Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Analýza textu metodami hlubokého učení

Bc. Jan Gemperle

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Jan Gemperle

Systémové inženýrství a informatika
Informatika

Název práce

Analýza textu metodami hlubokého učení

Název anglicky

Text analysis with deep learning methods

Cíle práce

Cílem práce je prokázat efektivitu moderních metod analýzy textu založenou na použití algoritmů hlubokého učení. Student si vybere úlohu, k jejímuž řešení je analýza textu nezbytná a pro kterou může získat (např. na internetu) vhodný datový soubor. Navrhne metodiku řešení úlohy založenou na algoritmech hlubokého učení, na datovém souboru ověří její efektivitu a porovná ji s efektivitou, kterou lze získat pomocí klasických algoritmů.

Metodika

Student se seznámí se stavem výzkumu v oblasti analýzy textu metodami hlubokého učení. Vybere si problém, který lze metodami hlubokého učení řešit a k němuž si může opatřit adekvátní datový soubor. Navrhne metodu řešení a použité algoritmy založené na hlubokém učení realizuje v open source softwaru tensorflow a keras. Efektivitu navržené metody řešení ověří na datovém souboru a získané výsledky porovná s výsledky, které lze získat pomocí některého ze standardně používaných klasických algoritmů (např. hustých vrstvenných neuronových sítí)

Doporučený rozsah práce

60 stránek

Klíčová slova

text analysis, deep learning, convolution networks

Doporučené zdroje informací

Aggarwal Ch.: Neural Networks and Deep Learning, Springer, 2018
Chollet F.: Deep learning v jazyku Python, Grada, 2019

Předběžný termín obhajoby

2020/21 LS – PEF

Vedoucí práce

doc. Ing. Arnošt Veselý, CSc.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 19. 11. 2020

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 19. 11. 2020

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 14. 03. 2021

Čestné prohlášení

Prohlašuji, že svou diplomovou práci „Analýza textu metodami hlubokého učení" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 28.03.2021 _____

Poděkování

Rád bych touto cestou poděkoval doc. Ing. Arnoštu Veselému, CSc. za odborné vedení a přínosné konzultace, které mi byly poskytnuty.

Analýza textu metodami hlubokého učení

Abstrakt

Tato diplomová práce se zabývá analýzou textu za pomoci metod hlubokého učení. Pro ověření přístupu byla zvolena úloha binární klasifikace nad množinou dat Fake News, která obsahuje označené pravdivé a nepravdivé novinové články.

V teoretické části jsou uvedeny základní definice o neuronových sítích a jejich využití ve vrstvách hlubokého učení. Teoretická část dále obsahuje informace o základním principu algoritmu zpětného šíření. Práce se dále zaměřuje na aktivační funkce, metriky a základní typy vrstev neuronových sítí, které jsou následně použity v praktické části.

Praktická část se zabývá vytvořením modelů, jejich učením na tréninkových a validačních datech a následném ladění, kdy dochází k postupnému upravování parametrů sítě pro získání co nejlepších výsledků.

Klíčová slova: strojové učení, hluboké učení, neuronové sítě, binární klasifikace, Fake News, Keras, TensorFlow

Text analysis with deep learnings methods

Abstract

This diploma thesis deals with the analysis of a text with the help of a deep learning method. To control access, the task of binary classification over the Fake News data set was chosen, which contains marked true and false newspaper articles.

The theoretical part presents the basic definitions of neural networks and their use in the layers of deep learning. The theoretical part also contains information about the basic principle of the backpropagation algorithm. The work also focuses on activation functions, metrics, and basic types of neural network layers, which are then processed in the practical part.

The practical part deals with the creation of models, their learning in training and validation data and subsequent tuning, at any time to gradually adjust the network parameters to obtain the best possible results.

Keywords: machine learning, deep learning, neural networks, binary classification, Fake News, Keras, TensorFlow

Obsah

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

Zkratka	Název
API	Application Programming Interface
CPU	Central Processing Unit
GPU	Graphics Processing Unit
AI	Artificial Intelligence
ML	Machine Learning
ROC	Receiver Operating Characteristic
AUC	Area Under Curve
CNN	Convolutional Neural Network

1 Úvod

V současné době si už chod společnosti bez umělé inteligence nedovedeme představit, i když ani nevíme, že ji používáme. V posledním desetiletí prošel tento obor významným rozvojem a začal se přemisťovat z výzkumných pracovišť do podnikové sféry, kde začal uplatňovat své místo na trhu práce.

Samozřejmě tento vývoj nezůstal bez odezvy, s obrovskou vlnou popularity se začaly objevovat i negativní ohlasy, které se snažily varovat před nebezpečím umělé inteligence. S přibývajícím časem začaly vznikat stále důmyslnější nástroje (DeepFake), které tato varování pouze potvrzovaly. [1]

Přesto strojové učení nemusí sloužit jen jako nástroj pro vkládání obrázků veřejně známých osob do kompromitujících situací. Pokud je algoritmus správně nastaven, může se stát velmi užitečným nástrojem pro usnadnění lidské práce.

Diplomová práce je rozdělena do tří částí. První část popisuje vztah mezi umělou inteligencí a hlubokým učením. Dále obsahuje teoretické informace o modelech neuronových sítí.

Ve druhé části je představena úloha rozpoznání pravdivých a nepravdivých novinových článků za pomoci čtyř modelů využívající metody hlubokého učení.

Ve třetí části jsou porovnány výsledky získané z testovací množiny dat. Na základě vyhodnocení je vybrán model s nejvyšší úspěšností při klasifikaci nepravdivých novinových článků.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem této práce je prokázat efektivitu moderních metod analýzy textu založenou na použití algoritmů hlubokého učení. K prokázání efektivity byla vybrána úloha rozpoznání pravdivých a nepravdivých novinových článků.

Pro úspěšní naplnění hlavního cíle bylo nezbytné splnit následující dílčí cíle:

- Vybrání neuronových vrstev, které dokážou úspěšně detekovat rozdíl mezi pravdivým a nepravdivým novinovým článkem.
- Vytvoření klasifikačních modelů, které úspěšně překonají referenční hodnotu náhodného klasifikátoru pro úlohu binární klasifikace.

2.2 Metodika

Tato práce se zabývá analýzou textu pomocí metod hlubokého učení. Proto byla pro praktickou část vybrána úloha binární klasifikace na základě porozumění významu textu poskytnutého z datové množiny Fake News získané ze serveru Kaggle.

Úloha bude řešena pomocí několika modelů sestavených z neuronových sítí, jejichž architektura bude zvolena na základě analýzy a syntézy poznatků z odborné literatury. K sestavení modelů budou využity open source knihovny TensorFlow a Keras. Pro učení modelu budou využita trénovací a validační data získaná z rozdělené datové množiny Fake News.

Efektivita navržených modelů bude ověřena na testovacích datech. Následně budou dosažené výsledky jednotlivých modelů porovnány pomocí vypočítaných metrik získaných z matice záměn.

3 Teoretická východiska

Představa umělé inteligence nabízí budoucnost plnou technických vymožeností, mezi kterými nemůžou chybět samořídící vozy, obsluhující roboti, drony doručující balíčky a mnohé další samostatně uvažující věci, které se zvládají samostatně rozhodovat bez zásahu lidí.

Koncept umělé inteligence nebo zkráceně AI (Artificial Intelligence) se stále častěji začal skloňovat v médiích a postupem času se začal dostávat do širšího povědomí lidí. Příkladem může být novinový článek z New York Times z roku 2010, který oznamoval testování samořídících vozidel na silnicích ve Spojených státech. [2]

Význam AI začal být v životě lidí nesporný. Tento potenciál neunikl velkým soukromým korporacím, mezi které patří například Google, Facebook nebo Amazon. A tyto společnosti podnítily masivní investice, a zahájily tak „závod o AI“. [3]

3.1 Umělá inteligence

Umělou inteligenci lze nejjednodušeji popsat jako samostatně vylepšující se systém nebo přístroj, který napodobuje lidskou inteligenci k vyřešení zadaných úkolů.

Pro umělou inteligenci existuje velké množství využití, od chatovacích robotů pro komunikaci se zákazníky, až po systémy pomáhající při vyvíjení léku na covid-19. Umělá inteligence se tedy převážně využívá pro analýzu velkého množství dat. [4]

3.1.1 Využití umělé inteligence

Využívání umělé inteligence dokáže zvýšit výkonnost a produktivitu podniku díky automatizaci procesů nebo úloh, které by jinak vyžadovaly lidské zdroje.

Dalším využitím umělé inteligence je analyzovat a dávat do kontextu rozsáhlá data, která by člověk zpracovat nedokázal. Této schopnosti začaly využívat obchodní společnosti jako Netflix, YouTube nebo Spotify. Každá z těchto společností se zaměřuje na výběr specifického obsahu pro individuálního uživatele.

