicky pokračovat v úpravách posledního členu $\frac{\partial h_{i_2}^{(2)}}{\partial w_{i_r i_{r-1}}^{(2)}}$. Po zobecnění dospějeme ke vztahu

$$\frac{\partial L}{\partial w_{i_r i_{r-1}}^{(r)}} = \sum_{i_k} \sum_{i_{k-1}} \cdots \sum_{i_{r+1}} \prod_{l=r+1}^{k} \phi'^{(l)}_{i_l} \cdot w_{i_l i_{l-1}}^{(l)} \cdot \frac{\partial h_{i_r}^{(r)}}{\partial w_{i_l i_{l-1}}^{(r)}}. \tag{1.31}$$

Počet členu v sumách roste exponenciálně, proto je výpočet v rovnici (1.31) implementován pomocí rekurentního vztahu, čímž se zamezí exponenciální složitosti výpočtu. Rekurentní vztah získáme následovně

$$\frac{\partial L}{\partial w_{i_r i_{r-1}}^{(r)}} = \sum_{i_{r+1}} \overbrace{\sum_{i_k} \sum_{i_{k-1}} \cdots \sum_{i_{r+2}} \prod_{l=r+2}^{k} \phi'^{(l)}_{i_l} w_{i_l i_{l-1}}^{(l)}}^{\frac{\partial L}{\partial h_{i_{r+1}}^{(r+1)}}} \phi'^{(r+1)}_{i_{r+1}} w_{i_{r+1} i_r}^{(r+1)} \frac{\partial h_{i_r}^{(r)}}{\partial w_{i_r i_{r-1}}^{(r)}}, \tag{1.32}$$

kde část označena jako $\frac{\partial L}{\partial h_{i_{r+1}}^{(r+1)}}$ byla již vypočtena k předchozím váhám. Po výpočtu parciálních derivací se aplikuje jeden z typů gradientního sestupu uvedených v sekci 1.4 nebo 1.6.6, kterým aktualizujeme všechny váhy.

1.6.5 Mizející a vybuchující (vanishing and exploding) gradient

Tyto problémy se objevují při trénování a jsou spojeny se sítěmi s mnoha vrstvami. Pro znázornění problému zkoumejme neuronovou síť, která v každé své vrstvě obsahuje pouze jeden neuron s aktivační funkcí sigmoid (1.9). Derivace aktivační funkce je

$$\dot{\Phi}(t) = \frac{\exp(-t)}{(1+\exp(-t))^2} = \Phi(t)\cdot(1-\Phi(t)) \tag{1.33}$$

a má maximum $0,25$. Předpokládejme že váhy se inicializují podle standardního rozdělení a nabývají hodnot $w \in (0,\ 1)$. Potom dle (1.32) je hodnota $\frac{\partial L}{\partial h^{(r)}_{i_r}}$ menší než $0,25 \cdot \frac{\partial L}{\partial h^{(r+1)}_{i_{r+1}}}$. Když bude síť dostatečně hluboká, tak již po 15 rekurentně dopočítaných vrstvách bude ona hodnota menší než $0,25^{15}$. Aktualizace vah v dřívějších vrstvách jsou tedy ve srovnání s aktualizacemi vah v posledních vrstvách velmi malé, a proto tento problém nese jméno mizející gradient. Jedním z řešení může být inicializace vah s vyššími hodnotami anebo použití aktivační funkce, jejichž derivace nabývají vyšších hodnot. To ale může způsobit přesně opačný problém, kdy hodnoty vah v dřívějších vrstvách budou aktualizovány v porovnání s těmi v posledních vrstvách s mnohem vyšší hodnotou, a proto tento problém nazýváme vybuchující gradient. Princip, jak k těmto problémům dochází, lze zobecnit i na ostatní konstrukce neuronových sítí [1].

1.6.6 Varianty trénovacího algoritmu založené na gradientním sestupu

V případě neuronových sítí je z výše vyjmenovaných algoritmů gradientního sestupu v sekci 1.4 nejpoužívanější mini-batch, který bývá optimalizován následujícími způsoby.

1.6.6.1 Setrvační (momentum) optimalizace

Motivací setrvační optimalizace je, že je do aktualizace vah navíc zahrnut vektorový faktor $\boldsymbol{v}$, který zdůrazní správný směr sestupu a algoritmus zkonverguje rychleji. Název se odvíjí od toho, že tato optimalizace udílí sestupu „setrvačnost“, díky které algoritmus neuvízne v lokálních minimech. Tento faktor je dále ovlivněn hyperparametrem $\beta \in (0,\ 1)$ nazývaným setrvační parametr (momentum parameter nebo friction parameter), který se při nízkých hodnot chová jako „brzda“ sestupu. Váhy se aktualizují následujícím dvoukrokovým způsobem

$$\begin{aligned} \boldsymbol{v}^{i+1} &= \beta \boldsymbol{v}^i - \eta \nabla_{\boldsymbol{w}^i} L \\ \boldsymbol{w}^{i+1} &= \boldsymbol{w}^i + \boldsymbol{v}^{i+1}. \end{aligned} \tag{1.34}$$

Jako vhodná volba hodnoty β se uvádí $0,2$ [2].

1.6.6.2 Adam

Jedná se o kombinaci dvou procesů, a to škálování pomocí A_k a setrvačních optimalizací F_k patřícím k váze w_k v podobě rovnic

Značení	Význam	Obvyklá hodnota [2]
$\beta_1 \in (0,\ 1)$	Škálovací hyperparametr	$0,999$
$\beta_2 \in (0,\ 1)$	Setrvační hyperparametr	$0,9$
$\epsilon > 0$	Zabrání 0 ve jmenovateli rovnice (1.37)	Velmi nízká kladná čísla (10^{-7})

Tabulka 1.4: Hyperparametry optimalizace Adam

$$A_k^{i+1} = \beta_1 A_k^i + (1-\beta_1)\left(\frac{\partial L}{\partial w_k}\right)^2 \quad \text{pro } \forall k, \tag{1.35}$$

$$F_k^{i+1} = \beta_2 F_k^i + (1-\beta_2)\left(\frac{\partial L}{\partial w_k}\right)^2 \quad \text{pro } \forall k, \tag{1.36}$$

$$w_k^{i+1} = w_k^i - \frac{\alpha^i}{\sqrt{A_k^{i+1} + \epsilon}} F_k^{i+1} \quad \text{pro } \forall k, \tag{1.37}$$

kde síla učení α^i závisí na iteraci a je definována

$$\alpha^i = \alpha \cdot \left(\frac{\sqrt{1-\beta_1^i}}{1-\beta_2^i}\right). \tag{1.38}$$

Ostatní symboly jsou popsány v tabulce 1.4. Síla učení α^i postupně konverguje k α. Inicializace hodnot A_k^0 a F_k^0 je rovna 0 [1, 2]. Jedná se o jednu z nejčastěji používaných optimalizací gradientního sestupu [1]. Setrvační část zrychluje konvergenci a škálování zvyšuje přesnost konvergence. Přesto v některých případech může fungovat lépe setrvační optimalizace [2].

1.6.7 Rekurentní sítě

Pro různé typy data setů se mohou stát výhodnějšími jiné architektury neuronových sítí než dopředné neuronové sítě. Dobrými příklady jsou časové řady, což jsou posloupnosti hodnot indexovaných podle času. Obvykle se v jejich členech nachází nějaká závislost toho, jaká data po sobě následují. Proto se jeví důležité, aby to dokázala zohlednit i neuronová síť. S touto myšlenkou byly vytvořeny rekurentní sítě [1].

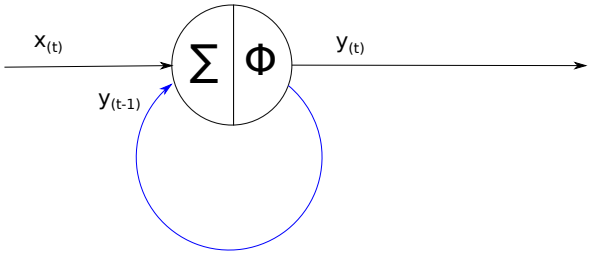
Oproti neuronu dopředné sítě do rekurentních neuronů vstupují i jejich minulé výstupy. Model neuronu jednoduché rekurentní sítě je znázorněn na obrázku 1.8. Indexem (t) se značí vstupy a výstupy pro člen časové řady v čase t. Pro výpočet výstupu $y_{(t)}$ do neuronu vstoupí vstupní vektor $\boldsymbol{x}_{(t)}$ a výstupní vektor $\boldsymbol{y}_{(t-1)}$ z předešlého času.

Tímto způsobem rekurentní neuron zohledňuje minulé členy. Celý proces se řídí podle rovnice

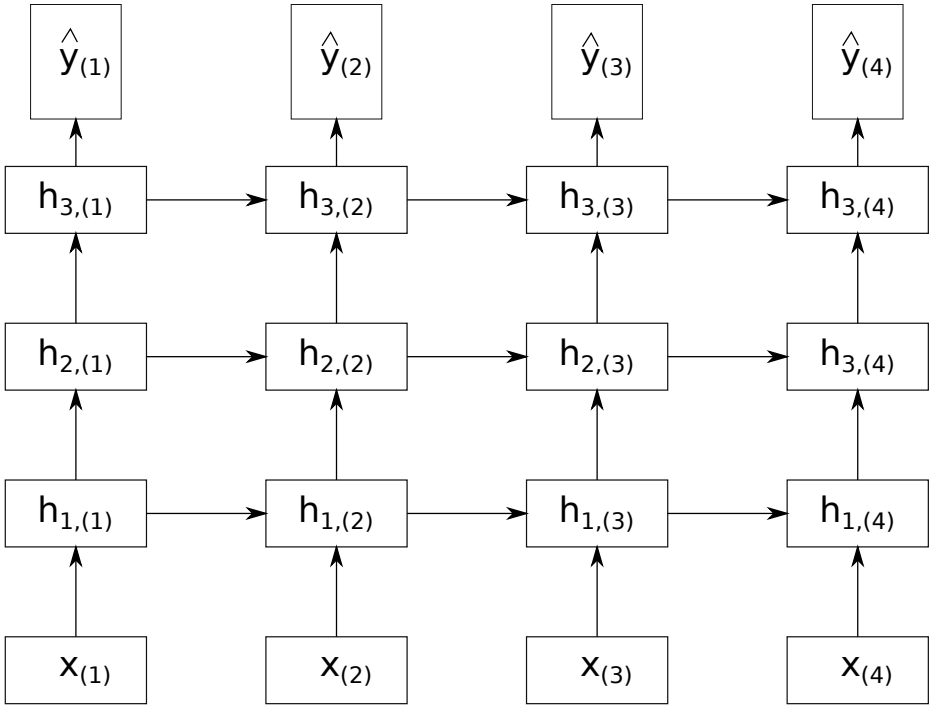
$$y_{(t)} = \Phi\left(\boldsymbol{w}_x \cdot \boldsymbol{x}_{(t)} + \boldsymbol{w}_y \cdot \boldsymbol{y}_{(t-1)}\right), \tag{1.39}$$

kde $\boldsymbol{w}_x$ je vektor obsahující váhy násobené s vektorem $\boldsymbol{x}_{(t)}$ a $\boldsymbol{w}_y$ je vektor obsahující váhy násobené s vektorem $\boldsymbol{y}_{(t-1)}$. Pokud výstup $\boldsymbol{y}_{(t-1)}$ neexistuje (například pro první vstupní vektor), dosadí se za něj 0 [2]. Ve vícevrstvých rekurentních sítích lze rovnici (1.39) zobecnit pro k-tou vrstvu na

$$\boldsymbol{h}_{k,(t)} = \boldsymbol{\Phi}\left(\mathbf{W}_k^T \cdot \boldsymbol{h}_{k-1,(t)} + \mathbf{W}_{k,h}^T \cdot \boldsymbol{h}_{k,(t-1)}\right), \tag{1.40}$$



Obrázek 1.8: Model neuronu jednoduché rekurentní neuronové sítě



Obrázek 1.9: Časově rozvinutý graf rekurentní neuronové sítě o 3 skrytých vrstvách a 4 časových krocích

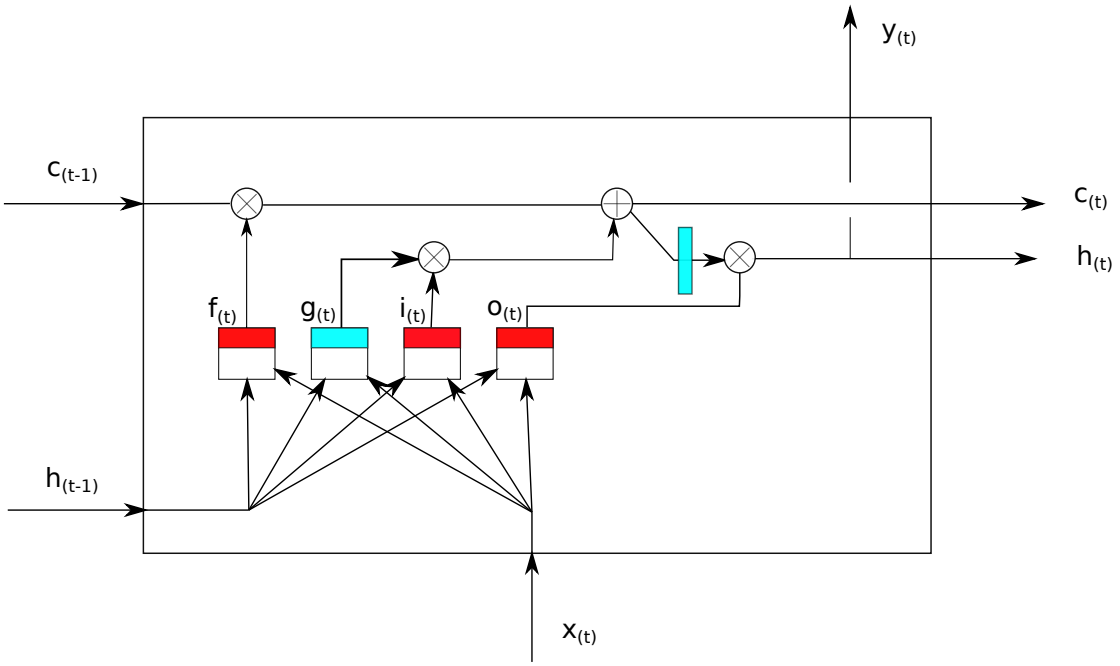
kde $\mathbf{W}_k$ je matice dimenze (počet neuronů v $(k-1)$-té vrstvě)×(počet neuronů v k-té vrstvě) obsahující váhy pro vstup z předešlé vrstvy v současném časovém t a $\mathbf{W}_{k,h}$ je matice stejné dimenze vyjadřující rekurentní vztah, protože obsahuje váhy působící na výstupy k-té vrstvy z předešlého časového kroku $t-1$. Rekurentní neuronovou síť je možno vizualizovat vícero způsoby. Jeden z nich je na obrázku 1.9, kde jsou rekurentní vztahy vyznačeny horizontálními šipkami. Toto schéma nazýváme časově rozvinutý graf rekurentní neuronové sítě.