Obecně lze říct, že umělá inteligence může přinést přínos jakémukoliv podniku, který dokáže využít:

– optimalizace cen na základě preferencí zákazníků,

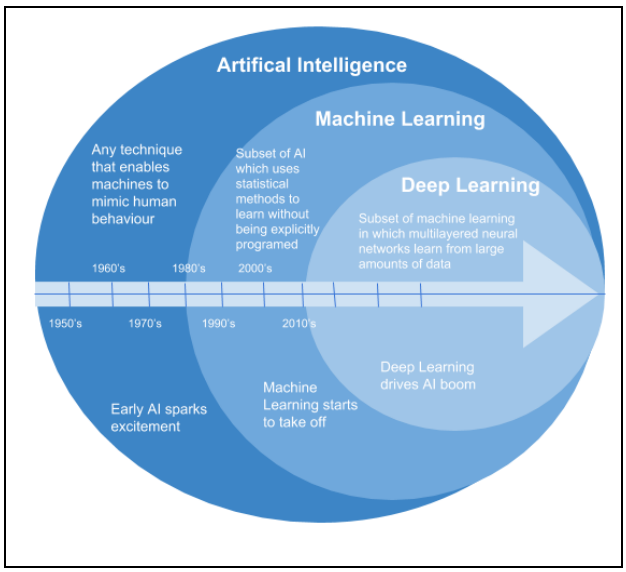
– výpočet predikcí budoucího nákupu zákazníků,

– různé klasifikace (obrazu, textu, aj.). [5]

3.1.2 Umělá inteligence vs. strojové učení

Umělá inteligence se stala všeobecným pojmem pro systémy, které v minulosti vyžadovaly lidské vstupy. Tento pojem se často zaměňuje s pojmy, které jsou jeho součástmi jako je například strojové učení nebo hluboké učení. Z obrázku 1 je patrné, že hluboké učení (Deep Learning) je součástí strojového učení (Machine Learning), které spadá pod umělou inteligenci (Artifical Intelligence). [6]

Obrázek 1 - Součásti AI.



Zdroj: https://towardsdatascience.com/what-is-artificial-intelligence-all-about-anyway-b57c7eb75f5f

3.2 Strojové učení

Algoritmy jsou sledem pokynů použitých k vyřešení problému. Algoritmy, vyvinuté programátory pro výuku počítačů, které řeší nové úkoly, jsou stavebními kameny moderního digitálního světa. Počítačové algoritmy organizují obrovské množství dat do informací a služeb na základě určitých pokynů a pravidel. Myšlenkou strojového učení je, že pravidla jsou utvářena algoritmy učení, nikoliv programátory.

Programátoři namísto psaní jednotlivých přesných instrukcí poskytují algoritmu pokyny, díky nimž je algoritmus schopen se učit z dat bez dalších pokynů programátora. Díky tomu je možné počítač použít pro nové, komplikované úlohy, které nelze ručně naprogramovat.

Základním procesem strojového učení je dát tréninková data do algoritmu učení. Algoritmus učení poté generuje novou sadu pravidel na základě závěrů z dat. Tím je vygenerován nový algoritmus, označovaný jako model strojového učení. Použitím různých tréninkových dat lze stejný algoritmus učení použít pro generování různých modelů.

Hlavní silou strojového učení je vyvodit nové instrukce z dat. To ovšem také dokazuje, jak kritickou roli hrají vybraná data. Čím více údajů je k dispozici pro zaškolení algoritmu, tím více se učí. [7]

Algoritmy strojového učení lze obecně rozdělit do tří obecných typů:

- řízené učení,
- neřízené učení,
- posilované učení.

3.2.1 **Řízené učení**

Algoritmy řízeného učení se ve strojovém učení používají nejčastěji. Algoritmu jsou předkládána tréninková data a skutečné cíle, na jejichž základě si algoritmus vytvoří vlastní pravidla, díky nimž bude schopen odhadnout výstup na nových datech, ke kterým již skutečné cíle přiložené nejsou. Mezi příklady řízeného učení patří algoritmy pro klasifikaci do dvou i více tříd nebo pro regresi.

Klasifikace právě do dvou tříd se nazývá binární klasifikace. V tomto typu úlohy jde o správné rozřazení vstupních dat právě do dvou kategorií. Klasifikace do více tříd je obdobná jako binární klasifikace s rozdílem, že data jsou třízena do více než dvou kategorií.

Regrese slouží k odhadování výstupní hodnoty na základě přijatých vstupních hodnot. Cílem regresní úlohy je tedy odhadnout změnu výstupní hodnoty na základě změn vstupních hodnot.

3.2.2 Neřízené učení

Neřízené učení využívá nezávislejší přístup, ve kterém se počítač učí rozpoznat složité procesy a vzorce bez toho, aby mu člověk poskytoval bližší trvalé vedení. Mezi příklady neřízeného učení patří například shlukování neoznačených dat.

Shlukování je proces seskupování případů s podobnými vlastnostmi do jednotlivých kategorií. [8]

3.2.3 Posilované učení

Algoritmus s posilovaným učením interaguje s dynamickým prostředím, které poskytuje zpětnou vazbu. Na základě této zpětné vazby algoritmus určí akci, která maximalizuje nějakou odměnu.

V současné době se posilované učení využívá převážně v herním průmyslu, například ve hře Go.

3.3 Hluboké učení

Hluboké učení umožňuje skládat výpočetní modely z více vrstev, a tak se algoritmus může naučit odlišné reprezentace dat s více úrovněmi abstrakce. Tyto metody výrazně zlepšily metody rozpoznávání řeči, rozpoznávání vizuálních objektů, detekci objektů a mnoho dalších domén, mezi které patří například genomika.

Hluboké učení odhaluje složitou strukturu ve velkých souborech dat pomocí algoritmu zpětného šíření, který ukazuje, jak by měl stroj změnit své vnitřní parametry, které se používají k výpočtu reprezentace v každé vrstvě z reprezentace v předchozí vrstvě.

Metody hlubokého učení využívají víceúrovňovou reprezentaci dat, získanou z jednoduchých nelineárních modulů, kde každý modul transformuje svůj vstup na abstraktnější úroveň. Moduly jsou složeny do celku, kde výstupy jednotlivých modulů slouží jako vstupy modulů následujících. Díky tomu se může model naučit rozpoznávat velmi složité funkce.

U klasifikačních úloh vyšší vrstvy modelu zesilují aspekty vstupu, které jsou důležité pro roztřídění do správných tříd a potlačují nedůležité aspekty. Klíčovou vlastností hlubokého učení je, že tyto aspekty jsou získány z dat pomocí algoritmu učení. [9]

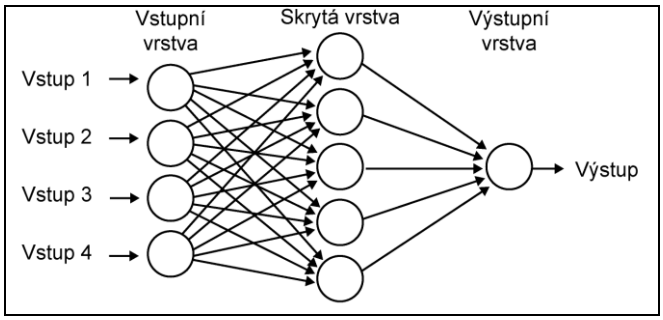
3.4 Umělé neuronové sítě

Umělá neuronová síť se skládá z matematických neuronů, kde každý neuron zpracovává váhově ohodnocené vstupní signály a generuje výstup.

Jednotlivé neurony v neuronových sítích jsou uspořádány do topologické struktury. Komunikace mezi neurony probíhá pomocí orientovaných ohodnocených spojů. Charakteristika neuronové sítě je dána typem neuronů, jejich topologickým uspořádáním a zvolenou strategií pro trénování sítě. [10]

Základní uspořádání neuronové sítě je dopředné, protože se signál šíří v síti jedním směrem. Od vstupní vrstvy k výstupní vrstvě, jak je uvedeno na obrázku 2.

Obrázek 2 - Síť umělých neuronů.



Zdroj: https://portal.matematickabiologie.cz/index.php?pg=analyza-a-hodnoceni-biologickych-dat--umela-inteligence--neuronove-site-jednotlivy-neuron--uvod-do-neuronovych-siti--koncept-umele-neuronove-site

Z obrázku 2 je patrné, že jednotlivé neurony jsou uspořádány do vrstev. Neurony v jedné vrstvě zpravidla propojeny nejsou, neurony v sousedních vrstvách jsou propojeny podle typu vrstvy.

Každý spoj mezi neurony představuje dráhu pro šíření signálu, je orientovaný a ohodnocený váhou, která pozměňuje intenzitu signálu.

První vrstva dopředné neuronové sítě se nazývá vstupní vrstva, poslední vrstva se nazývá výstupní vrstva.

Dopředné neuronové sítě lze obecně rozdělit do dvou kategorií. První kategorií jsou mělké sítě, které využívají pouze jednu skrytou vrstvu. Druhou kategorií jsou hluboké sítě, které využívají dvě a více skrytých vrstev. Z tohoto názvu vznikl také termín „*Hluboké učení*“. [11, s. 33]

U neuronových sítí je možné rozlišovat dvě fáze fungování. První fázi se říká fáze učení, kdy si síť za pomoci trénovacího algoritmu modifikuje své parametry. Poté následuje fáze, kdy síť produkuje výstupy na základě předložených vstupů. V této fázi se nastavení parametrů nemění.