Vztahy mezi neurony jsou složitější než u dopředných neuronových sítí, a proto se při učení používá upravený zpětného šíření zvaný časový algoritmus zpětného šíření (backpropagation through time). Jedná se v podstatě o aplikaci normálního algoritmu zpětného šíření na časově rozvinutý graf rekurentní neuronové sítě [2]. Tento typ architektury je náchylný na problém mizejícího a explodujícího gradientu.

Vstupy a výstupy mohou být více typů [2]:

- Sekvence-sekvence

  Tento typ je znázorněn na obrázku 1.9. Vstupní vektory $\boldsymbol{x}_{(1)}, \cdots, \boldsymbol{x}_{(4)}$ tvoří sekvenci, ze kterých se vytvoří sekvence $\hat{y}_{(1)}, \cdots, \hat{y}_{(4)}$. Je vhodný například pro předpovědi vývoje cen na burzách a mimo časové řady i pro textové data sety.



Obrázek 1.10: LSTM buňka. Barevné obdélníky jsou označení pro aktivační funkce. Blankytně modrá barva značí hyperbolický tangens a červená barva sigmoid funkci.

- Sekvence-vektor

  Rozdíl oproti předchozí případu bude pouze v tom, že výstup by byl jediný a to $\hat{y}_{(4)}$.

- Vektor-sekvence

  Do neuronové sítě by na obrázku 1.9 vstoupil pouze jeden vstupní vektor $\boldsymbol{x}_{(1)}$.

- Kóder-dekóder (Encoder-Decoder)

  Jedná se o spojení předchozích dvou typů. V tomto případě je neuronová síť rozdělena na dvě částí. První část neuronové sítě bude ve stylu sekvence-vektor a druhá část vektor-sekvence. Sekvence se zakóduje do jednoho vektoru, který je poté rozkódován do sekvence. Když bychom opět chtěli uvést příklad pomocí obrázku 1.9, odlišovat se bude v tom, že jediné vstupy budou $\boldsymbol{x}_1$, $\boldsymbol{x}_2$ a výstupy $\hat{y}_3$, $\hat{y}_4$. Častou úlohou pro tento typ bývá překládání slov [16].

1.6.7.1 LSTM

Původní koncept rekurentních sítí dává důraz především na výstupy z nedávné minulosti a informace, kterou neuronová síť získala dříve, se pomalu vytrácela. To vedlo k tomu, že neuronová síť nemohla dobře odhalit sekvenciálně delší vzory v datech a fungovala s krátkodobou pamětí (short memory). Z téhle motivace vznikl model LSTM (Long-Short Term Memory) buňky (název pro neuron v kontextu LSTM sítí), který má dlouhodobou paměť pro sekvenciálně delší vzory a také si ponechává vlastnosti obyčejných rekurentních sítí pro krátkodobou paměť.

Konstrukce buňky je na obrázku 1.10. Významy symbolů jsou v tabulce 1.5. Do buňky vstupují 3 vektory. Vektor $\boldsymbol{c}_{(t-1)}$ představující dlouhodobou paměť z předchozího kroku, vektor $\boldsymbol{h}_{(t-1)}$ vyjadřující krátkodobou paměť z předchozího kroku a vstupní vektor $\boldsymbol{x}_{(t)}$. Původní část jednoduché rekurentní neuronové sítě je ve funkci $\boldsymbol{g}_{(t)}$, do které vstoupí vektor $\boldsymbol{h}_{(t-1)}$ a vektor $\boldsymbol{x}_{(t)}$. V konstrukci se dále objevují 3

Značení	Význam
$\boldsymbol{x}_{(t)}$	Vstupní vektor
$\hat{\boldsymbol{y}}_{(t)}$	Výstupní vektor
$\boldsymbol{c}_{(t)}$	Vektor s dlouhodobou pamětí
$\boldsymbol{h}_{(t)}$	Vektor s krátkodobou pamětí
$\boldsymbol{f}_{(t)}$	Ovladač brány zapomnění
$\boldsymbol{g}_{(t)}$	Původní rekurentní část
$\boldsymbol{i}_{(t)}$	Ovladač brány zapamatování
$\boldsymbol{o}_{(t)}$	Ovladač brány vzpomínání

Tabulka 1.5: Značení pro LSTM buňky s časovým krokem t

brány. Brána zapomnění je ovládána funkcí $\boldsymbol{f}_{(t)}$ a určuje, kterou část vektoru dlouhodobé paměti $\boldsymbol{c}_{(t-1)}$ z předchozí kroku je třeba vyřadit. Vyřazené hodnoty jsou nahrazeny v bráně zapamatování ovládané funkcí $i_{(t)}$. Ta vybere, které části $\boldsymbol{g}_{(t)}$ se mají uložit do dlouhodobé paměti $\boldsymbol{c}_{(t)}$. Poslední brána nese název brána vzpomínání, ovládá ji funkce $\boldsymbol{o}_{(t)}$ a jejím cílem je přečíst hodnoty dlouhodobé paměti, které se přidají do výstupů $\hat{\boldsymbol{y}}_{(t)}$ a $\boldsymbol{h}_{(t)}$.

Proces, jakým způsobem přes LSTM buňku proteče datový signál, je zapsán do rovnic

$$\begin{aligned}
\boldsymbol{i}_{(t)} &= \boldsymbol{\sigma}\left(\mathbf{W}_{xi}^{T}\boldsymbol{x}_{(t)} + \mathbf{W}_{hi}^{T}\boldsymbol{h}_{(t-1)}\right),\\
\boldsymbol{f}_{(t)} &= \boldsymbol{\sigma}\left(\mathbf{W}_{xf}^{T}\boldsymbol{x}_{(t)} + \mathbf{W}_{hf}^{T}\boldsymbol{h}_{(t-1)}\right),\\
\boldsymbol{o}_{(t)} &= \boldsymbol{\sigma}\left(\mathbf{W}_{xo}^{T}\boldsymbol{x}_{(t)} + \mathbf{W}_{ho}^{T}\boldsymbol{h}_{(t-1)}\right),\\
\boldsymbol{g}_{(t)} &= \mathbf{tanh}\left(\mathbf{W}_{xg}^{T}\boldsymbol{x}_{(t)} + \mathbf{W}_{hg}^{T}\boldsymbol{h}_{(t-1)}\right),\\
\boldsymbol{c}_{(t)} &= \boldsymbol{f}_{(t)} * \boldsymbol{c}_{(t-1)} + \boldsymbol{i}_{(t)} * \boldsymbol{g}_{(t)},\\
\hat{\boldsymbol{y}}_{(t)} &= \boldsymbol{h}_{(t)} = \boldsymbol{o} * \mathbf{tanh}\left(\boldsymbol{c}_{(t)}\right),
\end{aligned} \tag{1.41}$$

kde indexy matic označují v prvním indexu transformovaný vektor a v druhém funkci, do které cílí. Symbol $*$ značí násobení vektorů po složkách (na obrázku 1.10 značeno $\otimes$).

Kapitola 2

Přeučení

Pod pojmem přeučení je myšleno, že vysoká úspěšnost na trénovacích datech nezaručí i dobrou funkčnost na datech testovacích. Projevuje se například, když je množina trénovacích dat v porovnání se složitostí modelu malá. Model se v trénovacích datech naučí vzory, která jsou typická pouze v těchto datech, tj. naučí se trénovací data v podstatě nazpaměť, a nezná dostatečně obecné vzory pro dobré predikce v datech testovacích. Schopnost modelu poskytnout kvalitní předpovědi pro data, která dosud nespatřil, se nazývá generalizace. Složitost modelu chápeme jako množství parametrů, které obsahuje.

Jedním z praktických způsobu, jak poznat přeučení, je použití validačních dat. V průběhu trénování se na těchto datech po každé epoše vyhodnotí ztrátová funkce a výsledek lze srovnat s její hodnotou pro trénovací data.

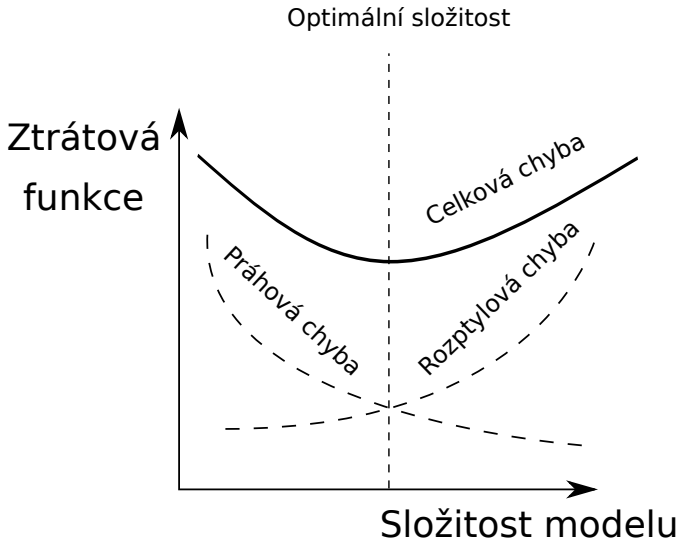
2.1 Způsoby předcházení přeučení

Podle prahově-rozptylového kompromisu (bias-variance trade-off) [1] jsou chyby předpovědi tří druhů:

1. Prahová chyba
    - Způsobena předpoklady modelu, které špatně aproximují daný problém. Například lineární model na polynomiální předpovědi (podučení)
2. Rozptyl
    - Způsoben příliš mnoha parametry modelu (přeučení)
3. Šum
    - Chyba v datech
    - Opravuje se čištěním dat (data cleaning), tj. například odstraněním odlehlých hodnot (outliers) [2].

Cílem je vyhodnotit, která z těchto chyb se v modelu nejvíce projevuje a podle toho model optimalizovat. Znázornění kompromisu nalezneme na obrázku 2.1. Jak je psáno v [1], lze rozdělit střední hodnotu MSE na tři části, které odpovídají výše popsaným druhům chyb. To jednoduše odvodíme z rovnice (2.1)

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 = \frac{1}{n}\sum_{i=1}^{n}\left[g\left(\boldsymbol{x}_i,\ \mathfrak{X}\right) - f\left(\boldsymbol{x}_i\right) - \epsilon_i\right]^2, \tag{2.1}$$



Obrázek 2.1: Prahově-rozptylový kompromis. Velikost chyb podle složitosti modelu (počtu naučených parametrů).

Symbol	Význam
ϵ_i	Šum
$\mathfrak{x}$	Datová sada
g	Funkce znázorňující model
f	Neznámá funkce $y_i = f(x_i) + \epsilon_i$

Tabulka 2.1: Přehled značení v prahově-rozptylovém kompromisu

[1]

$$
\begin{aligned}
E\,(\mathrm{MSE}) &= \frac{1}{n}\sum_{i=1}^{n} E\left[g\left(\boldsymbol{x}_i,\ \mathfrak{X}\right) - f\left(\boldsymbol{x}_i\right) - \epsilon_i\right]^2 \qquad (2.2)\\
&\underset{1}{=} \frac{1}{n}\sum_{i=1}^{n} E\left[f\left(\boldsymbol{x}_i\right) - E\left[g\left(\boldsymbol{x}_i,\ \mathfrak{X}\right)\right] + E\left[g\left(\boldsymbol{x}_i,\ \mathfrak{X}\right)\right] - g\left(\boldsymbol{x}_i,\ \mathfrak{X}\right)\right]^2 + \frac{1}{n}\sum_{i=1}^{n} E\left(\epsilon_i^2\right)\\
&= \frac{1}{n}\sum_{i=1}^{n} E\left(f\left(\boldsymbol{x}_i\right) - E\left[g\left(\boldsymbol{x}_i,\ \mathfrak{X}\right)\right]\right)^2 + \frac{2}{n}\sum_{i=1}^{n}\left(f\left(\boldsymbol{x}_i\right) - E\left[g\left(\boldsymbol{x}_i,\ \mathfrak{X}\right)\right]\right)\cdot \overbrace{\left(E\left[g\left(\boldsymbol{x}_i,\ \mathfrak{X}\right)\right] - E\left[g\left(\boldsymbol{x}_i,\ \mathfrak{X}\right)\right]\right)}^{=0}\\
&+ \frac{1}{n}\sum_{i=1}^{n} E\left(E\left[g\left(\boldsymbol{x}_i,\ \mathfrak{X}\right)\right] - g\left(\boldsymbol{x}_i,\ \mathfrak{X}\right)\right)^2 + \frac{1}{n}\sum_{i=1}^{n} E\left(\epsilon_i\right)^2\\
&= \overbrace{\frac{1}{n}\sum_{i=1}^{n}\left(f\left(\boldsymbol{x}_i\right) - E\left[g\left(\boldsymbol{x}_i,\ \mathfrak{X}\right)\right]\right)^2}^{\text{Prahová chyba}} + \overbrace{\frac{1}{n}\sum_{i=1}^{n} E\left[g\left(\boldsymbol{x}_i,\ \mathfrak{X}\right) - E\left[g\left(\boldsymbol{x}_i,\ \mathfrak{X}\right)\right]\right]^2}^{\text{Rozptyl}} + \overbrace{\frac{1}{n}\sum_{i=1}^{n} E\left(\epsilon_i^2\right)}^{\text{Šum}}.
\end{aligned}
$$

2.1.1 Regularizace[2]

L2 Tikhonovova regularizace přidává do ztrátové funkce sumu kvadrátů vah vynásobenou koeficientem $\lambda > 0$ [1, 2]. Přičítá se pouze při trénování a cílem je držet hodnoty vah co nejnižší. Používání jiné ztrátové funkce během trénování a testování není nic neobvyklého. Například u klasifikátorů je kvůli gradientnímu sestupu vhodné použít diferencovatelné ztrátové funkce, za to u testování se klade důraz na její přesnost. V [1] je tato regularizace popsána rovnicí

$$L = \sum_{\mathbb{X}} (y - \hat{y})^2 + \lambda \sum_{i=1}^{d} w_i^2. \qquad (2.3)$$

Při trénování modelu lineární regrese metodou sestupného gradientu má použití (2.3) za následek aktualizaci vah ve tvaru

$$w_{i+1} = w_i(1 - \alpha\lambda) - \alpha\frac{\partial L}{\partial w_i}, \qquad (2.4)$$

kde α je učící čas. Faktor $(1 - \alpha\lambda)$ drží váhy blízko nule a dá se interpretovat jako faktor zapomínání, který zabraňuje modelu přesně si zapamatovat trénovací data, a tudíž zabraňuje přeučení. Podle [1] lze ukázat, že šum v datech má velmi podobné účinky jako L2 regularizace, a proto se do trénovacích dat někdy přidává uměle.

L1 Dle [1, 2] je snahou L1 (Lassovy) regularizace vyrušit váhy méně důležitých vlastností (feature selector) úpravou ztrátové funkce

$$L = \sum_{\mathfrak{X}} (y - \hat{y})^2 + \lambda \sum_{i=1}^{d} |w_i|. \qquad (2.5)$$

[1] Předpokládá se, že střední hodnota šumu je 0.

[2] Oba dva příklady jsou pro jednoduchost značení reprezentovány na lineární regresi

	Solvery				
Regularizace	**liblinear**	**lbfgs**	**newton-cg**	**sag**	**saga**
Multinomiální + L2 penalizace	×	✓	✓	✓	✓
OVR[3] + L2 penalizace	✓	✓	✓	✓	✓
Multinomiální + L1 penalizace	×	×	×	×	✓
OVR + L1 penalizace	✓	×	×	×	✓
žádná penalizace	×	✓	✓	✓	✓
Chování					
Rychlejší na obrovských datasetech	×	×	×	✓	✓
Robustní na nevyscalených datasetech	✓	✓	✓	×	×

Tabulka 2.2: Volba solveru podle volby penalizace

Váhy v lineární regresi se aktualizují následovně

$$w_{i+1} = w_i - \alpha\lambda \cdot \text{sign}\,(w_i) - \alpha\frac{\partial L}{\partial w_i}. \tag{2.6}$$

V případě implementace logistické regrese se v balíčku `scikit-learn` nachází důležitým hyperparametr C, který určuje sílu regularizace. Volba regularizace, označena hyperparametrem `penalty`, je dále ovlivněna hyperparametrem `solver`, který spouští další možné možné optimalizace gradientního sestupu. Vhodnost jejich použití znázorňuje následující tabulka 2.2 převzata ze `[17]`. V praxi se častěji používá L2 regularizace.

2.1.2 Early stopping

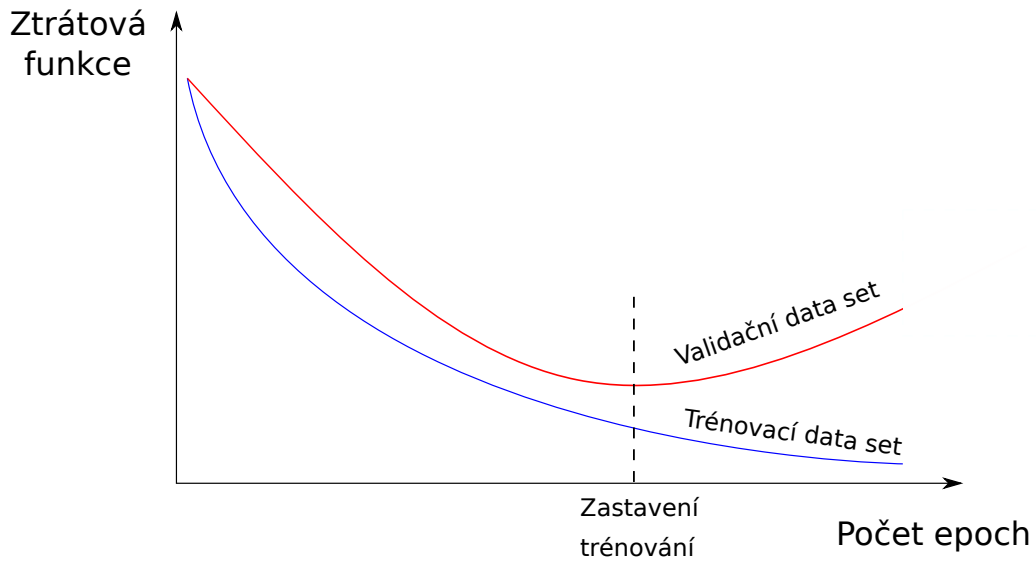
Early stopping funguje na jednoduchém principu, ve kterém se po trénování nastaví váhy na hodnoty, pro něž měla ztrátová funkce dosud nejnižší hodnotu a v následujících epochách nebo dávkách už rostla, anebo oscilovala kolem oné hodnoty. Tento děj je vizualizován na obrázku 2.2. V [2] je doporučeno použití spíše více epoch nebo vrstev a neuronů než méně, protože obsahují dost parametrů pro aproximaci složitějších problémů a early stopping nedopustí přeučení.

2.1.3 Dropout

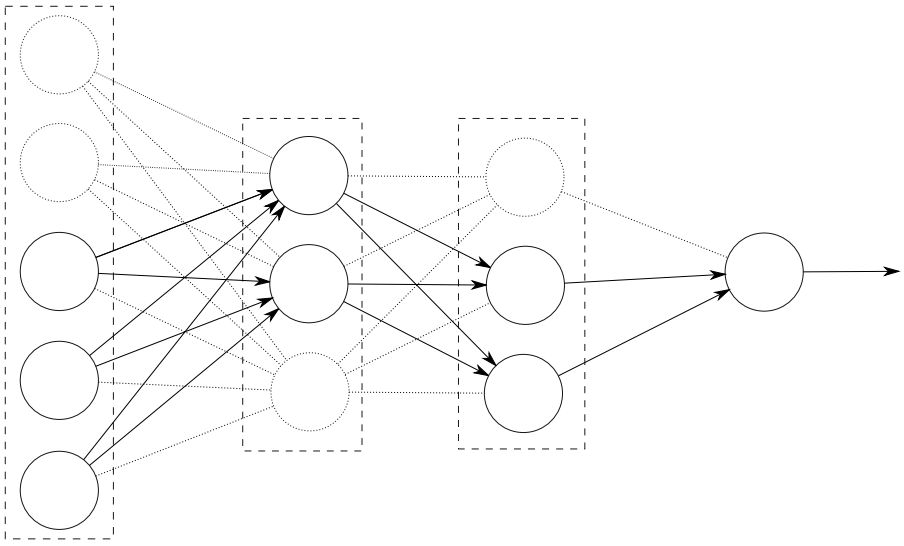
Jak uvádí [1, 2, 4], dropout je technika, ve které se neuron ve vrstvě neuronové sítě neaktivuje při zpracování každého vstupního vektoru s pravděpodobností p. Situaci při určitém vstupním vektoru ilustrujme na obrázku 2.3. K dropoutu dochází pouze při trénování. Hyperparametr p se nazývá dropout rate. Pro rekurentní sítě se doporučuje ve škále $20\% - 30\%$, pro konvoluční v hodnotách $40\% - 50\%$. Nazvěme neuronovou síť různé konstalace neaktivovaných neuronů pomocnou (thinned) neuronovou sítí, potom se celkově při trénování neuronové sítě s n neurony objeví 2^n různých pomocných neuronových sítí. Konečný model můžeme chápat jako jakési zprůměrování těchto sítí, které se docílí přenásobením každé váhy právě číslem p. V plně propojené neuronové síti k dropoutu s pravděpodobností p dojde v l-té vrstvě tak, že se výstup aktivační funkce $\boldsymbol{h_l}$ vynásobí po složkách s vektorovou náhodnou veličinou $\boldsymbol{r_l} \sim Be(p)$ (2.7) a až poté se pronásobí váhami. Experimentálně bylo na modelech [4] zjištěno, že dropout zvyšuje přesnost modelu až o 2 procentní body.

$$\tilde{\boldsymbol{h}}_l = \boldsymbol{r_l} * \boldsymbol{h_l} \tag{2.7}$$

[3]OVR - One vs. Rest způsob multitřídové klasifikace, kde pro každou je právě jeden binárních klasifikátorů, který posuzuje, zda do této třídy prvek patří či nikoli.



Obrázek 2.2: Early stopping. Modrá křivka zobrazuje vývoj ztrátové funkce na trénovacím data setu a červená na validačním data setu. Je patrné, že k zastavení učení dojde při epoše, ve které dosáhne ztrátová funkce na validačním data setu minima.



Obrázek 2.3: Dropout
Neurony a váhy, které se při určitém vstupním vektoru z důvodu dropoutu neuplatní, jsou znázorněny přerušovanou čarou.

Biologickou motivaci najdeme v teorie evoluční sexuální reprodukce. Každý z rodičů poskytne potomkovi náhodných 50% genů a tím se snižuje míra koadaptace, geny nemůžou být tolik závislé na ostatních genech, což zvyšuje odolnost genů vůči poruchám. Stejně tak neuronové síti se po použití dropoutu sníží závislost jednotlivých neuronů a síť se stává komplexnější. [4]

Kapitola 3

Praktická část

3.1 Použitý software a datové sady

Jak již bylo na začátku kapitoly 1 zmíněno, jedním z nejpopulárnějších programovacích jazyků pro strojové učení je Python, byl použit i pro implementaci algoritmů v této práci. Jedná se o skriptovací jazyk, který je velmi univerzální, jednoduchý pro uživatele a má bohaté knihovny s dobře popsanými dokumentacemi. V implementaci modelů je použito několik jeho balíčků:

- Tensorflow a Keras

  Tensorflow je široká knihovna (low-level API) pro používání neuronových sítí. Keras je jeho součástí a představuje high-level API pro neuronové sítě. Více kapitola 1.

- Scikit-learn

  Knihovna obsahující velkou část algoritmů strojového učení. Dále obsahuje i nástroje pro práci s daty.

- NumPy

  NumPy je základní balíček pro numerické výpočty. Nachází se v něm celá řada funkcí pro řešení matematických problémů (aritmetika polí, lineární algebra), a proto je využíván v mnoha dalších balíčcích. Základním datovým typem balíčku je numpy pole nazývané ndarray, které má oproti klasickým pythonovským datovým typům několik rozdílných vlastností (typovost, vektorové operace implementované v C++).

- Pandas

  Pandas je balíček pro práci s daty a je využíván pro jejich zpracování a analýzu v mnoha oblastech. Jeho součástí jsou dva hlavní datové typy `Series` (1-rozměrné) a `DataFrame` (2-rozměrné). `DataFrame` je tabulková datová struktura. V jejich sloupcích se mohou objevovat různé datové typy a řádky jsou značeny indexy.

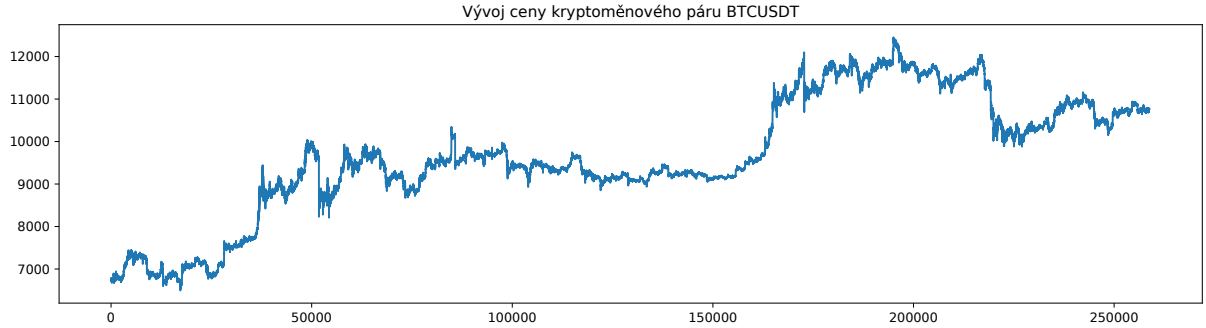
- Matplotlib

  Tento balíček je určen pro vykreslování grafů a další vizualizace.

Balíčky byly nainstalovány ve virtuální prostředí, které umožňuje, aby jednotlivé balíčky byly aktivní jen uvnitř tohoto prostředí a ne v celém počítači. Virtuální prostředí bylo vytvořeno v distribuci Linuxu Fedora 28. Přechod do virtuálního prostředí se provádí v bashi, a poté je možno v tomto prostředí začít pracovat. K samotné instalaci balíčků byl použit správce instalací balíčků Pythonu pip. Postup příkazů,

Výpis kódu 3.1 Aktivování virtuálního prostředí

```
$ python3.6 venv název_prostředí
$ source název_prostředí/bin/activate
$ pip install název_balíčku
```



Obrázek 3.1: Vývoj ceny kryptoměnového páru BTCUSDT

kterými se vytvoří virtuální prostředí je na výpisu kódu 3.1.

Práce byla vypracovávána ve dvou vývojových prostředích Jupyter Notebook a Pycharm. Jupyter Notebook funguje v prohlížeči a jeho dokument se nazývá notebook. Notebook je členěn do buněk, které umožňují živou interakci s kódem, tj. vizualizace dat v průběhu programu, zkoušení programu řádek po řádku apod. Naopak Pycharm je klasické vývojové prostředí od společnosti JetBrains s mnoha pokročilými funkcemi (například debugger a unit testing).

Jako datová sada jsou použita data z kryptoměnové burzy Binance. Server Binance obsahuje širokou podporu pro obchodování s kryptoměnami a vzdělání v tomto oboru. Data jsou ve tvaru vývoje cen kryptoměnových párů v časovém rozestupu jedné minuty. Jedná se tedy o časovou řadu, pro kterou by mělo být výhodné použít rekurentní neuronovou síť, konkrétně LSTM (viz podsekce (1.6.7.1)). V této práci je objektem zkoumání kryptoměnový pár BTCUSDT, jehož vývoj je vizualizován na obrázku 3.1 a obsahuje 259026 vzorků

3.2 Klasifikační model

Cílem práce byl pokus o napodobení technické analýzy, která zkoumá budoucí vývoj ceny analyzováním současných cenových grafů. V klasifikační úloze model předpovídá, zda cena kryptoměnového páru za volený časový úsek klesne, vzroste, nebo se příliš nezmění. Volený časový úsek byla volena jedna hodina. Pro klesnutí je pozorovaná hodnota určena 0, pro vzrůst 1 a 2 bylo pro menší změny definované jako

$$y = 2 \Leftrightarrow \left(\mid 1 - \frac{\text{Cena v budoucnu}}{\text{Aktuální cena}} \mid < 0.002 \right). \tag{3.1}$$

Tento poměr byl volen tak, aby případný zisk pokryl poplatky. Predikce neuronové sítě by tedy mohly napovědět, kdy nakoupit kryptoměnu za dolary, a kdy ji poté prodat. Práce s neuronovou sítí se skládá ze tří kroků:

- Předzpracování dat
- Implementace modelu

	BTCUSDT
0	6733, 45
1	6714, 33
2	6719, 57
3	6722, 48
4	6721, 58
5	6718, 07
6	6717, 55
7	6720, 77
8	6723, 79
9	6731, 64

Tabulka 3.1: Prvních 10 řádků datové sady

- Trénování modelu, ladění hyperparametrů a interpretace výsledků

3.2.1 Předzpracování dat

Původní načtená syrová data jsou ve tvaru v tabulce 3.1 a nejdříve je nutné je předzpracovat. Pod pojmem předzpracování dat je skryto několik kroků, které se provedou před vstupem dat do neuronové sítě. Předzpracování dat ve tvaru časové řady se skládá z:

- Tvorba vlastností vstupního vektoru (feature engineering)
- Vytvoření pozorovaných hodnot y
- Rozdělení dat na testovací a trénovací
- Normalizace a standardizace
- Vybalancování a zamíchání data setu
- Vektorizace

V programu je využito objektového programování a všechny tyto kroky vykonávají tři třídy.