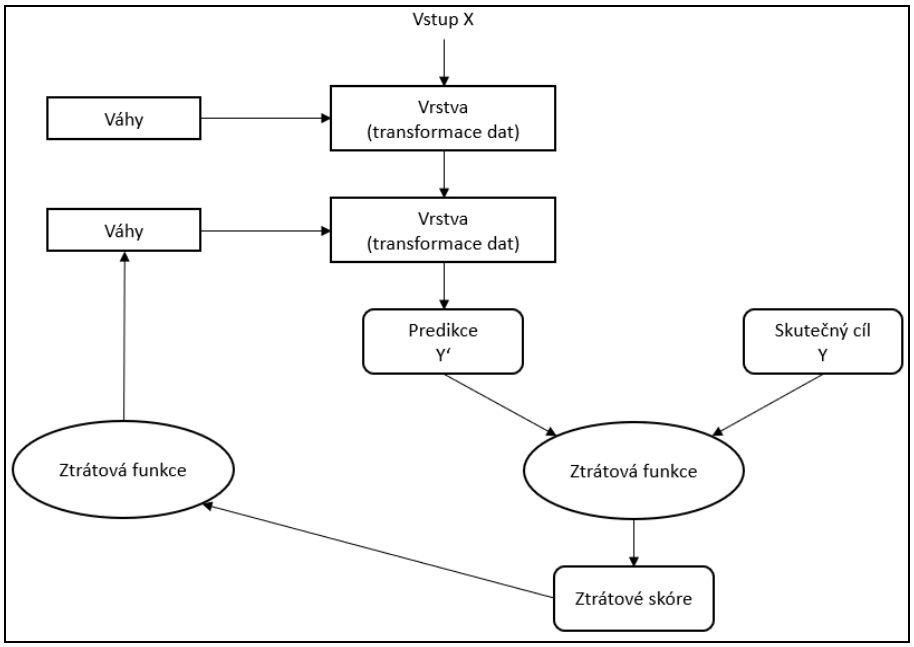
3.4.1 **Základní princip neuronových sítí**

Stavebním kamenem neuronových sítí jsou vrstvy. Tyto vrstvy extrahují reprezentace ze vstupních dat, které následně transformují do užitečnější formy a předají je na výstup. Hlavní myšlenkou hlubokého učení je seskládání několika jednoduchých neuronových vrstev v neuronovou síť tak, aby každá vrstva transformovala vstupní data do formy, která je pro počítač lépe čitelná.

Transformace dat v jednotlivých vrstvách probíhá na základě nastavených parametrů ve vrstvě. Těmto parametrům se říká váhy a jsou na začátku trénování náhodné, s postupem trénovacích příkladů jsou aktualizovány tak, aby predikovaný výstup co nejlépe odpovídal skutečnému cíli.

Aktualizace vah probíhá pomocí ztrátové funkce, která váhy ovlivňuje pomocí zpětné vazby, jak je popsáno na obrázku 3. Ztrátová funkce vypočítává ztrátové skóre vzdálenosti mezi predikcí sítě a skutečným cílem. Pro určení, zda se mají váhy v jednotlivých vrstvách snížit, nebo zvýšit, existuje ve zpětné vazbě optimalizátor využívající minidávkový stochastický gradientní sestup, který dokáže upravit všechny parametry vrstev najednou.

Obrázek 3 - Algoritmus zpětného šíření.



Zdroj: [11, s. 27]

3.4.2 **Reprezentace dat pro neuronovou síť**

Pro správné vložení vstupních dat do modelů neuronových sítí je nutné využívat multidimenzionální pole neboli tenzory. Do tenzorů jsou nejčastěji ukládána čísla, kde dimenze tenzoru popisuje tvar dat. Právě velikost dimenze určuje počet os každého tenzoru.

Náhled, jak by jednotlivé tenzory mohly být zobrazeny, je uveden na obrázku 4. Do modelu neuronové sítě jsou data vkládána postupně pomocí dávek, nikoliv najednou. Po vyhodnocení průchodu jedné dávky pomocí ztrátové funkce jsou aktualizovány hodnoty vah.

3.4.2.1 2D tenzory

Tento tenzor má dvě osy. Vizuálně je možné tyto tenzory zobrazit jako tabulky, kde jsou jednotlivé vektory o stejném tvaru psány pod sebe. Využívá se pro vektorová data, kde na první ose jsou jednotlivé příklady a na druhé ose jsou hodnoty konkrétního příkladu.

3.4.2.2 3D tenzory

Tento tenzor se skládá z pole po sobě jdoucích matic. Využívá se pro časové řady a sekvenční data, kde na první ose jsou jednotlivé příklady, na druhé ose je časový údaj a na třetí ose jsou hodnoty k časovým údajům.

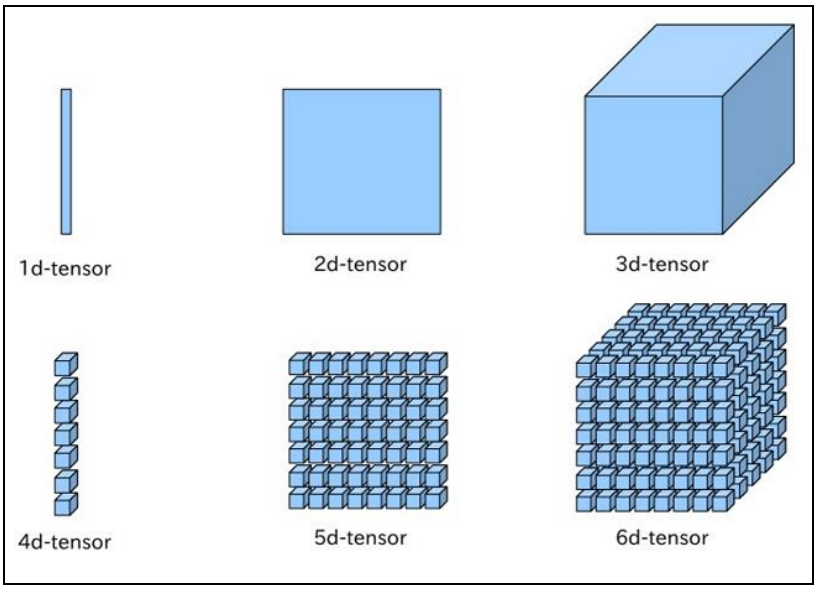
3.4.2.3 4D tenzory

Tento tenzor se skládá z pole po sobě jdoucích 3D tenzorů. Využívá se pro reprezentaci obrázků, kde na první ose jsou jednotlivé obrázky, na druhé ose je výška obrázku, na třetí ose je šířka obrázku a na čtvrté je kanál obrázku, který udává, z kolika barev je obrázek složen.

3.4.2.4 5D tenzory

Tento tenzor se skládá z pole po sobě jdoucích 4D tenzorů. Využívá se pro reprezentaci videa, kde první osa představuje počet jednotlivých videí, na druhé ose jsou rámy videa, na třetí ose je výška videa, na čtvrté ose je šířka videa a na páté je kanál, který udává, z kolika barev je obraz videa složen. [12]

Obrázek 4 - Zobrazení tenzorů



Zdroj: https://medium.com/@anoorasfatima/10-most-common-maths-operation-with-pytorchs-tensor-70a491d8cafd

3.4.3 **Vrstvy neuronové sítě**

Základní stavební jednotkou neuronové vrstvy je umělý neuron. Do neuronu může vést neomezené množství vstupů ohodnocených váhou *w*. Jedná se o vstupní data, nebo o výstup z předchozí vrstvy.

Poté, co jednotlivé neurony přijmou vstupní hodnoty x_i, vynásobí je váhami w_i. Pokud buňka neuronu obsahuje více než jeden součin vstupu x_i váhy w_i, tak tyto součiny sečte. Od tohoto součtu je odečten stanovený práh θ a následně je provedena transformace pomocí přechodové (aktivační) funkce *f*. Obecná rovnice vypadá takto:

$$y = f\left(\sum_{i=1}^{n} w_i \cdot x_i - \theta\right)$$

A právě z takto vytvořených umělých neuronů jsou poskládány neuronové vrstvy. Různé vrstvy provádějí na svých vstupech různé transformace. A jednotlivé vrstvy jsou vybírány podle řešených úkolů. [10]

3.4.3.1 Dense vrstvy

Vrstva Dense je základní vrstvou neuronové sítě. Tato vrstva umožňuje, aby byl každý její neuron napojen na každý neuron další neuronové sítě (Obrázek 2). Tímto propojením vznikají takzvané hustě propojené sítě. Hlavní výhodou této sítě je, že každá vrstva v síti se učí ze všech možných zjištěných příznaků vrstvy předchozí.

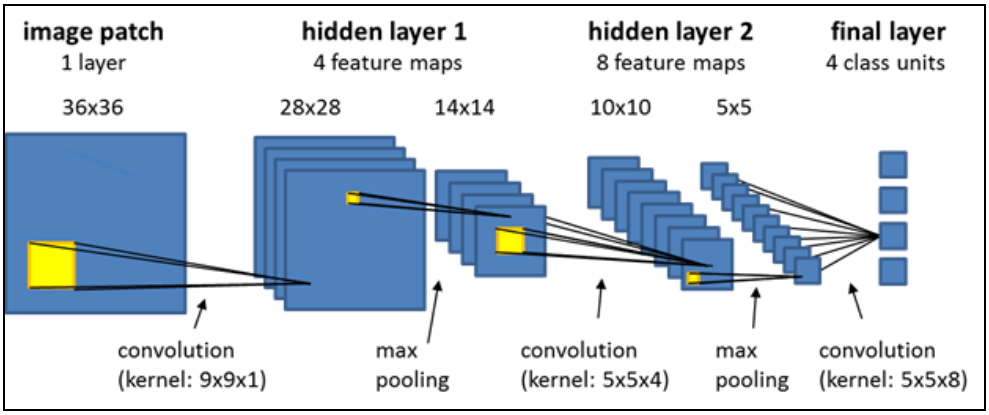
3.4.3.2 Konvoluční vrstvy

Konvoluční vrstvy pracují s 3D tenzory, kterým se říká mapy příznaků. Díky těmto mapám je možné detekovat vzory, které se objevují v různých místech příkladu. Dále je možné tyto vzory hierarchicky uspořádat, tedy v prvních vrstvách rozpoznat vzory a v dalších vrstvách rozpoznávat uskupení těchto vzorů.