3.2.1.1 Třída `Bollinger`

První nese název `Bollinger` a jejím úkolem je tvorba vlastností, které budou součástí vstupního vektoru. Název je převzat ze stejnojmenného indikátoru používaného v technické analýze [20]. Celkově přidá do vstupního vektoru tři další vlastnosti:

- Klouzavé průměry MA

  Jedná se o průměry za posledních n po sobě jdoucích časových kroků

$$\mathrm{MA}_k = \frac{\sum_{i=k-n+1}^{k} t_i}{n} \tag{3.2}$$

- Součet klouzavého průměru MA a součinu klouzavé směrodatné odchylky σ s kladným reálným koeficientem r

$$\mathrm{MA}_+ = \mathrm{MA} + r \cdot \sigma \tag{3.3}$$

	BTCUSDT	Moving_average	Moving_std_up	Moving_std_down
119	6756, 69	6750, 953167	6787, 466623	6714, 439711
120	6758, 87	6751, 165000	6787, 563632	6714, 766368
121	6752, 83	6751, 485833	6787, 247979	6715, 723688
122	6742, 19	6751, 674333	6786, 993634	6716, 355032
123	6743, 31	6751, 847917	6786, 791215	6716, 904618
124	6745, 00	6752, 039750	6786, 571990	6717, 507510
125	6743, 95	6752, 255417	6786, 251002	6178, 259832
126	6746, 22	6752, 494333	6785, 904028	6719, 084638
127	6740, 67	6752, 660167	6785, 629345	6719, 690988
128	6732, 04	6752, 728917	6785, 489006	6719, 968827

Tabulka 3.2: Vytvoření dalších vlastností vstupního vektoru. Moving_average je klouzavý průměr s periodou 120. Moving_std_up je součet Moving average a dvojnásobku směrodatné odchylky a Moving_std_down je jejich rozdíl. Vizualizováno je prvních 14 řádků. První index je 119 kvůli výpočtu klouzavého průměru dle (3.2).

- Rozdíl klouzavého průměru a součinu klouzavé směrodatné odchylky σ s kladným reálným koeficientem r

$$\mathrm{MA}_{-} = \mathrm{MA} - r \cdot \sigma \tag{3.4}$$

Atributy třídy `Bollinger` jsou:

- Data

  Jedná se o `dataframe`, jehož řádky symbolizují vstupní vektory a sloupce jednotlivé vlastnosti. Inicializuje se s jedním sloupcem obsahujícím cenu kryptoměnového páru a funkcí `create_moving` se vytvoří výše zmíněné vlastnosti.

- Délka periody klouzavého průměru

  Určuje velikost n v rovnici (3.2), tj. počet průměrovaných prvků v jedné periodě.

- Koeficient r, kterým se násobí směrodatná odchylka.

Hodnoty byly zvoleny $n = 120$ a $r = 2$ podle inspirace v [21]. Po vytvoření těchto dalších vlastností se `dataframe` rozroste o 3 další sloupce a její vzhled zachycuje tabulka 3.2.

Kód se nachází ve výpisu kódu 3.2.

3.2.1.2 Třída `Preprocess` a `Classification`

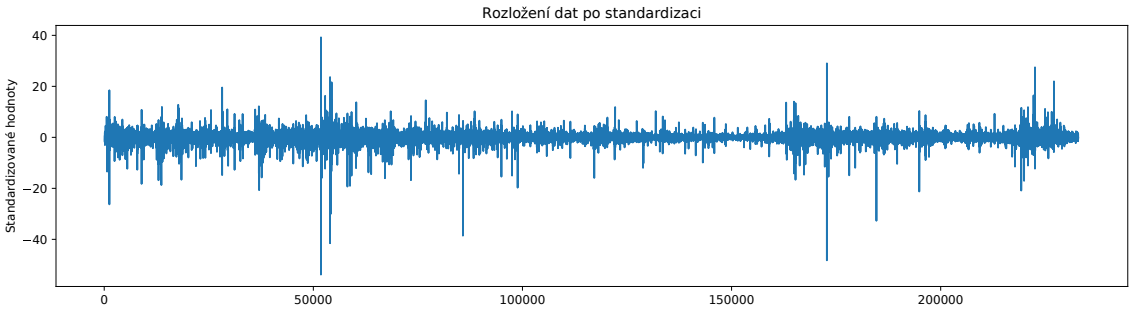
Třída `Preprocess` zajišťuje normalizaci dat a rozdělení na data testovací a data trénovací. Normalizace dat je nutná pro rychlejší konvergenci algoritmu. Probíhá ve dvou krocích. Data jsou nahrazena procentuálními změnami oproti předchozí hodnotě a poté škálovaná pomocí funkce `scale` z balíčku `sklearn` třídy `preprocessing`, který data škáluje dle vzorce

$$\hat{x} = \frac{x - \mu}{\sigma}, \tag{3.5}$$

tj. vystředuje rovnici (3.5) podle průměru μ a podělí směrodatnou odchylkou σ. Vystředování podle průměru se nazývá standardizace dat. Distribuce hodnot po standardizaci je vizualizována na obrázku 3.2.

Výpis kódu 3.2 Třída `Bollinger`

```
class Bollinger:
    def __init__(self, data, period=28800, r=2, market = 'BTCUSDT'):
        self.data = data
        self.period = period
        self.r = 2
        self.market=market
    def create_moving(self):
        Data = self.data
        Period = self.period
        r = self.r
        Data['Moving_average'] = Data.iloc[:,0].rolling(window=Period).mean()
        self.data['Moving_average'] = Data['Moving_average']
        Data['std'] = Data.iloc[:, 0].rolling(window=Period).std()
        self.data['Moving_std_up'] = Data['Moving_average'] + r * Data['std']
        self.data['Moving_std_down'] = Data['Moving_average'] - r * Data['std']
        self.data = self.data.drop('std', 1)
        self.data.dropna(inplace=True)
```



Obrázek 3.2: Vizualizace dat po standardizaci

Výpis kódu 3.3 Třída `Preprocess`

```
class Preprocess:
    def __init__(self, data, scaler=scale, seq_length=60, fut_period=3,
                 market='BTCUSDT', train=True):
        self.data = data
        self.scaler = scaler
        self.seq_length = seq_length
        self.fut_period = fut_period
        self.market = market
        self.train_data = 0
        self.test_data = 0
        self.train = train
    def split(self, test_split=0.3):
        numbers = sorted(self.data.index.values)
        pct70_100 = sorted(self.data.index.values)[-int(test_split * len(numbers))]
        self.test_data = self.data[(self.data.index >= pct70_100)]
        self.train_data = self.data[self.data.index < pct70_100]
    def normalize(self, Data):
        for column in Data.columns:
            if column != "labels":
                    Data[column] = Data[column].pct_change()   # vezmu procentuální změnu
                    Data[column] = scale(Data[column].values)  # vyškálování
        Data.dropna(inplace=True)                              # odhození prázdných řádků
        return Data
```

	BTCUSDT	Moving_average	Moving_std_up	Moving_std_down	labels
119	6756, 69	6750, 953167	6787, 466623	6714, 439711	0
120	6758, 87	6751, 165000	6787, 563632	6714, 766368	0
121	6752, 83	6751, 485833	6787, 247979	6715, 723688	0
122	6742, 19	6751, 674333	6786, 993634	6716, 355032	0
123	6743, 31	6751, 847917	6786, 791215	6716, 904618	0
124	6745, 00	6752, 039750	6786, 571990	6717, 507510	0
125	6743, 95	6752, 255417	6786, 251002	6178, 259832	0
126	6746, 22	6752, 494333	6785, 904028	6719, 084638	0
127	6740, 67	6752, 660167	6785, 629345	6719, 690988	0
128	6732, 04	6752, 728917	6785, 489006	6719, 968827	0

Tabulka 3.3: Vytvoření pozorovaných hodnot

Funkce `split` rozděluje data na testovací a trénovací v základním nastavení v poměru 9:1. Data je nutno před normalizací nejdříve rozdělit, protože v rovnici (3.5) by v průměru a směrodatné odchylce byly zahrnuty informace zároveň z trénovacích a testovacích dat. Kód třídy je uveden ve výpisu kódu

Třetí třída `Classification` je potomkem třídy `Preprocess` a obsahuje funkce, které upraví data set ke klasifikační úloze. To znamená, že vytvoří pozorované hodnoty y k příslušným vstupním vektorům. Výsledný `dataframe` je v tabulce 3.3. Dále jsou v této funkci data vektorizována.

Vektorizací je myšleno vytvoření sekvencí délky zvolené v atributech třídy `Classification` a převedení do `numpy` pole o rozměrech (počet sekvencí)×(délka sekvence)×(počet vlastností). Délku sekvence je možno zvolit. Aby nedošlo k přeučení, je možnost data „zamíchat“ a vybalancovat počet vstupních vektorů příslušných k jednotlivým třídám. Míchají se a balancují se pouze trénovací data, protože testovací data již nemají vliv na hodnoty vah. Vybalancování dat probíhá tak, že určím počet prvků nejméně obsáhlé třídy a této hodnotě vyrovnám počty prvků v ostatních třídách. Potom každá třída obsahuje přesně $1/3$ trénovacích dat. Kód třídy je uveden ve výpisu kódu 3.4.

Výpis kódu 3.4 Třída `Classification`

```
class Classification(Preprocess):
        def __init__(self, data, scaler=scale, seq_length=60, fut_period=3, market='BTCUSDT'):
            super().__init__(data, scaler, seq_length, fut_period, market)
        def create_labels(self):
            def classify(present, future):
                if float(future) / float(present) > 1.002:
                    return 1
                elif float(future) / float(present) < 0.998:
                    return 0
                else:
                    return 2
            self.data['future'] = self.data[f'{self.market}'].shift(-self.fut_period)
            self.data.dropna(inplace=True)
            self.data['labels'] = list(map(classify,
                                           self.data[f'{self.market}'], self.data['future']))
            self.data = self.data.drop("future", 1)
        def vectorized(self, Data, ballance=True):
            seq_data = []
            prev_prices = deque(maxlen=self.seq_length)

            for i in Data.values:
                prev_prices.append(
                    [n for n in i[:-1]])
                if len(prev_prices) == self.seq_length:
                    seq_data.append(
                        [np.array(prev_prices), i[-1]])
            if ballance == True:
                down = []
                up = []
                draw = []

                for seq, label in seq_data:
                    if label == 0:
                        down.append([seq, label])
                    elif label == 1:
                        up.append([seq, label])
                    elif label == 2:
                        draw.append([seq, label])
                minimum = min(len(sells), len(buys), len(draw))
                up = buys[:minimum]
                down = sells[:minimum]
                draw = draw[:minimum]

                seq_data = up + down + draw
                random.shuffle(seq_data)
            X = []
            y = []
            for seq, label in seq_data:
                X.append(seq)
                y.append(label)
            return np.array(X), np.array(y)
```

Parametr	Hodnota
Počet neuronů	Voleno z trénování
Sekvenční výstup	`True` (poslední LSTM vrstva `False`)
Stateful	`False`
Tvar vstupních dat	(délka sekvence)×(počet vlastností) pouze u první LSTM vrstvy

Tabulka 3.4: Hodnoty parametrů LSTM vrstvy

Celý proces předzpracování dat je shrnut ve vývojovém diagramu na obrázku 3.3.

3.2.2 Tvorba modelu

Po předzpracování dat následuje tvorba modelu. Model neuronové sítě byl implementován s balíčky Tensorflow a Keras. V balíčku Keras existuje několik rozhraní pro tvorbu modelu. V práci je zvoleno Sequential API, kde se model implementuje jako zásobník, jehož prvky tvoří vrstvy neuronové sítě. Byly použity 4 různé typy vrstev:

- LSTM

  Detailně teoreticky rozebraná v podsekci 1.6.7.1. Používané parametry jsou počty neuronů, `return_sequences` (dále sekvenční výstup), `stateful` a `input_shape` (dále tvar vstupních dat). Tvar vstupních dat se volí pouze v případě první LSTM vrstvy. `Stateful` znamená, zda je mezi následujícími sekvencemi návaznost, tj. po posledních prvku sekvence následuje první prvek následující sekvence. V případě zamíchání dat nemá jeho použití smysl. Sekvenční výstup zadává vrstvě, jestli má být výstup opět sekvence. Jeho hodnota musí být `True` pro všechny LSTM vrstvy, na které navazuje další LSTM vrstva, a pro poslední LSTM vrstvu `False`. Počet neuronů a počet vrstev nelze předem odhadnout a je nutné o nich rozhodnout z výsledků trénování podle úspěšností na validačních datech. Nastavení parametrů LSTM vrstev je shrnuto v tabulce 3.4.

- BatchNormalization (Dále dávková normalizace)

  Vstupní vektor po průchodu skrytou vrstvou přestává být standardizovaný. Úkolem svazové normalizace je výstup skrytých vrstev opět standardizovat vzhledem k současné dávce. Dále tato vrstva standardizovaná data znovu vyškáluje a posune podle parametrů, které se naučí z algoritmů zpětného šíření. Svazovou normalizaci lze použít jako první a tím nahradit potřebu data v předzpracování standardizovat [2]. Celý proces svazové normalizace se skládá z rovnic:

  $$\hat{\boldsymbol{x}} = \frac{\boldsymbol{x} - \boldsymbol{\mu}}{\sqrt{\sigma^2 + \epsilon}}, \tag{3.6}$$

  $$\boldsymbol{z} = \boldsymbol{\gamma} * \boldsymbol{x} + \boldsymbol{\beta}, \tag{3.7}$$

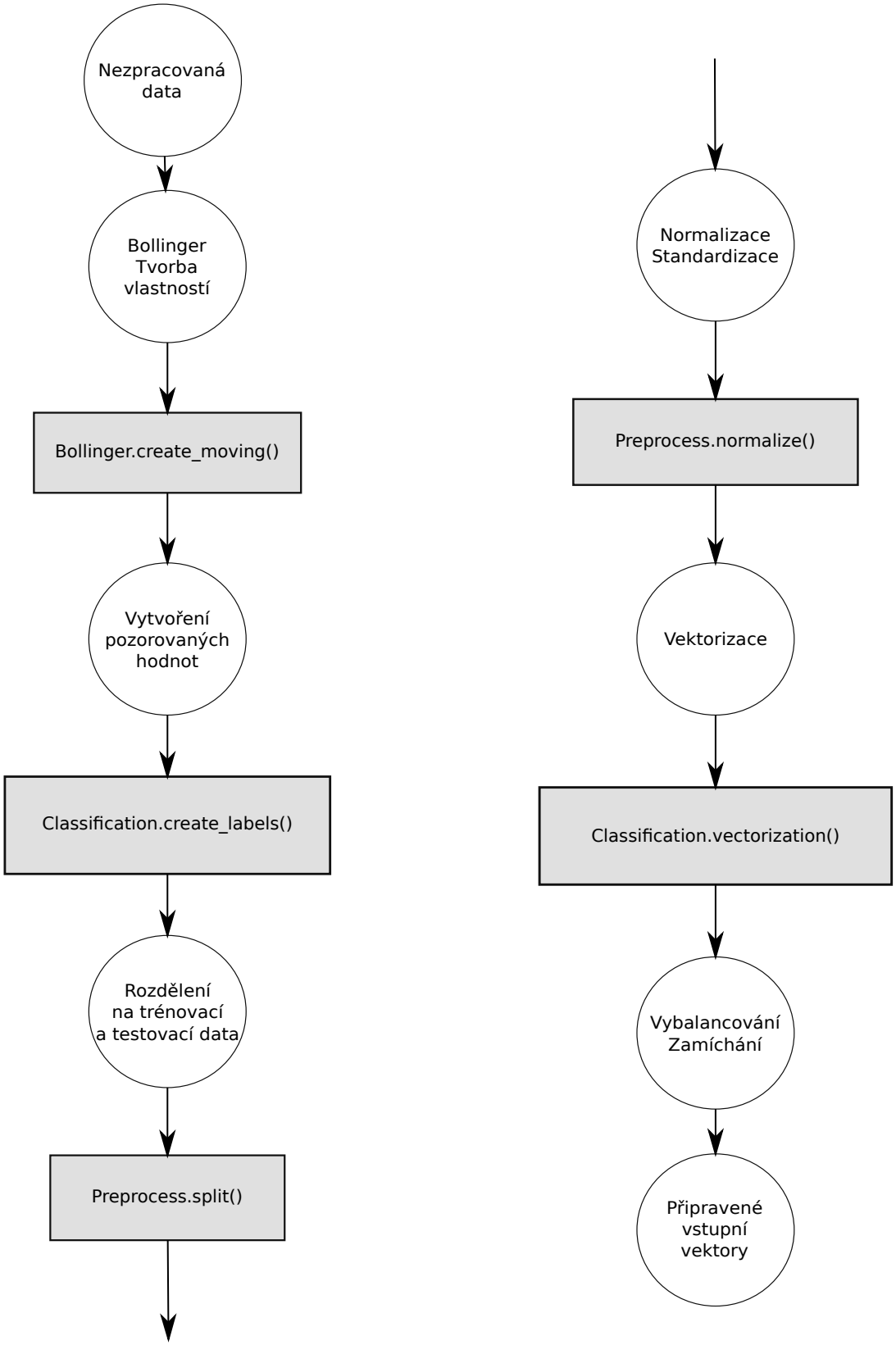
  jejichž symboly jsou popsány v tabulce 3.5.

- Dropout

  Detailně teoreticky rozebrán v podsekci 2.1.3. Jediným používaným parametrem je `dropout_rate`. Vliv Dropoutu na úspěšnost předpovědí je rozebrán v podsekci 2.1.3.

- Dense

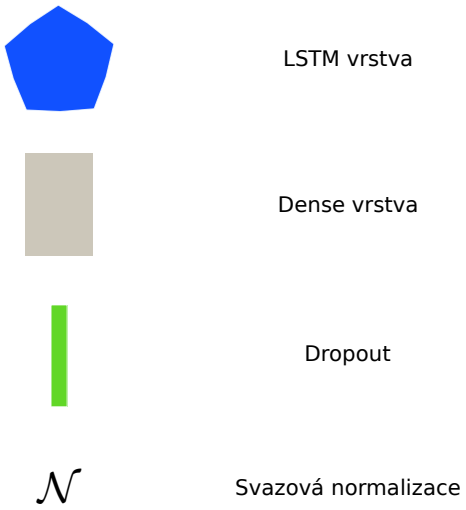
  Jedná se o standardní plně propojenou dopřednou vrstvu, která je použita jako poslední pro tvorbu klasifikačních výstupů. Jako aktivační funkce je proto volena softmax (1.11) a obsahuje pouze tři neurony.



Obrázek 3.3: Vývojový diagram předzpracování dat. V kruzích jsou sepsány jednotlivé kroky a v šedých obdélnících metody, které tyto kroky zrealizují.

Symboly	Význam
μ	Průměr vstupů vzhledem k dávce
σ	Směrodatná odchylka vzhledem k dávce
$\epsilon > 0$	Regularizační člen zabraňující dělení 0
β	Výstupní parametr posunu (hodnota určena v rámci algoritmu zpětného šíření)
γ	Výstupní škálovací parametr (hodnota určena v rámci algoritmu zpětného šíření)
z	Výstup ze svazové normalizace

Tabulka 3.5: Významy symbolů ve svazové normalizaci



Obrázek 3.4: Vizualizace jednotlivých vrstev. Počet neuronů ve vrstvě bude značen číslem uvnitř objektu představujícího vrstvu.

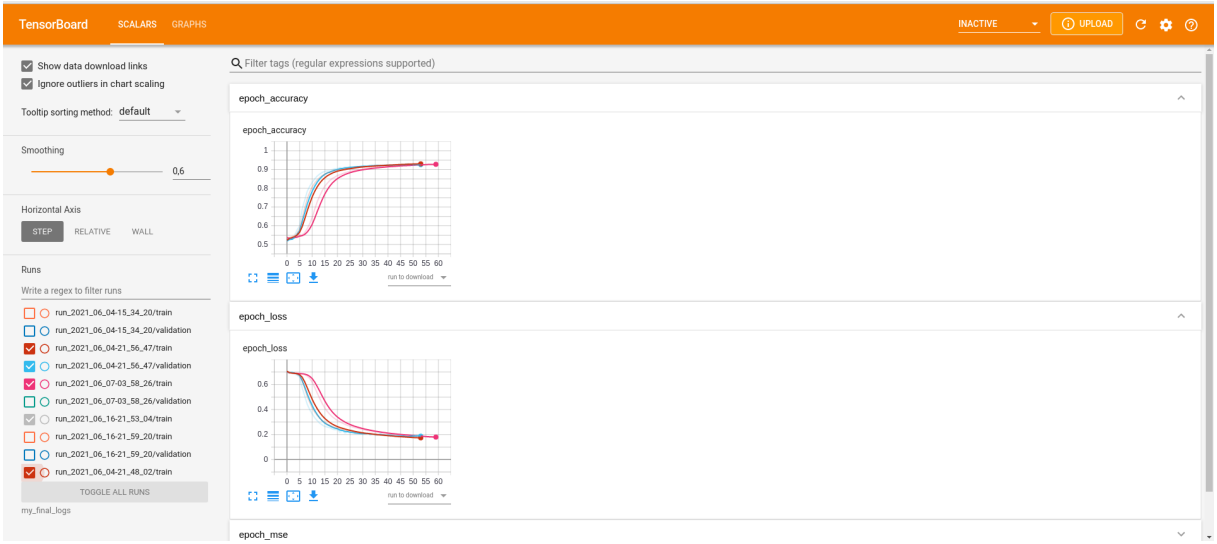
Vizualizace jednotlivých vrstev je na obrázku 3.4. K optimalizaci gradientního sestupu byl použit algoritmus Adam (podsekce 1.6.6.2).

Protože se jedná o klasifikační úlohu, je jako ztrátová funkce volena jedna z variant křížové entropie implementované v balíčku Keras. Pozorované hodnoty jsou kategorické proměnné a pro vstupní vektor je pozorovaná hodnota reprezentována přirozeným číslem příslušné kategorie. Z toho důvodu je variantou křížové entropie volena `sparse_categorical_crossentropy`. Dalším způsobem reprezentace pozorovaných hodnot je tzv. one hot encoding, kdy je pozorovaná hodnota vektorem o velikosti rovné počtu kategorií. Kategorie jsou seřazeny jako prvky tohoto vektoru. Složka vektoru reprezentující správnou kategorii ke vstupnímu vektoru má hodnotu 1 a ostatní složky mají hodnotu 0. Pro tento případ by byla jako křížová entropie volena `Categorical_crossentropy`. V modelu je vhodné používat metriky, které již při jeho trénování zobrazí, jak úspěšně model předpovídá na trénovacích a validačních datech. V případě klasifikačních úloh se volí skupina metrik `accuracy`, vyjadřující procentuální úspěšnost správných předpovědí. Knihovna Keras si již podle pozorovaných hodnot a volby ztrátové funkce sám zvolí, kterou přesně použije. V našem případě se jedná o `sparse_categorical_accuracy`.

Učení modelu se inicializuje funkcí fit, ve které je potřeba zvolit počet epoch, velikost dávek, validační rozdělení a zpětné vazby (callbacks). Počet epoch je volen vyšší, protože při použití early stopping se model nepřeučí. Validační rozdělení určí procento trénovacích dat, jež se použijí jako validační data. V programu je tato hodnota volena jako $0,1$. Velkou výhodou balíčku Tensorflow je možnost použití zpětných vazeb. V modelu jsou použity následující zpětné vazby:

Výpis kódu 3.5 Tvorba log adresáře. Pro odlišení různých záznamů je název volen s časem spuštění.

```
root_logdir = os.path.join(os.curdir, filepath)
def get_run_logdir():
    run_id = time.strftime("run_%Y_%m_%d-%H_%M_%S")
    return os.path.join(root_logdir, run_id)
```



Obrázek 3.5: Snímek obrazovky s otevřeným Tensorboardem

- Early Stopping

  Parametr `patience` je volen 20.

- Tensorboard

  Je to nástroj, který slouží k zaznamenávání činnosti neuronové sítě při učení a následné vizualizace. Při jeho použití je nutné nejdříve vytvořit log adresář (výpis 3.5), do kterého se uloží záznamy o učení neuronové sítě. Tensorboard se spouští v terminálu, který spustí http server na lokálním stroji. Poté je možné využít libovolný webový prohlížeč k přístupu k široké škále vizualizací jako například:

  – Grafy metrik a ztrátových funkcí v průběhu epoch,
  – Grafická vizualizace architektury neuronové sítě,
  – Distribuce a histogramy vah v průběhu učení.

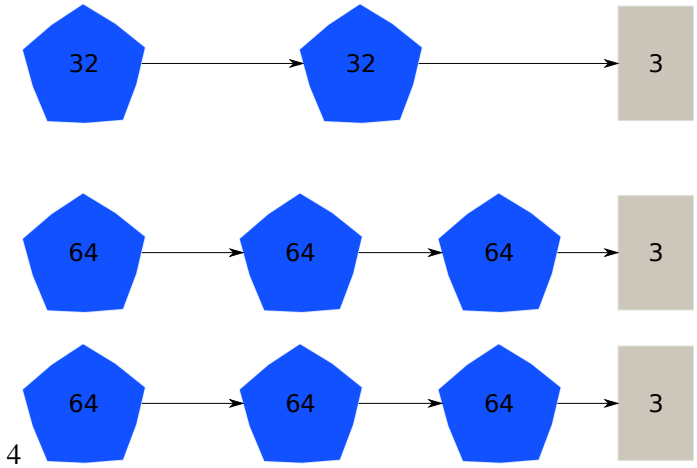
  Použití Tensorboardu je vhodné pro porovnávání výsledků různých architektur neuronových sítí, monitorování gradientních problémů nebo pro ověření, zda jsou vrstvy správně postaveny. Snímek obrazovky s otevřeným Tensorboardem je na obrázku 3.5.

- Checkpoint

  V průběhu učení se může stát, že program spadne, anebo dojde k jinému problému, po kterém bude nutné učení neuronové sítě opakovat. To obvykle zabere mnoho času, a proto je použita zpětná vazba Checkpoint, která po každé epoše uloží stav neuronové sítě. Při technickém problému je možné neuronovou síť opět načíst, pokračovat v učení a ušetřit čas.

Data a hyperparametry	Nastavení
Počet vzorků v trénovacích datech	127953
Počet vzorků v testovacích dat	77553
Procento použití testovacích dat na validační	50%
Délka sekvence	120
Vybalancování trénovacích dat	ANO
Vybalancování validačních dat	NE
Velikost dávek (`batch_size`)	64
Počet epoch (`epochs`)	60
`patience`	20

Tabulka 3.6: Použité nastavení dat a hyperparametrů stejné pro všechny neuronové architektury



Obrázek 3.6: Vizualizace základních architektur použitých neuronových sítí v podsekci 3.2.4

3.2.3 Interpretace výsledků

Pro vyhodnocení výsledků je použita matice záměn (confusion matrix). Matice záměn je čtvercová matice, jejíž sloupce odpovídají třídám vstupních vektorů předpovídané modelem a řádky skutečné příslušnosti vstupních vektorů ke třídám (tedy shody jsou na diagonále). Její prvky jsou pak relativní četnosti předpovědí a skutečných příslušností. Matice dokáže vizualizovat, které třídy dokázala neuronová síť předpovídat lépe než ostatní, nebo naopak které dokázala odhalit jen stěží. Součet na diagonále udává úspěšnost předpovědí.

3.2.4 Základní architektura

Nejdříve bylo voleno nastavení pouze s LSTM vrstvami a bylo zkoumáno, jaký vliv budou mít na výsledky tři různé volby počtu vrstev a počtu neuronů. Vizualizace architektury je na obrázku 3.6. Nastavení, které je stejné pro všechny použité neuronové architektury, je v tabulce 3.6.

Počet epoch byl volen 60. Z obrázku 3.7 lze pozorovat zastavení trénování po jiném počtu epoch. To je způsobeno použitím early stopping, který trénování sítě zastavil ve chvíli, kdy se začala přeučovat. Parametr `restore_best_weights` je `False`, tudíž cílem early stopping bylo pouze zastavit trénování sítě v případě, že se začne přeučovat dříve, než dosáhne 60 epoch. `Patience` byl volen 20, takže proběhne dost epoch na to, aby šlo z grafů vizualizující trénování zpozorovat, zda validační ztrátová funkce

přestává klesat trendově, anebo se jedná pouze o výjimku (nastavení early stopping je ve všech modelech stejné). Největšího počtu epoch dosáhla síť se 3 LSTM vrstvami o 64 neuronech, jejichž průběh učení je na obrázku 3.7c. Validační ztrátová funkce dosáhla minima po 3. epoše, a poté s různými oscilacemi rostla. Podobné chování je vidět i u dalších architektur na obrázcích 3.7a a 3.7b. Vývoj `accuracy` a ztrátové funkce na trénovacích datech je pro všechny tři architektury stejný. `Accuracy` mírně stoupá a a ztrátová funkce mírně klesá. Mezi `accuracy` na validačních datech a trénovacích datech lze pozorovat velký posun.

Vizualizace výsledků je na obrázku 3.8. Všechny sítě dokázaly nejlépe rozeznat kategorii 2, tj., že cena se příliš nezměnila. Úspěšnosti na validačních datech dosahují u všech architektur více než 33%, tudíž síť funguje lépe než náhodný výběr. Nicméně to stále není příliš a z obrázku 3.7 nelze pozorovat, že by validační ztrátové funkce měly stabilní vývoj.