Hlavní rozdíl oproti hustě propojeným vrstvám je v tom, že husté vrstvy se učí globální vzory, zatímco konvoluční vrstvy se učí lokální vzory o definované velikosti a hierarchickém uspořádání. [13]

Konvoluční vrstvy se používají převážně na zpracování počítačového vidění, přesto je možné je použít i na zpracování sekvencí, jako je například text.

Obrázek 5 - Schéma CNN



Zdroj: https://docs.ecognition.com/v9.5.0/eCognition_documentation/Modules/7%20Tutorials/Tutorial%20Overview.htm#Tutorial

Na obrázku 5 je vyobrazeno schéma 2D konvoluční neuronové sítě. Modré čtverce představují vstupní data do jednotlivých vrstev. Žluté čtverce představují mapy příznaků, kterými učící algoritmus po modrých čtvercích posouvá, aby objevil lokální vzory pomocí konvolučního jádra.

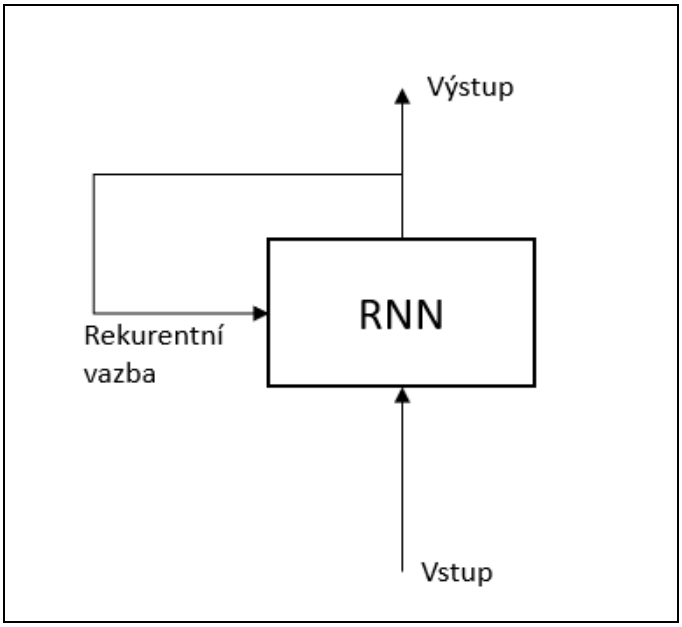
Pomocí dalších skrytých vrstev jsou pak objevovány další lokální vzory, které mohou mít hierarchické uspořádání. K tomu slouží operace sdružování (v tomto případě dle maxima), která převzorkuje a sníží mapu příznaků pomocí operace tensorového maxima. Výsledkem je snížení počtu koeficientů mapy příznaků, které je nutné zpracovat. [14]

3.4.3.3 Rekurentní vrstvy

Předchozí dvě zmíněné vrstvy se vyznačují tím, že nemají paměť. Každý vstup je v takovýchto sítích zpracováván nezávisle na vstupu předcházejícím. Opakem tohoto přístupu jsou rekurentní neuronové sítě, které si pomocí rekurentní vazby dokážou udržet informace o předchozích výstupech (Obrázek 6).

Rekurentní sítě udržují informace o předchozích výstupech do té doby, než dojde na vstupu RNN ke zpracování nové nezávislé sekvence.

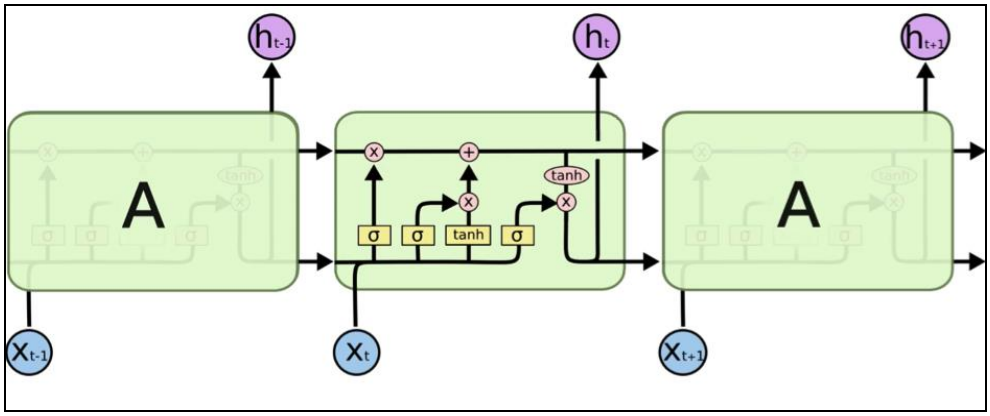
Obrázek 6 - RNN se zpětnou vazbou



Zdroj: [11, s. 185]

Sítě Long Short Therm Memory (Obrázek 7) jsou speciálním druhem rekurentních neuronových sítí. Struktura LSTM připomíná řetěz, kde každý článek obsahuje čtyři samostatné neuronové sítě. Dále každý článek obsahuje paměťové bloky zvané buňky. Informace jsou uchovávány v buňkách a k manipulaci s nimi slouží brány (Gates).

Obrázek 7 - LSTM vrstva



Zdroj: https://medium.com/bitgrit-data-science-publication/a-crash-course-in-sequential-data-prediction-using-rnn-and-lstm-c1a1eaf03463

Zapomínající brána (Forget Gate) slouží k odstranění již nedůležitých záznamů. O výsledků rozhoduje aktuální vstup a předchozí výstup, kde jsou oba vynásobeny váhovými maticemi. Výsledek poté transformuje aktivační funkce na binární výstup.

Pokud je výsledná hodnota rovna 0, informace je zapomenuta. Pokud je výsledná hodnota rovna 1, informace je zachována pro budoucí použití.

Vstupní brána (Input Gate) slouží k přidání užitečných informací do paměti buňky. Nejprve jsou informace regulovány pomocí funkce Sigmoid a hodnoty jsou uchovávány, nebo zapomínány jako v zapomínající bráně. Následně je vytvořen vektor pomocí funkce Tanh, který obsahuje všechny možné hodnoty z minulých výstupů a aktuálních vstupů. Nakonec jsou regulované hodnoty a vzniklý vektor vynásobeny pro získání užitečných informací.

Výstupní brána (Output Gate) slouží k extrahování užitečných informací, které mají být prezentovány jako výstup. Při výstupu jsou hodnoty odeslány jako vstup do další buňky. [15]

3.4.4 **Aktivační funkce**

Aktivační funkce určují výstup neuronové sítě. Stejná funkce je připojena ke každému jednotlivému neuronu ve vrstvě. Je tedy možné a žádoucí, aby různé vrstvy v síti měly různé aktivační funkce.

V neuronových sítích jsou data do vstupní vrstvy nejčastěji převáděna pomocí numerických tenzorů. Výstup neuronu je získán vynásobením vstupu s hodnotou vah jednotlivých neuronů. Tím je získán postsynaptický potenciál, ke kterému je přičtena prahová hodnota neuronu. Následně je na tento součet použita aktivační funkce. Výstup z aktivační funkce je poté použit jako vstup ve vrstvě následující. [16]

3.4.4.1 Lineární aktivační funkce

Lineární aktivační funkce má tvar

$$f(x) = c \cdot x$$

kde x je vstup vynásobený hodnotou vah a c je konstanta. Výstup je poté úměrný vstupu. Nevýhodou je, že při použití lineární aktivační funkce v jedné vrstvě se další vrstvy zhroutí, a vznikne tak pouze jedna vrstva.

3.4.4.2 Nelineární aktivační funkce

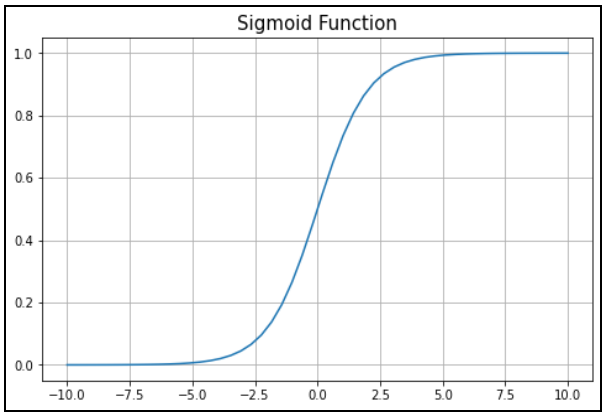
Moderní modely neuronových sítí používají nelineární aktivační funkce. Umožňují modelu vytvářet komplexní mapování mezi vstupy a výstupy sítě, které jsou nezbytné pro učení a modelování komplexních dat, jako jsou obrázky, video, audio a datové soubory.