3.2.5 Vliv Dropoutu a dávkové normalizace

V další architektuře se nejdříve zkoumá vliv svazové normalizace, která se nachází po každé vrstvě, jak je ukázáno na obrázku 3.9. Vývoj trénování vizualizuje obrázek 3.10. Vývoje accuracy a ztrátové funkce na trénovacích datech vykazují rychlejší změnu než v předchozím případě. To můžeme přiznat právě vlivu svazové normalizace. Vývoj validačních křivek je podobný jako v předchozím případě mimo poslední architekturu na obrázku 3.10c, která dosáhla nejvyššího počtu epoch 35. Matice záměn jsou vizualizovány na obrázku 3.11 nejlépe dokáží architektury neuronových sítí poznat kategorii 2. Stejně jako v minulém případě jsou úspěšnosti vyšší než 33%, ale nepřesahují příliš tuto hranici a validační ztrátová funkce velmi osciluje kolem určité hodnoty, anebo od začátku roste.

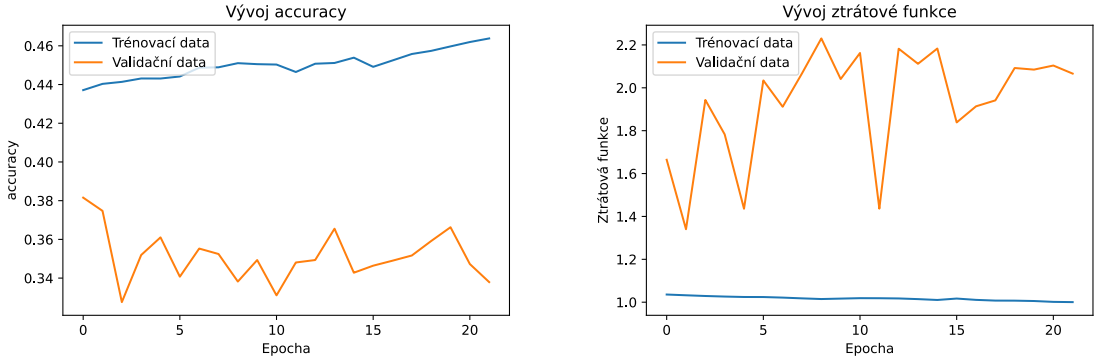
Další zkoumanou architekturou je přidání vlivu dropoutu. Tato vrstva (drop rate volen jako 0.2) se bude nacházet vždy mezi LSTM vrstvou a svazovou normalizaci. Vizualizace se nachází na obrázku 3.12. Na obrázku 3.13 vykazují křivky podobné chování jako v předchozích případech, až na validační ztrátovou funkci, která osciluje mezi nižšími hodnotami. Síť se 3 LSTM vrstvami dosáhla nejlepších výsledků v matici záměn na obrázku 3.14b, avšak vývoj ztrátové funkce, která roste, nenasvědčuje tomu, že by tento výsledek byl stabilní.

3.2.6 Vliv přidání Dense jako předposlední vrstvy

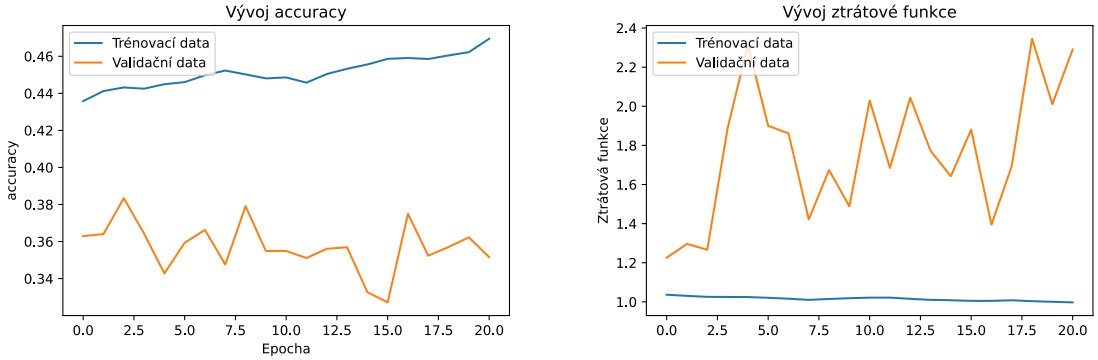
V různých LSTM architekturách na stránkách Kaggle se používá jako poslední skrytá vrstva Dense, proto byl zkoumán její vliv. Umístění v architekturách je na obrázku 3.15. Na obrázku 3.16 je vizualizováno trénování tří architektur, nicméně validační ztrátová funkce osciluje ze všech architektur mezi nejvyššími hodnotami.

3.2.7 Výsledky

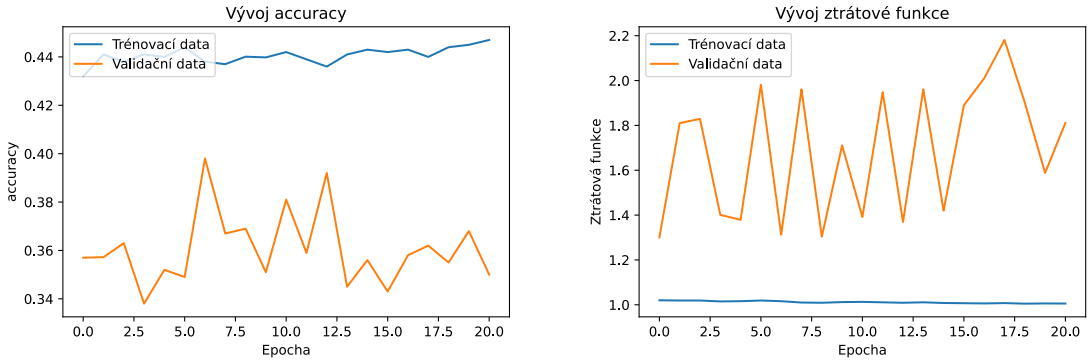
Přestože validační ztrátové funkce nevykazovaly stabilní chování a obvykle velmi brzy začaly oscilovat anebo růst, z výsledků lze pozorovat, že modely fungují lépe než náhodný výběr a jejich úspěšnosti přesahují 33% na validačních datech. Nicméně nepřesahují tuto hodnotu příliš a modely dokáží nejlépe rozeznat, že se cena příliš nezmění, což není pro obchodování výdělečné. Jelikož je tento nepoměr typický i pro trénovací data, která jsou vybalancovaná, důvod bude, že síť nedokázala odhalit vzory potřebné pro rozeznání, kdy cena poroste nebo klesne, a proto upřednostňoval kategorii 2. Z těchto důvodů nelze tyto modely aplikovat v praxi.



(a) 2 LSTM vrstvy o 32 neuronech

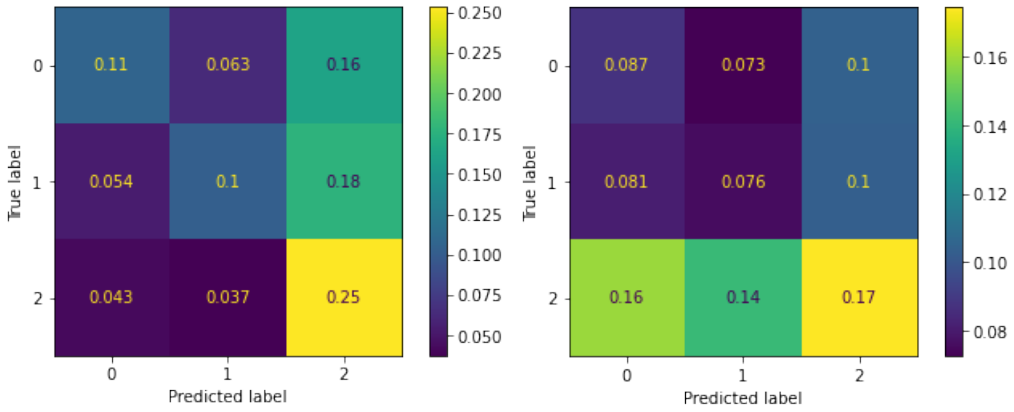


(b) 3 LSTM vrstvy o 32 neuronech

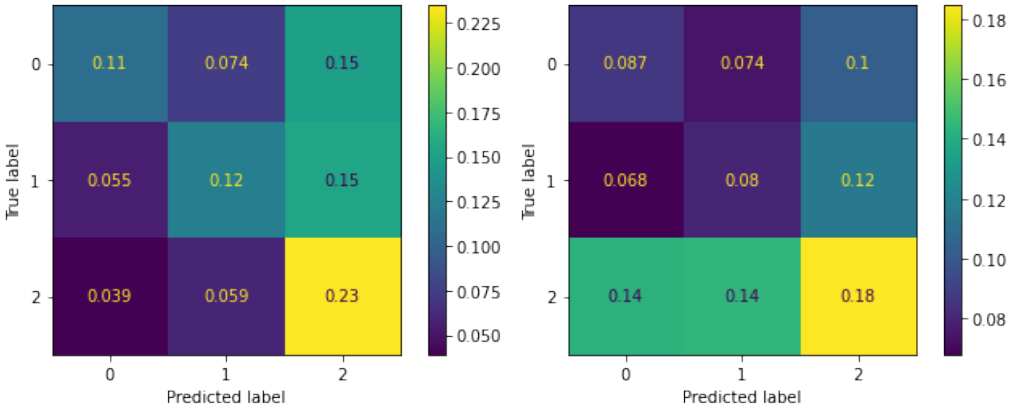


(c) 3 LSTM vrstvy o 64 neuronech

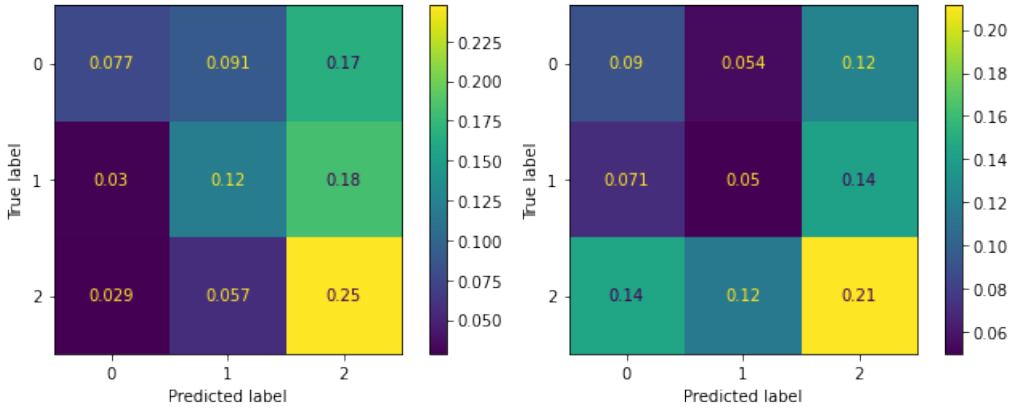
Obrázek 3.7: Vývoje metrik a ztrátových funkcí během trénování architektur na obrázku 3.6



(a) 2 LSTM vrstvy o 32 neuronech

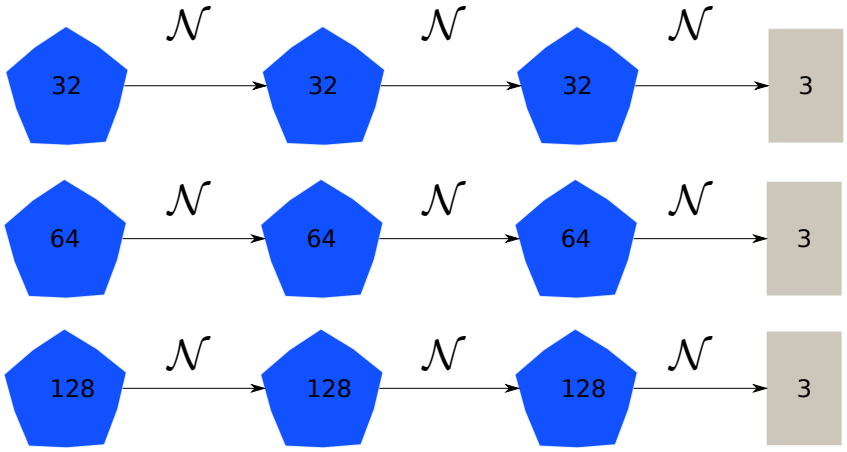


(b) 3 LSTM vrstvy o 32 neuronech

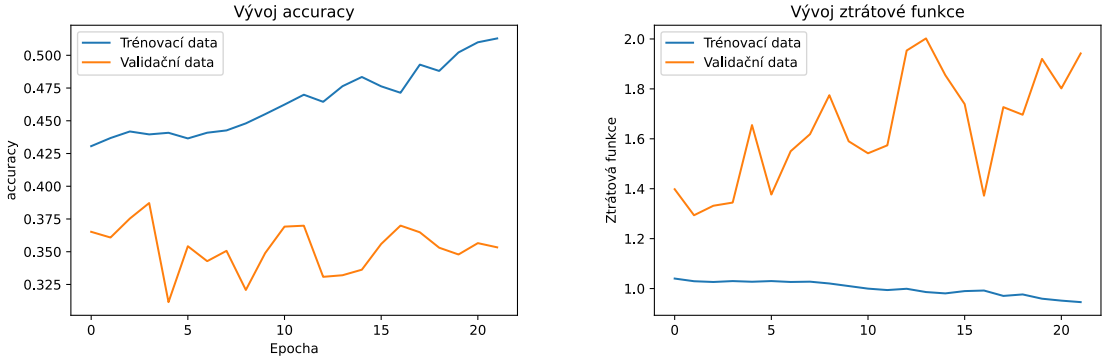


(c) 3 LSTM vrstvy o 64 neuronech

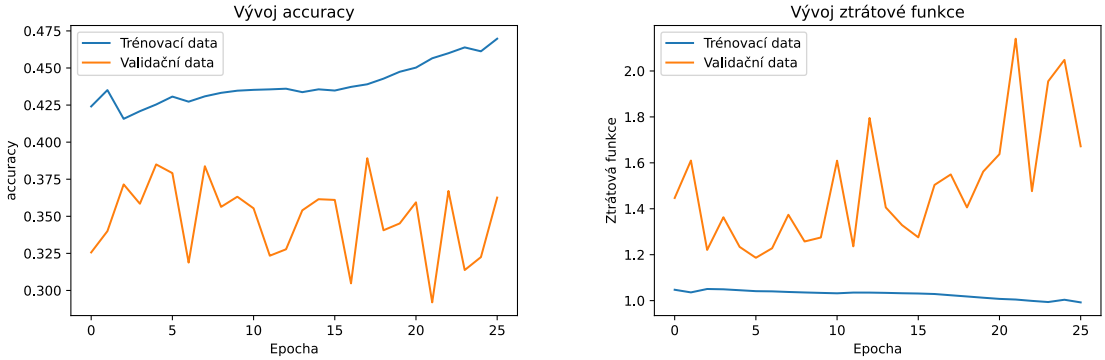
Obrázek 3.8: Matice záměn architektur na obrázku 3.6 pro trénovací (vlevo) a validační (vpravo) data



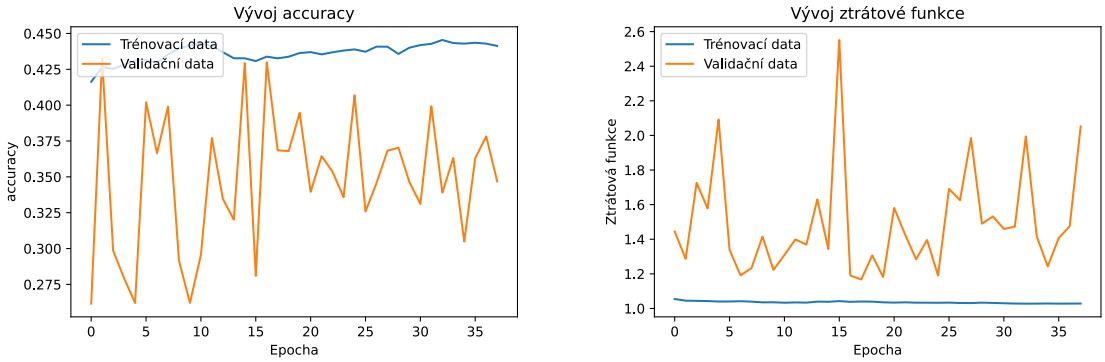
Obrázek 3.9: Vizualizace architektur použitých neuronových sítí s přidáním dávkové normalizace, viz podsekce3.2.5



(a) 3 LSTM vrstvy o 32 neuronech

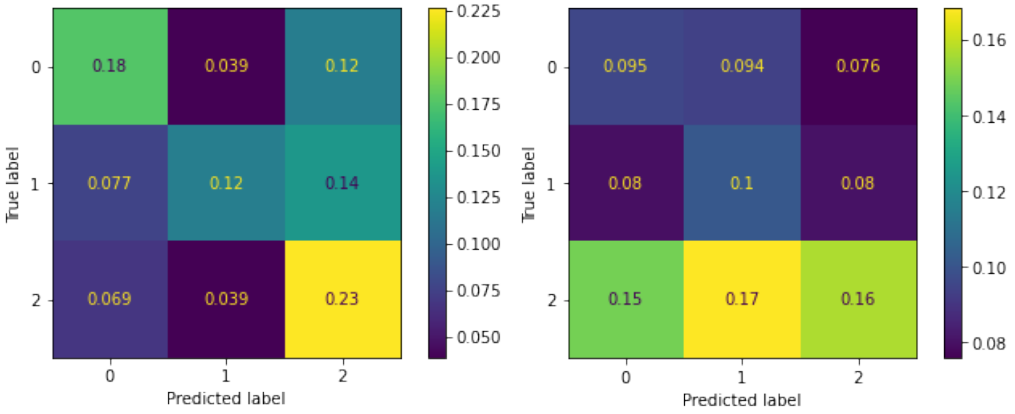


(b) 3 LSTM vrstvy o 64 neuronech

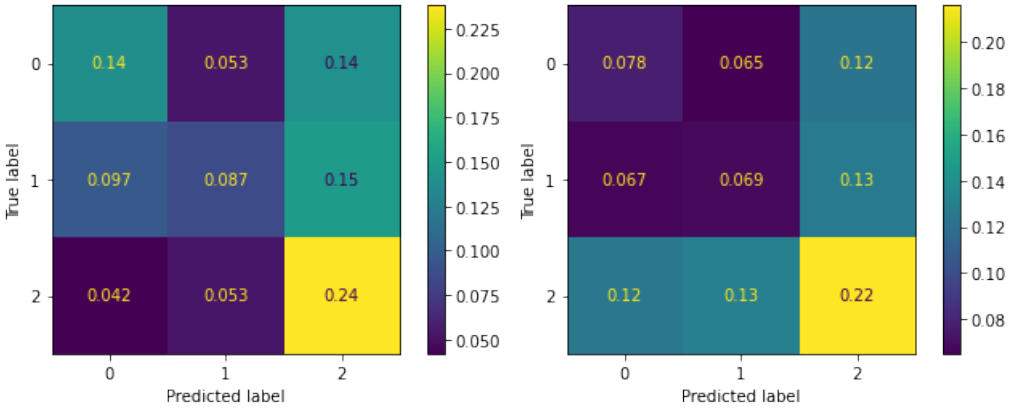


(c) 3 LSTM vrstvy o 128 neuronech

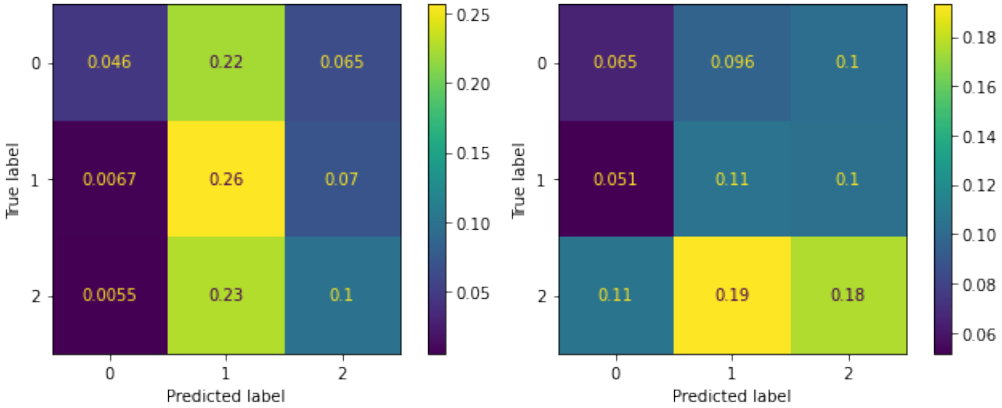
Obrázek 3.10: Vývoje metrik a ztrátových funkcí během trénování architektur na obrázku 3.9



(a) 3 LSTM vrstvy o 32 neuronech

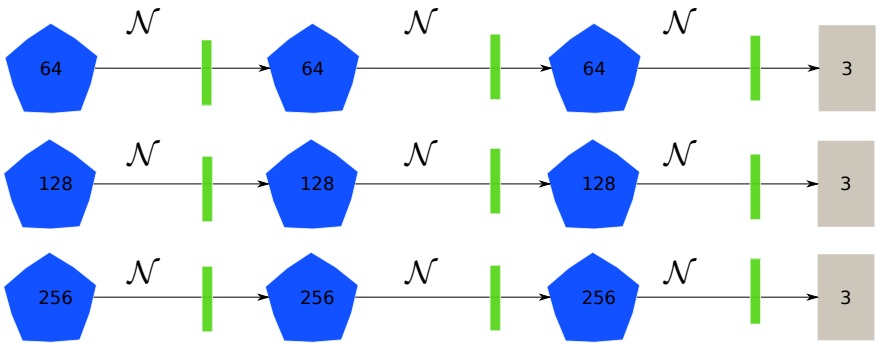


(b) 3 LSTM vrstvy o 64 neuroench

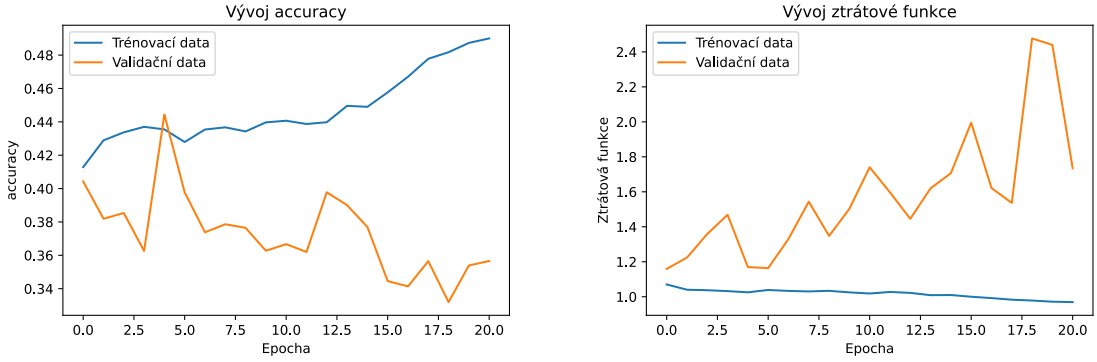


(c) 3 LSTM vrstvy o 128 neuronech

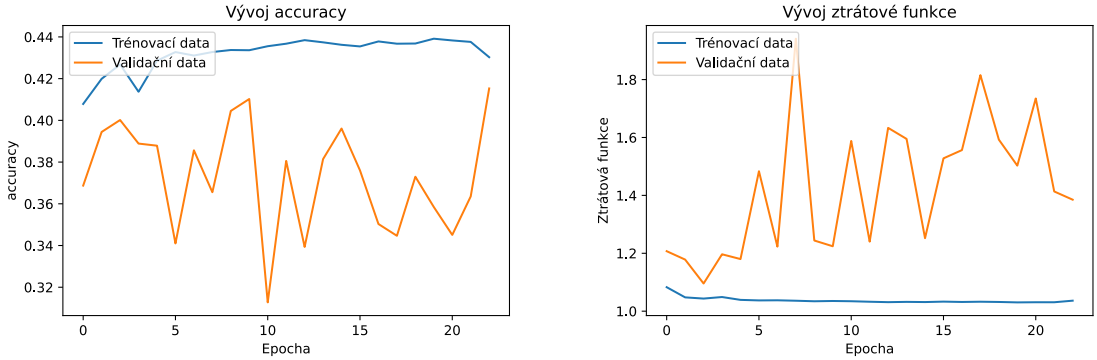
Obrázek 3.11: Matice záměn architektur na obrázku 3.9 pro trénovací (vlevo) a validační (vpravo) data



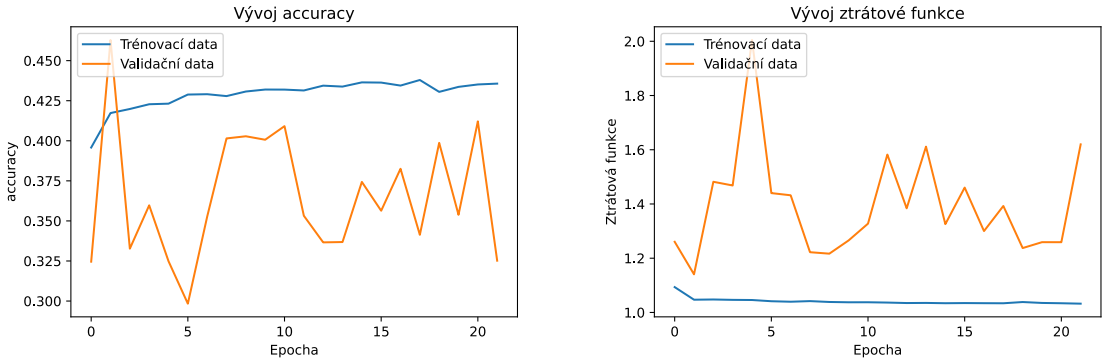
Obrázek 3.12: Vizualizace architektur použitých neuronových sítí s přídáním dávkové normalizace a dropoutu, viz podsekce 3.2.5



(a) 3 LSTM vrstvy o 64 neuronech

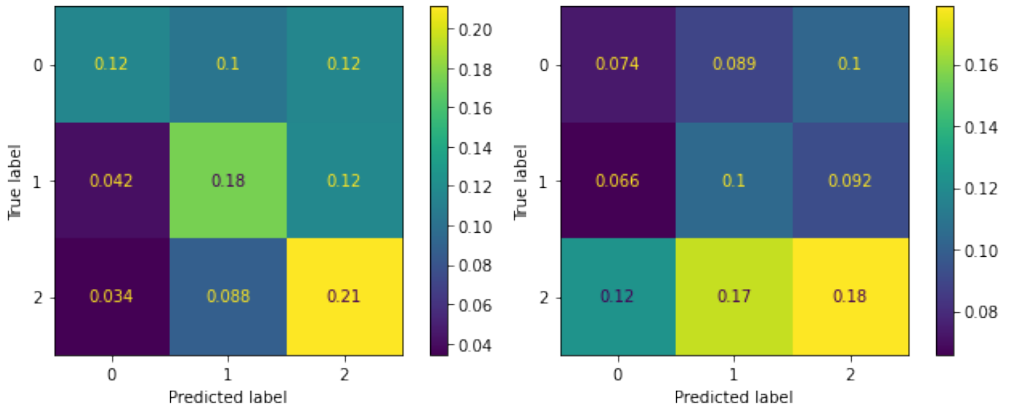


(b) 3 LSTM vrstvy o 128 neuronech

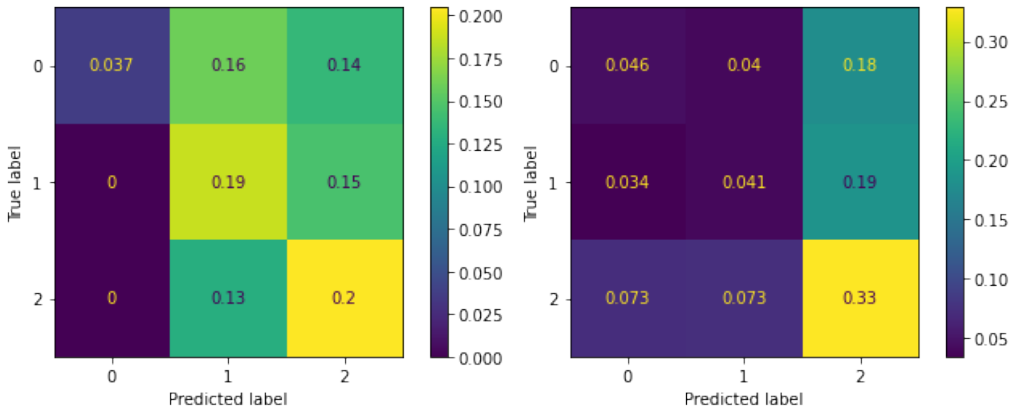


(c) 3 LSTM vrstvy o 256 neuronech

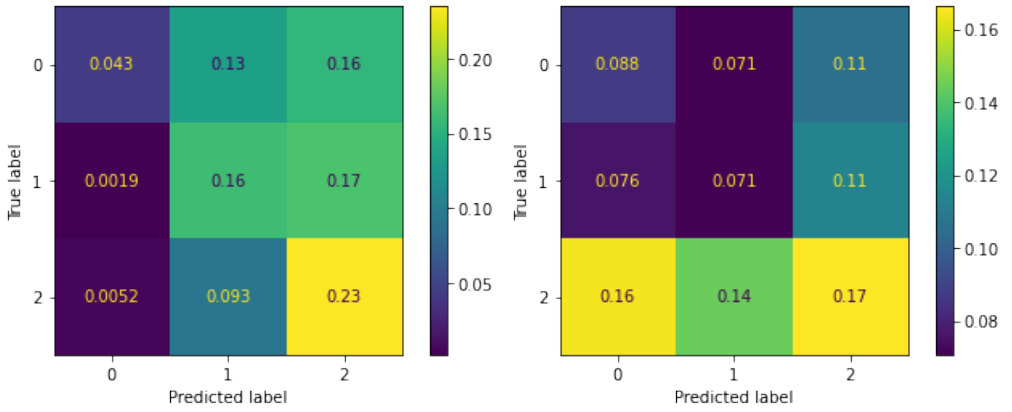
Obrázek 3.13: Vývoje metrik a ztrátových funkcí během trénování architektur na obrázku 3.12