Sigmoid

Funkce Sigmoid dokáže relativně dobře identifikovat malé změny na vstupu a ty poté projevit jako velké změny na výstupu. Tato vlastnost umožňuje funkci Sigmoid být dobrý binární klasifikátor. Nevýhodou této funkce je, že pokud jsou vstupní hodnoty dále od středu, na výstupu nejsou patrné. V této funkci se také vyskytuje problém mizejícího gradientu. Výstup je v rozmezí od 0 do 1, tvar funkce je zobrazen na obrázku 8. Rovnice funkce má tvar:

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

Obrázek 8 - Graf funkce Sigmoid



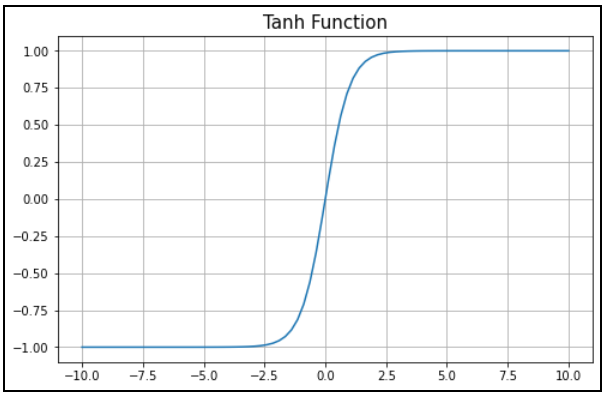
Zdroj: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

Tanh

Funkce Tanh je obdobná jako funkce Sigmoid s rozdílem, že výstup se pohybuje v intervalu od -1 do 1. Z obrázku je patrné, že Tanh konverguje rychleji než Sigmoid. Obdobně jako u Sigmoid se i v této funkci vyskytuje problém mizejícího gradientu. Tvar funkce je zobrazen na obrázku 9. Rovnice funkce má tvar:

$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

Obrázek 9 - Graf funkce Tanh



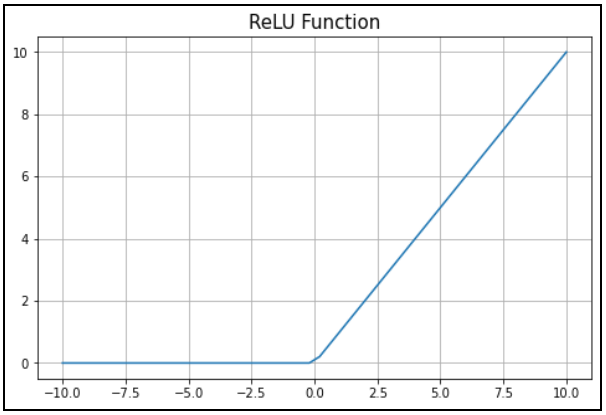
Zdroj: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

ReLu

ReLu je v dnešní době asi nejpoužívanější aktivační funkcí na světě. Výstup funkce je stejný jako vstup pro vstupní hodnoty větší než 0, pro jiné hodnoty je výstup roven 0. Tato vlastnost snižuje schopnost modelu správně se učit z dat. Tvar funkce je zobrazen na obrázku 10. Rovnice funkce má tvar:

$$f(x) = \max(0, x)$$

Obrázek 10 - Graf funkce ReLu



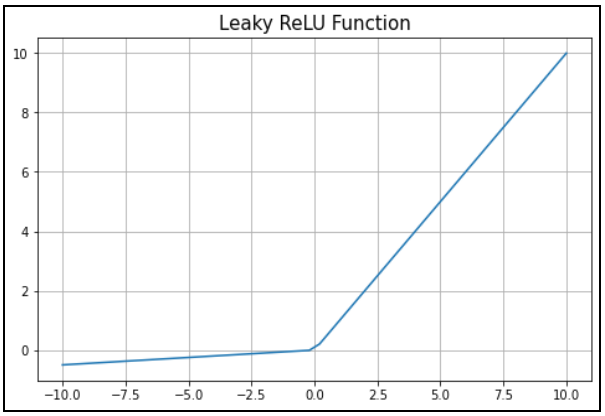
Zdroj: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

Leaky ReLu

Leaky ReLu je funkce podobná standardnímu ReLu s rozdílem, že záporné vstupy jsou vynásobeny zvolenou konstantou. Konstanta a má přednastavenou hodnotu 0,01, pokud má jinou hodnotu, jedná se o Randomized ReLu. Funkce je monotónní. Tvar funkce je zobrazen na obrázku 11. Rovnice funkce má tvar:

$$f(x) = \max(a \cdot x, x)$$

Obrázek 11 - Graf funkce Leaky ReLu



Zdroj: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

Softmax

Funkce Softmax je aktivační funkce, která se nejčastěji používá ve výstupní vrstvě úloh s klasifikací do více než dvou tříd. Výstupem poslední vrstvy s aktivační funkcí Softmax jsou hodnoty, které udávají pravděpodobnosti, se kterými lze určit třídu vstupu. Součet všech pravděpodobností pro jeden vstupní tenzor je roven 1. [17]

3.5 Kompilace modelu

Kompilace modelu slouží k nastavení numerických knihoven na back-end (mezi které patří např. TensorFlow nebo Theano) pomocí knihovny Keras. Tímto způsobem může knihovna Keras využívat vždy nejlepší knihovnu na výpočetní operace pro danou úlohu. Dále je při kompilaci predikováno, na jakém HW budou probíhat výpočty nejrychleji, jestli např. na CPU nebo na GPU. Pro správný průběh kompilace musí být vybrán typ ztrátové funkce, optimalizátor a metriky. [18]

3.5.1 **Ztrátová funkce**

Neuronové sítě využívající princip hlubokého učení jsou trénovány pomocí algoritmu optimalizace sestupu stochastického gradientu. Hlavním principem optimalizace je opakované odhadování chyb pro aktuální stav modelu. Proto je nutné vybrat ztrátovou funkci, kterou je možné využít k odhadnutí ztráty modelu a aktualizaci vah. Díky tomu je možné snížit ztrátu při příštím vyhodnocení.

Proto musí ztrátová funkce odpovídat řešenému problému jako je např. binární klasifikace nebo regrese. Pro typ úlohy binární klasifikace jsou vhodné tyto ztrátové funkce:

- Binary Cross-Entropy,
- Hinge,
- Squared Hinge. [19]

3.5.2 **Optimalizátor**

Cílem optimalizátoru je minimalizovat danou funkci, která je v tomto případě ztrátová funkce neuronové sítě. K tomu je nutné vypočítat sklon (gradient), který je vektorem, jehož složky jsou parciální derivace prvního řádu v daném bodě. Dalším krokem je posun parametrů v opačném směru od gradientu.

Hlavní myšlenkou při učení modelu je nechat algoritmem zpětného šíření zpracovat tréninkovou sadu dat, vypočítat gradient a aktualizovat parametry vrstev. Tím dojde ke snížení ztrátové funkce, a tedy zvyšování správnosti modelu.

3.5.2.1 Dávkový gradientní sestup

V dávkovém gradientním sestupu jsou všechna tréninková data zohledněna v jediné dávce. Gradient je vypočítán jako průměr všech gradientů z celé dávky dat. Parametry sítě jsou tedy aktualizovány pouze jednou po vypočítání průměrného gradientu. Tato metoda je vhodná pro data s relativně malými odchylkami.

3.5.2.2 Stochastický gradientní sestup

Stochastický gradientní sestup aktualizuje váhové parametry po vyhodnocení ztrátové funkce z každého vzorku dat. Z toho důvodu, že je gradient vypočítán z každého vzorku dat, dochází ke kolísání kolem minima ztrátové funkce. [11, s. 58]

3.5.2.3 Minidávkový stochastický gradientní sestup

Minidávkový gradientní sestup je kompromis mezi stochastickým gradientním sestupem a dávkovým gradientním sestupem. V minidávkovém gradientním sestupu jsou po průchodu minidávky neuronovou sítí vypočítány gradienty pro všechny vzorky z dávky.

Následně jsou tyto gradienty zprůměrovány a na základě výsledného gradientu jsou upraveny parametry vrstev. Tento cyklus pokračuje, dokud nejsou vyčerpány všechny minidávky, ze kterých se skládají trénovací data. [11, s. 59]

3.5.3 **Metriky**

Zatímco ztrátová funkce slouží k optimalizaci modelu při trénování, metriky se používají pro sledování a měření výkonu modelu jak při trénování, tak při testování.

Metrika	**Vzorec**
Accuracy	$\frac{T_p + T_n}{T_p + T_n + F_n + F_p}$
Precision	$\frac{T_p}{T_p + F_p}$
Recall - (TPR)	$\frac{T_p}{T_p + F_n}$

False alarm - (FPR)	$\frac{F_p}{F_p + T_n}$
Specificity	$\frac{T_n}{T_n + F_p}$
F1 Score	$2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$

Tabulka 1 - Vybrané metriky

Základní metriky zobrazené v tabulce 1 lze vypočítat z matice záměn, která je uvedena v tabulce 2. Matice slouží k vizualizaci vztahu mezi správně a špatně klasifikovanými třídami.

		Skutečná třída	
		A	B
Predikovaná třída	A	Tp	Fp
	B	Fn	Tn

Tabulka 2 - Matice záměn.

Tp – Hodnota true positive označuje počet správně označených predikcí z třídy A.

Fn – Hodnota false negative označuje počet špatně označených predikcí z třídy A.

Fp – Hodnota false positive označuje počet špatně označených predikcí ze třídy B.

Tn – Hodnota true negative označuje počet správně označených predikcí z třídy B.

Rozdělení do tříd je uskutečněno na základě hodnoty vypočítané pravděpodobnosti, která je získána z poslední neuronové vrstvy.

Pokud se jedná o klasifikaci do více tříd, tak je příklad klasifikován podle maximální hodnoty ze získaných pravděpodobností.