(a) 3 LSTM vrstvy o 64 neuronech

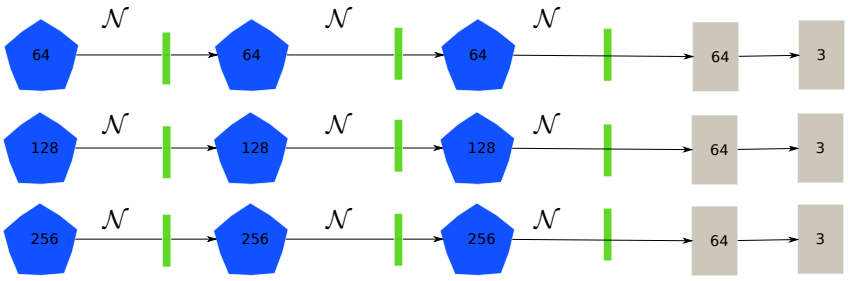


(b) 3 LSTM vrstvy o 128 neuronech

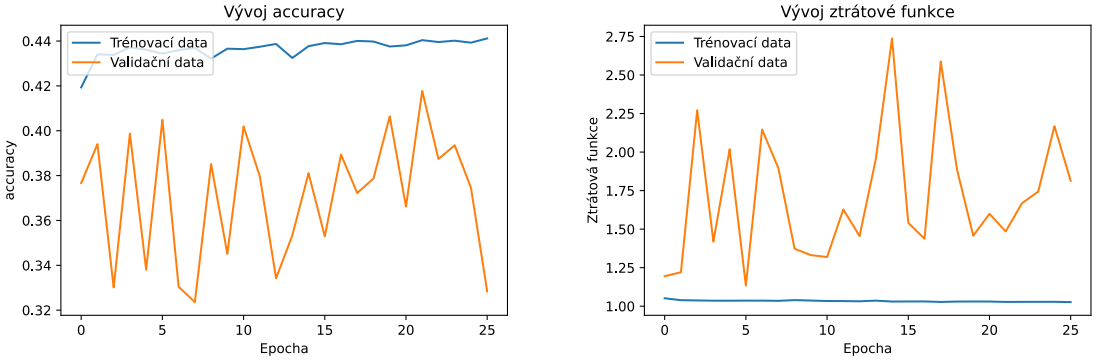


(c) 3 LSTM vrstvy o 256 neuronech

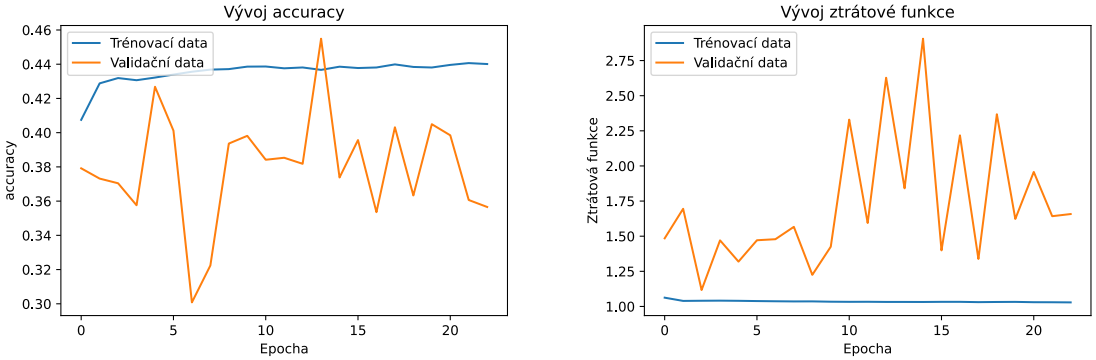
Obrázek 3.14: Matice záměn architektur na obrázku 3.12 pro trénovací (vlevo) a validační (vpravo) data



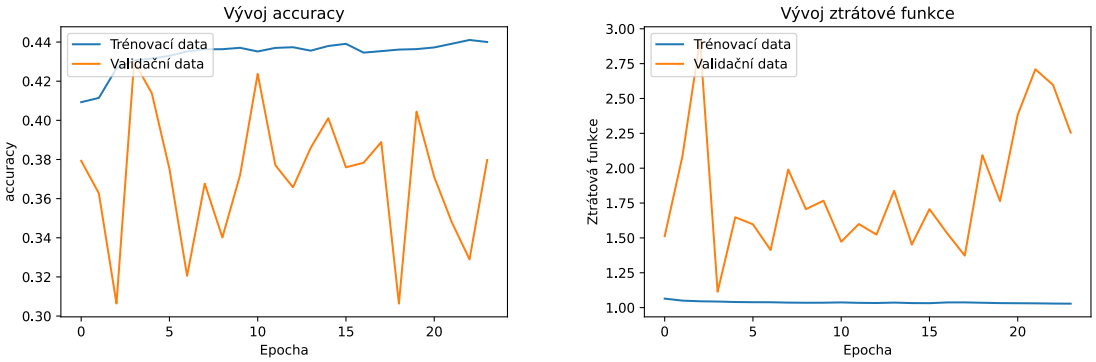
Obrázek 3.15: Vizualizace architektur použitých neuronových sítí s přidáním předposlední vrstvy Dense, viz podsekce 3.2.6



(a) 3 LSTM vrstvy o 64 neuronech

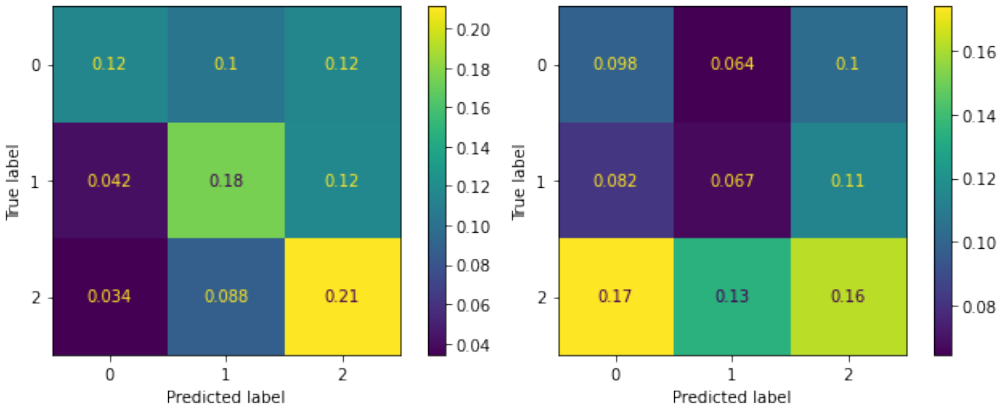


(b) 3 LSTM vrstvy o 128 neuronech

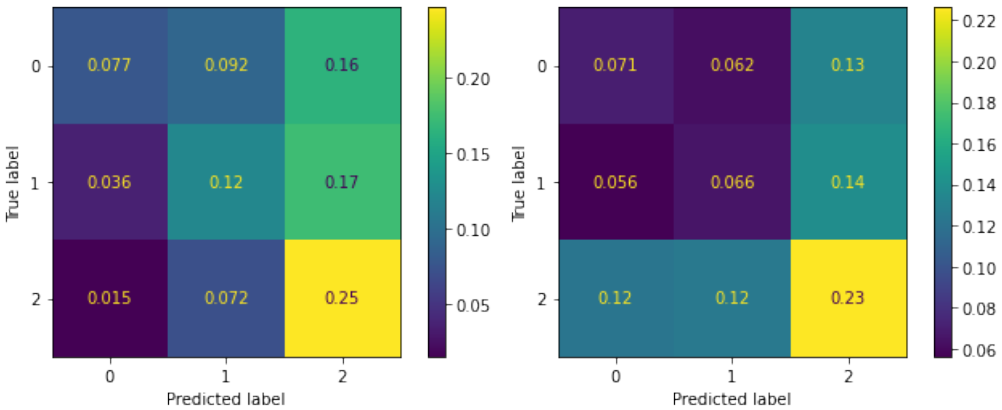


(c) 3 LSTM vrstvy o 256 neuronech

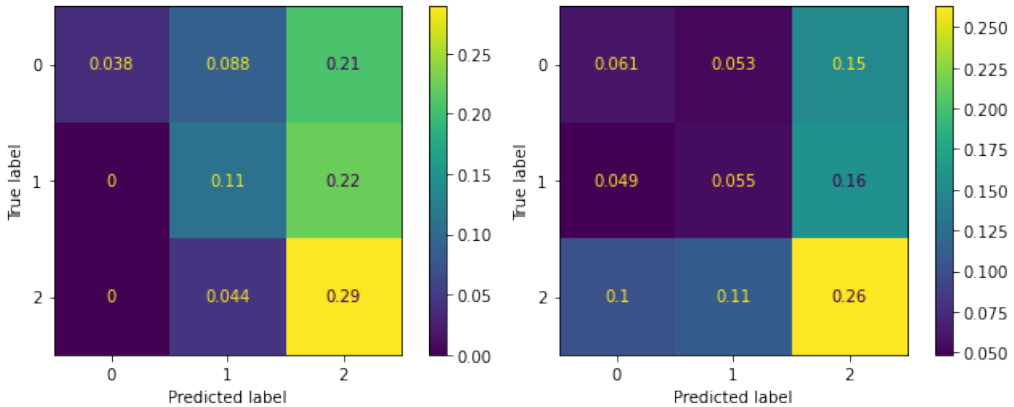
Obrázek 3.16: Vývoje metrik a ztrátových funkcí během trénování architektur na obrázku 3.15



(a) 3 LSTM vrstvy o 64 neuronech



(b) 3 LSTM vrstvy o 128 neuronech



(c) 3 LSTM vrstvy o 256 neuronech

Obrázek 3.17: Matice záměn architektur na obrázku 3.15 pro trénovací (vlevo) a validační (vpravo) data

Závěr

Cílem práce bylo teoretické seznámení se s dopřednými a rekuretními neuronovými sítěmi a jejich použití na časové řady z kryptoměnové burzy. Důležitou součástí toho procesu bylo předzpracování časových řad, bez kterého by nebylo možné natrénovat model. Dále byl zkoumán vliv nastavení různých hyperparametrů na funkčnost sítě, jejichž výsledky byly interpretovány.

První dvě kapitoly se věnují teoretické části. Algoritmy lineární a logistické regrese nebyly použity, ale sloužily k demonstraci základních pojmů strojového učení. Architektury neuronových sítí vedly ke konstrukci rekurentního typu neuronových sítí, který byl použit v implementaci modelu. Dále je rozebrán problém přeučení s metodami, které tomuto problému předcházejí.

Třetí kapitola se skládá z úvodu, jaký software a datové sady byly použity. Dále se věnuje předzpracování časové řady a implementaci modelu neuronové sítě. Byly zkoumány architektury LSTM rekurentní neuronové sítě na časových řadách cenového vývoje kryptoměnové burzy. Na konci kapitoly jsou interpretovány výsledky jednotlivých různých nastavení hyperparametrů.

Různé modely neuronových sítí dokázaly na validačních datech přesáhnout 33% a volba různých hyperparametrů úspěšnosti predikcí příliš neměnila. Vývoje validačních ztrátových funkcí, ale při trénování neměly stabilní chování a nejlépe dokázaly sítě poznat, že cena se nezmění. S přihlédnutím k těmto faktům lze konstatovat, že tyto modely nelze aplikovat v praxi. Nicméně možností, které by mohly vést k lepším výsledků, je několik. Je možné přidat další vlastnosti do vstupního vektoru, jako je například objem transakcí během časového úseku, použití více měnových párů, použití fundamentu (tj. analýza klíčových slov ze zpravodajských serverů a sociálních sítí), apod. Dále není zřejmé, zda časové rozlišení dat 1 minuty je vhodné a zda zvolený časový rámec predikce je kompatibilní s informacemi obsaženými ve zvolené délce sekvence. Možné zlepšení může způsobit i použití jiné architektury neuronové sítě.

Literatura

[1] Charu C. Aggarwal. *Neural Networks and Deep Learning*. Springer, 2018. ISBN 978-3-319-94463-0.

[2] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Inc, 2019. 2. vydání. ISBN 978-1-492-03264-9.

[3] Anuja Nagpal. *L1 and L2 Regularization Methods. Towards Data Science*. Říjen 2017. https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c (cit. 30. 06. 2021)

[4] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov: *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. Journal of Machine Learning Research 15(56), 2014, p. 1929−1958. ISSN: 1533-7928. https://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf (cit. 12. 04. 2021)

[5] Arthur Samuel. *Some Studies in Machine Learning Using the Game of Checkers.* IBM J. Res. 3, 1959, p. 210-229. https://www.semanticscholar.org/paper/Some-Studies-in-Machine-Learning-Using-the-Game-of-Samuel/e9e6bb5f2a04ae30d8ecc9287f8b702eedd7b772?p2df (cit. 12. 04. 2021)

[6] Michael Collins: *Convergence Proof for the Perceptron Algorithm*. Department of Computer Science, Columbia University. Únor 2012. http://www.cs.columbia.edu/~mcollins/courses/6998-2012/notes/perc.converge.pdf (cit. 12. 04. 2021)

[7] Scikit-learn Documentation. https://scikit-learn.org/stable/ (cit. 12. 04. 2021)

[8] Tensorflow Documentation. https://www.tensorflow.org/ (cit. 12. 04. 2021)

[9] Google Brain research team. https://research.google/teams/brain/ (cit. 12. 04. 2021)

[11] Keras Documentation. https://keras.io/ (cit. 12. 04. 2021)

[12] Pytorch Documentation. https://pytorch.org/ (cit. 12. 04. 2021)

[13] Facebook AI research team. https://ai.facebook.com/ (cit. 12. 04. 2021)

[14] Jason Brownlee. *Difference Between a Batch and a Epoch in a Neural Networks*. Říjen 2019. https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/ (cit. 12. 04. 2021)

[15] Kurt Hornik, Maxwell Stinchcombe, Halbert White. *Multilayer feedforward networks are universal approximators.* Neural Networks 2(5), 1989, p. 359–366. ISSN 0893-6080.

[16] Simeon Kostadinov. *Understanding Encoder-Decoder Sequence to Sequence Model*. Únor 2019. https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346 (cit. 18. 05. 2021)

[17] User Guide scikit-learn. https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (cit. 19. 05. 2021)

[18] Jason Brownlee. *Difference Between Classification and Regression in Machine Learning*. Květen 2019. https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/ (cit. 19. 05. 2021)

[19] User Guide scikit-learn. https://scikit-learn.org/stable/modules/linear_model.html (cit. 30. 06. 2021)

[20] The Investopedia Team. *The Basics of Bollinger Bands*. Říjen 2020. https://www.investopedia.com/articles/technical/102201.asp (cit. 25. 6. 2021)

[21] Admiral Markets. *Breakout strategie s Bollinger Bands a Admiral Keltner indikátory.* Červen 2017. https://www.fxstreet.cz/zpravodajstvi-90470.html (cit. 25. 06. 2021)