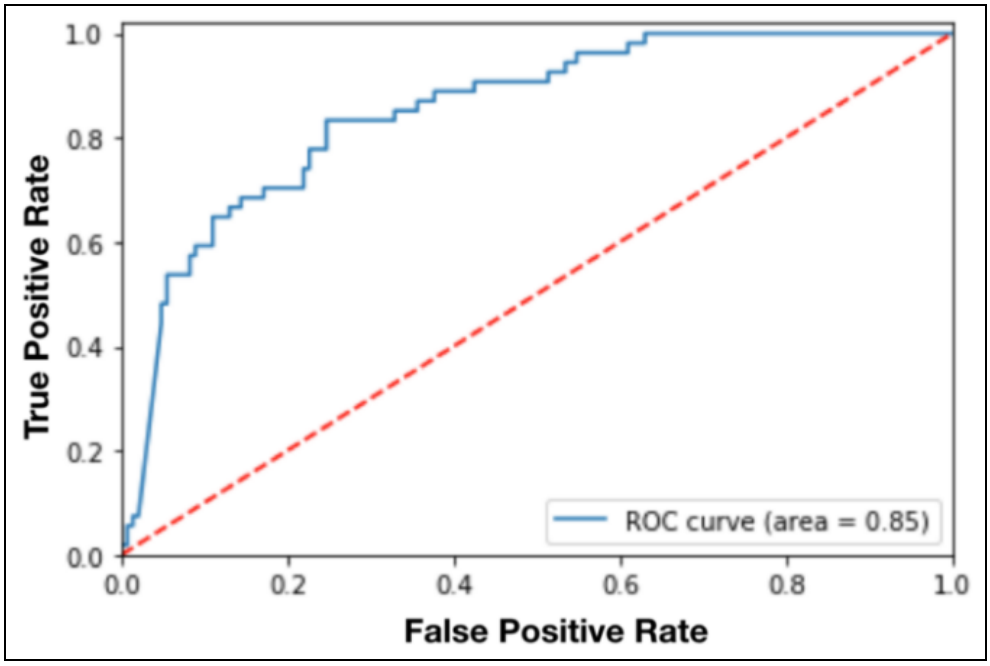
Pokud se jedná o úlohy binární klasifikace, tak pravděpodobnost, s jakou daný příklad patří do zkoumané třídy, je porovnána s přednastavenou prahovou hodnotou a na základě výsledku je příklad klasifikován. [20]

3.5.3.1 Křivka ROC

K určení účinnosti modelů s binárním klasifikátorem je možné použít křivku ROC (Receiver Operating Characteristic), která ukazuje poměr TPR (True Positive Rate) proti FPR (False Positive Rate) při nastavení různých mezních prahových hodnot.

Plocha pod křivkou ROC se označuje AUC (Area Under Curve) a dá se interpretovat jako pravděpodobnost, že model ohodnotí náhodný příklad spadající pod pozitivní třídu lépe než náhodný příklad spadající pod negativní třídu. Hodnota AUC se pohybuje v rozmezí 0 a 1. [21]

Obrázek 12 - Křivka ROC



Zdroj: https://towardsdatascience.com/20-popular-machine-learning-metrics-part-1-classification-regression-evaluation-metrics-1ca3e282a2ce

3.6 Metody předzpracování textu

Písmo je jedním z nerozšířenějších komunikačních nástrojů mezi lidmi. Obecně se dá říct, že jednotlivé znaky písma mohou sloužit k uchovávání jakéhokoliv mluveného záznamu, samozřejmě při dodržení gramatik nějakého jazyka. Pokud jsou jednotlivé znaky písma uskupeny do slov a slova do vět, vzniká spojitý útvar zvaný text.

Bohužel algoritmy strojového učení, které využívají neuronové vrstvy popsané v kapitole *3.4.3 Vrstvy neuronové sítě*, nedokážou porozumět textu jako takovému. Text pro tyto algoritmy musí být upraven do přijatelnější podoby, jako jsou například numerické tenzory.

3.6.1 Kódování 1-z-n

Kódování slov a znaků kódem 1-z-n, nebo také algoritmus one hot word, je jedním z nejzákladnějších způsobů jak, vektorizovat slova nebo věty. Pro převedení věty na vektor je nutný slovník s očíslovaným pořadím slov, kde se každé slovo může vyskytovat pouze jednou.

Výsledkem je binární vektor o velikosti N, kde N je velikost slovníku. Každé slovo ve větě je pak ve vektoru reprezentováno číslem 1 na místě určeném číslem pořadí ze slovníku. [22]

3.6.2 Shlukování slov

Dalším způsobem, jak nahradit text pomocí numerických tenzorů, je využití hustých slovních vektorů vznikajících při použití metody shlukování slov (word embedding). Zatímco metoda 1-z-n využívá velmi rozměrné binární vektory, metoda shlukování slov si vystačí s reálnými vektory s nízkou dimenzí. Existují dva způsoby, jak model naučit shlukovaná slova:

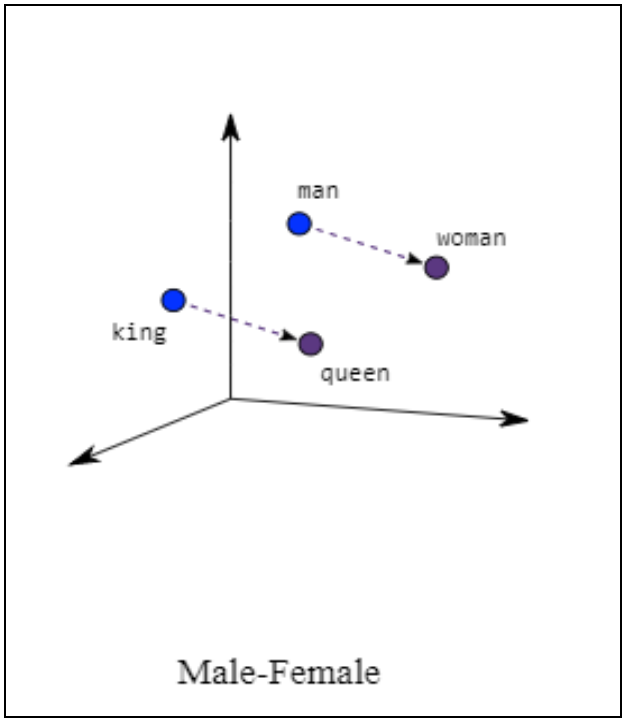
- Pomocí speciální vrstvy při trénování modelu. Vznikají tak nová spojení slov, která jsou postupně vydefinována pro danou úlohu.
- Pomocí již předtrénované vrstvy, která se do modelu vloží. Tato metoda se používá v případě, kdy není dostatek trénovacích dat.

Speciální vrstva pro vytváření slovních spojení se jmenuje Embedding a lze ji chápat jako slovník, který transformuje vstupní vektor s přirozenými čísly do hustých vektorů.

Vrstva Embedding je nakonfigurována několika vstupními parametry, mezi které patří velikost vstupní dimenze, dimenze výstupního vektoru a délka vstupní sekvence. Velikost vstupní dimenze určuje počet slov ve slovníku, na které se vstupní vektory mapují. Velikost výstupního vektoru udává velikost vektoru pro další vrstvu.

Délka vstupní sekvence určuje z kolika slov se může vstupní text skládat. Vstupní vektor musí mít přednastavenou vstupní délku sekvence, přesto je možné ho z jedné strany vyplnit číslem 0, pokud je text kratší. [23]

Obrázek 13 - Prostor shlukovaných slov



Zdroj: https://developers.google.com/machine-learning/

Husté vektory popisují vztah mezi jednotlivými slovy v prostoru. Z obrázku 13 je patrné, že vektor mezi slovy man a woman popisuje skoro stejný vztah jako vektor mezi slovy king a queen.

3.7 Použité technologie

V této kapitole jsou popsány technologie a knihovny použité při řešení praktické části diplomové práce.

3.7.1 TensorFlow

TensorFlow je open source knihovna od společnosti Google poskytující komplexní a flexibilní sadu nástrojů, které umožňují výzkumným pracovníkům a vývojářům využívat nejmodernější technologie v oblasti strojového učení.

Knihovna TensorFlow umožnuje práci s daty v grafech. Každý uzel v grafu reprezentuje matematickou operaci a každá hrana je multidimenzionální datové pole nebo tenzor. Uzly a hrany jsou reprezentovány jako objekty za pomoci programovacího jazyka Python, který zastřešuje celkovou abstrakci a možnost pracovat ve vysokoúrovňovém jazyce.

Matematické operace, které zaštiťují například práci s tenzory, jsou pak psány v jazyce C ++. Aplikace TensorFlow lze spustit na CPU i na GPU, stejně tak je možné aplikace spouštět i na speciálních cloudech, které poskytují výrazně lepší výpočetní výkon. [24]

3.7.2 Keras

Keras je jedním z nejpoužívanějších API pro vytváření modelů neuronových sítí na světě. Mezi jeho hlavní přednosti patří podpora několika výpočetních engine na back-end, mezi které patří např. TensorFlow nebo Thaneo.

Hlavní důraz byl kladen na uživatelskou přívětivost, modulárnost a rozšiřitelnost. Mezi hlavní přednosti knihovny Keras patří bohatě vybavená knihovna, které obsahuje neuronové vrstvy, ztrátové funkce, optimalizátory, metriky, aktivační funkce a mnohé další. [25]

3.7.3 **Výpočetní zařízení**

Výpočty probíhaly na zařízení uvedeném v tabulce 3.

	Notebook
OS	Windows 10
CPU	Intel Core i5-8265U
GPU	Integrovaná
RAM	8 GB
Python	3.7.7
TensorFlow	2.2.0
Keras	2.4.3

Tabulka 3 - Výpočetní zařízení

4 Vlastní práce

Tato část diplomové práce obsahuje popis řešení úlohy klasifikace takzvaných Fake news. Pojem Fake news označuje nepravdivé zprávy, kterými se šíří dezinformace za účelem ovlivnit či zmanipulovat příjemce informace. [26]

Cílem této práce bylo vytvořit model, který dokáže rozpoznat, zda se jedná o nepravdivou zprávu na základě informací poskytnutých ve zprávě. Model využívá metody hlubokého učení, popsané v kapitole *3. Teoretická východiska.*

Jak ale sestavit nejlepší model pro danou úlohu? Chollet se v literatuře velmi často zmiňuje, že přesně definovaný postup neexistuje. Existují pouze obecné poučky a principy, pomocí kterých se dá nalézt základní model, který by měl porazit referenční hodnotu stanovenou pro danou úlohu náhodným klasifikátorem. [11, s. 115]

Pro úlohy binární klasifikace, pod které spadá i tento případ, je referenční hodnota 0,5, tedy základní modely by měly mít správnost větší než 50 %. Z toho důvodu byly pro tuto úlohu vytvořeny čtyři rozdílné modely. Každý model pracoval s jiným typem skrytých vrstev. Pro jejich vyhodnocení sloužila jejich úspěšnost při klasifikaci, doba zpracování jednotlivých epoch i počet epoch nutných k dosažení přeučení. [11, s. 86]

4.1 Vstupní data

Vstupní data byla získána od společnosti Kaggle, komunity datových vědců a odborníků na strojové učení. Společnost Kaggle se dostala do povědomí od roku 2010, kdy začala pořádat soutěže v oblasti strojového učení a v této aktivitě pokračuje dodnes. Mezi její další služby patří poskytování datových množin a výuková platforma strojového učení.

4.1.1 Fake news Dataset

Datová sada Fake news se skládá z článků uložených ve třech souborech *train.csv, test.csv* a *submit.csv*, které jsou rozděleny do tabulky se sloupci „*id*“, „*title*“, „*author*“, „*text*“ a „*label*“. Label je určen binární klasifikací – 0 pro pravdivý článek (Non-Fake news) a 1 pro nepravdivý článek (Fake news). Ukázka datové sady je zobrazena na obrázku 14.

Soubor *train.csv* obsahuje 20 800 ohodnocených článků, soubor *test.csv* obsahuje 5 200 článků a soubor *submit.csv* obsahuje ohodnocení k souboru *test.csv*.

Články byly seskupeny, promíchány a následně rozděleny do tří skupin:

- trénovací data sloužící pro natrénování modelu,
- validační data sloužící pro ověření natrénovaného modelu,
- testovací data sloužící pro finální otestování modelu.

Testovací data by na modelu měla být použita pouze jednou pro finální otestování, aby nedošlo k prosáknutí informací do modelu. Pro ověření správnosti modelu se používají validační data.

Obrázek 14 - Ukázka trénovacích dat

id	title	author	text	label
0.00 - 1039.95 Count: 1,040 0 20.8k	[null] 3% The Dark Agenda B... 0% Other (20237) 97%	nan 9% Pam Key 1% Other (18600) 89%	20387 unique values	0 1
0	House Dem Aide: We Didn't Even See Comey's Letter Until Jason Chaffetz Tweeted It	Darrell Lucus	House Dem Aide: We Didn't Even See Comey's Letter Until Jason Chaffetz Tweeted It By Darrell Lucus o...	1
1	FLYNN: Hillary Clinton, Big Woman on Campus - Breitbart	Daniel J. Flynn	Ever get the feeling your life circles the roundabout rather than heads in a straight line toward th...	0
2	Why the Truth Might Get You Fired	Consortiumnews.com	Why the Truth Might Get You Fired October 29, 2016 The tension between intelligence analysts and po...	1
3	15 Civilians Killed In Single US Airstrike Have Been Identified	Jessica Purkiss	Videos 15 Civilians Killed In Single US Airstrike Have Been Identified The rate at which civilians a...	1
4	Iranian woman jailed for fictional unpublished story about woman stoned to death for adultery	Howard Portnoy	Print An Iranian woman has been sentenced to six years in prison after Iran's Revolutionary Guard s...	1

4.1.2 **Příprava dat**

K rozhodnutí, zda se jedná o pravdivý, nebo nepravdivý článek, byl použit nadpis článku získaný ze sloupce „*title*“. Maximální délka nadpisu je 37 slov a průměrná délka nadpisu je 12 slov. Bohužel algoritmus strojového učení nedokáže porozumět textu jako takovému, a proto musel být vstup upraven do podoby pro algoritmus čitelnější.

Z nadpisu byly odebrány všechny ostatní znaky kromě písmen a všechna velká písmena byla zmenšena. Tento postup byl opakován pro všechny nadpisy celá datové sady. Tím bylo dosaženo standardizování textu do jednotné formy. Dále bylo zjištěno, že některé záznamy v datech neobsahovaly nadpisy, proto byly z dat odebrány.

4.2 **Model Alfa**

Model Alfa byl sestaven z vrstev Dense, tedy z hustě propojených neuronových sítí. Pro tento typ modelu bylo nutné upravit vstupní data na vektory nul a jedniček pomocí metody 1-z-n, která je popsána v kapitole *3.6.1*. Jako aktivační funkce pro horní mezivrstvy byla zvolena funkce ReLu a pro poslední vrstvu funkce Sigmoid.

Pro zkompilování modelu byl vybrán optimalizátor RMSprop. Jako ztrátová funkce byla vybrána binary_crossentropy. A jako metrika byla zvolena správnost modelu accuracy.

Pro trénování modelu bylo zvoleno 10 epoch o velikosti dávky 128. Poměr validačních dat ku trénovacím datům byl 0,2.

Obecnou praktikou vytváření modelu je vytvoření modelu s nízkým počtem vrstev a skrytých jednotek. Natrénováním modelu dojde k podučení modelu. Následně se metodou „pokus omyl“ přidávají skryté vrstvy a jednotky a podle validačních výsledků se určuje, který model je lepší.

4.2.1 **Trénování 01**

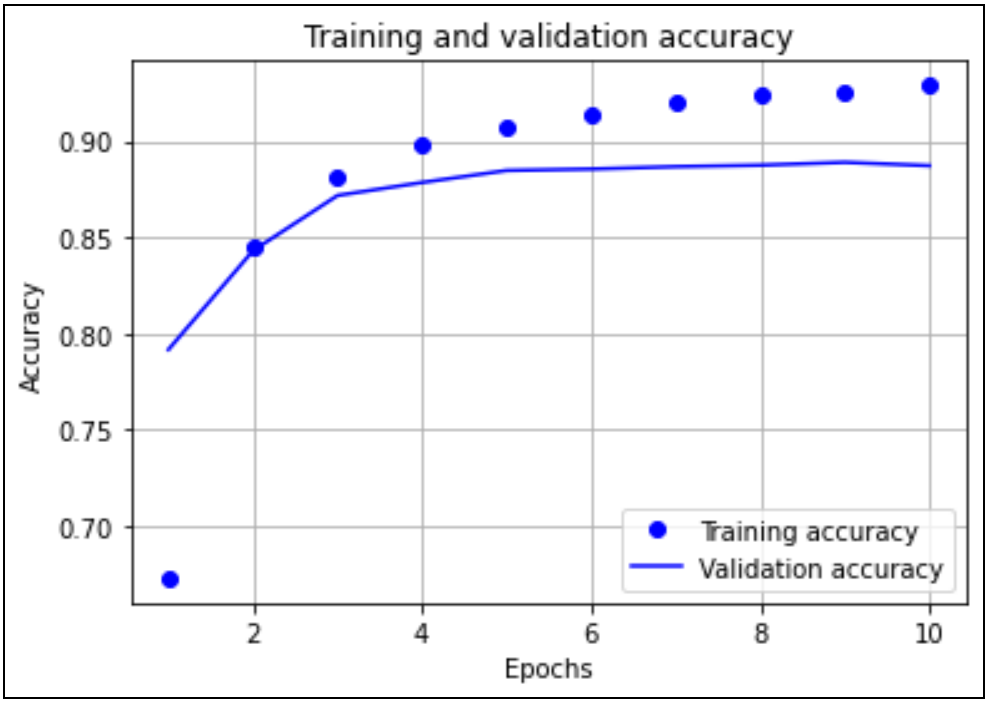
Jak je vidět na obrázku 15, první pokus modelu Alfa se skládal ze dvou vrstev Dense, první vrstva měla čtyři skryté jednotky, které určovaly, jak složité vzory se model dokázal naučit. Poslední vrstva měla pouze jednu skrytou jednotku, pomocí které vytvářel skalární predikci týkající se spolehlivosti nadpisu. Celkový počet skrytých parametrů v celé neuronové síti byl 20 009.

Obrázek 15 - Alfa Summary 01

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 4)                 20004
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 5
=================================================================
Total params: 20,009
Trainable params: 20,009
Non-trainable params: 0
```

Na obrázku 16 je zobrazena trénovací a validační správnost modelu. Tečkovaná křivka zobrazuje trénovací správnost. Plná křivka zobrazuje validační správnost. Rozdíl mezi křivkami značí zvyknutí si modelu na trénovací data, tedy přeučení. I když je model přeučen, validační správnost stále roste. Model má tedy z trénovacích dat stále užitek a může se naučit více nezávislých vzorů. Proto je možné přidat více skrytých parametrů. Validační správnost dosahuje hodnoty 0,8890 při 10. epoše. A to je více než referenční hodnota 0,5 dosažená náhodným klasifikátorem.

Obrázek 16 - Alfa Accuracy 01



4.2.2 **Trénování 02**

Ve druhém pokusu byl navýšen počet skrytých parametrů v první vrstvě na osm, jak je vidět na obrázku 17. Tím se zvedl celkový počet parametrů na 40 017.

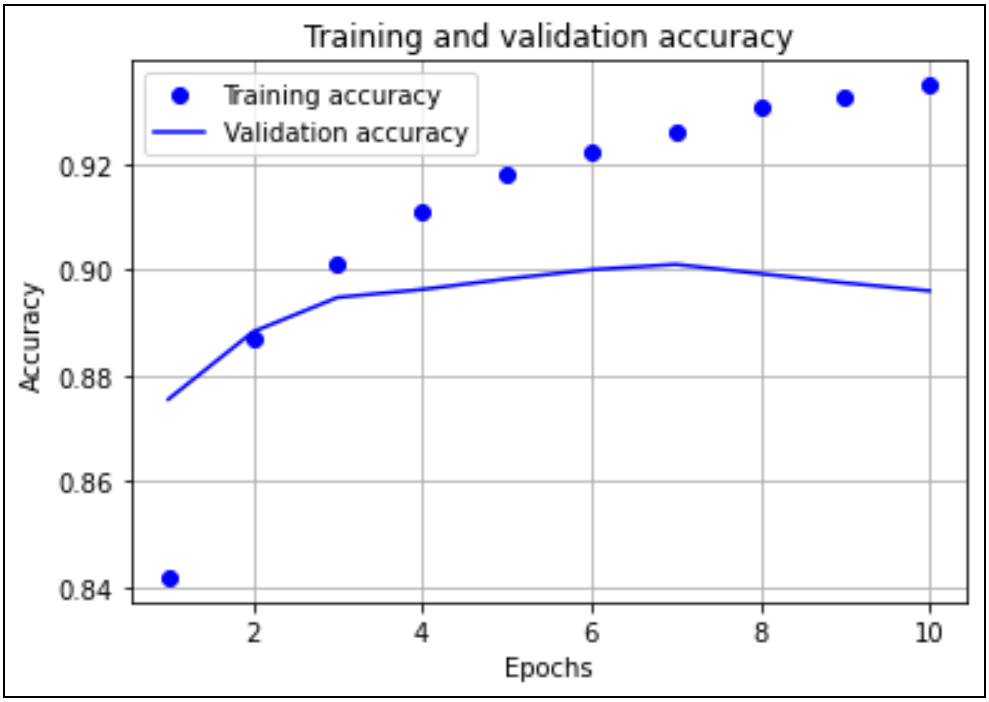
Obrázek 17 - Alfa Summary 02

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 8)                 40008
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 9
=================================================================
Total params: 40,017
Trainable params: 40,017
Non-trainable params: 0
```

Na obrázku 18 je vidět, že se validační správnost modelu od 7. epochy nelepší.

Byla dosažena hranice toho, co se z trénovacích dat model mohl naučit. Model může být vylepšen přidáním skryté vrstvy pro zvýšení rychlosti učení při zachování počtu skrytých parametrů. Validační správnost dosáhla hodnoty 0,9010 při 7. epoše.

Obrázek 18 - Alfa Accuracy 02



4.2.3 **Trénování 03**

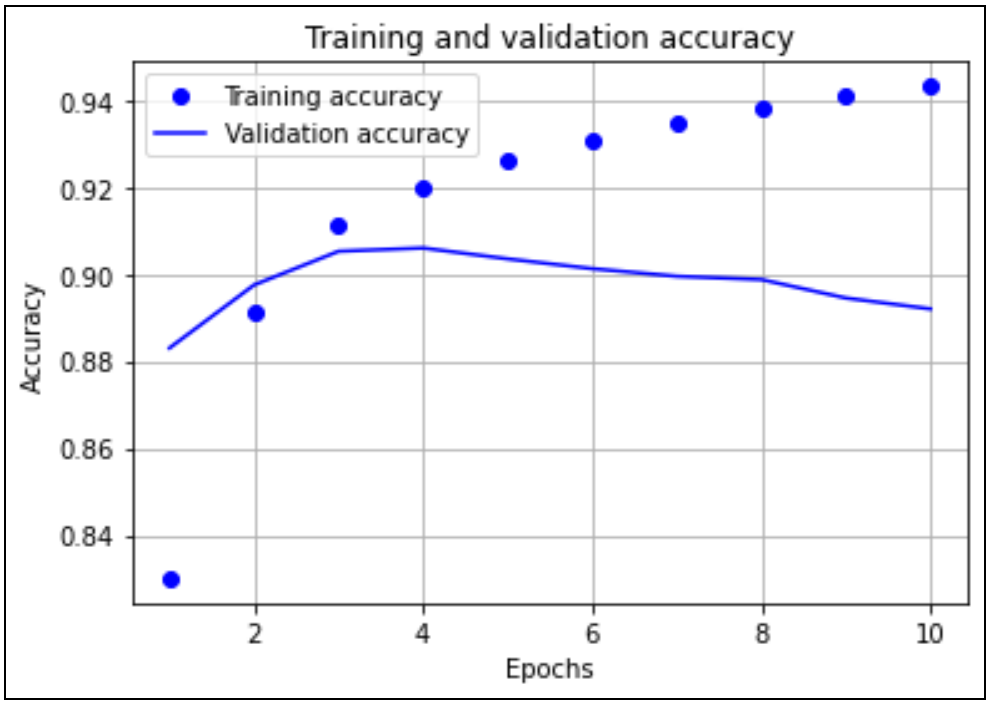
Ve třetím pokusu byla přidána skrytá mezivrstva s osmi skrytými jednotkami, jak je vidět na obrázku 19. Tím vznikl model s celkovým počtem 40 089 parametrů.

Obrázek 19 - Alfa Summary 03

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 8)                 40008
_________________________________________________________________
dense_5 (Dense)              (None, 8)                 72
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 9
=================================================================
Total params: 40,089
Trainable params: 40,089
Non-trainable params: 0
```

Na obrázku 20 je vidět, že se validační správnost modelu od 4. epochy nezlepšila. Bylo dosaženo velmi podobné hranice jako v předešlém trénování, jen o několik epoch dříve. Validační správnost dosáhla hodnoty 0,9060 při 4. epoše.

Obrázek 20 - Alfa Accuracy 03



Výsledný model Alfa byl vybrán z trénování 03. Validační správnost modelu byla 90,6 %.

4.3 Model Beta

Model Beta byl, obdobně jako model Alfa, sestaven z hustě propojených vrstev Dense. Tento model používal jako vstup husté slovní vektory popsané v kapitole *3.6.2*.

Pro trénování modelu bylo k dispozici 20 000 textů, z toho důvodu byla zvolena metoda naučení se shlukování slov společně s hlavní úlohou pomocí vrstvy Embedding.

Jako aktivační funkce pro horní mezivrstvy byla zvolena funkce ReLu a pro poslední vrstvu funkce Sigmoid.

Pro zkompilování modelu byl vybrán optimalizátor RMSprop. Jako ztrátová funkce byla vybrána binary_crossentropy. A jako metrika byla zvolena správnost modelu accuracy.

Pro trénování modelu bylo zvoleno 10 epoch o velikosti dávky 128. Poměr validačních dat ku trénovacím datům byl 0,2.

Znovu bylo postupováno pomocí obecné praktiky vytvoření modelu s nízkým počtem vrstev a skrytých jednotek a následným přidáváním skrytých vrstev a jednotek podle validačního skóre.

4.3.1 Trénování 01

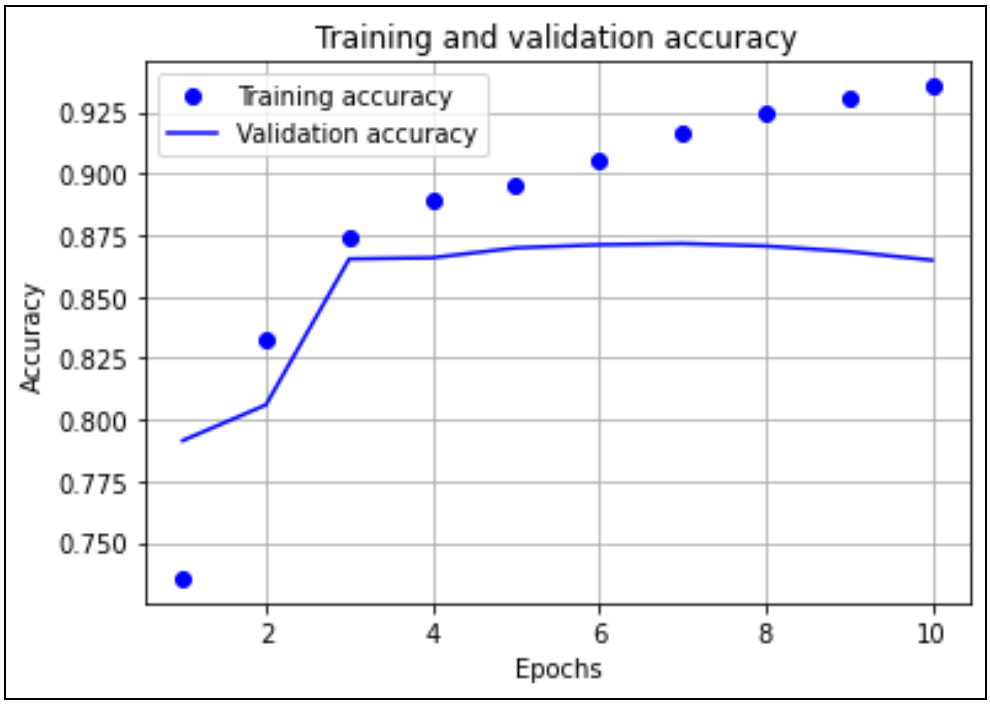
Jak je vidět na obrázku 21, první pokus modelu Beta se skládal z vrstvy Embedding, která měla za úkol identifikování vztahů mezi jednotlivými slovy. Následovala vrstva Flatten, která zajišťovala zploštění výstupního vektoru o jednu dimenzi tak, aby na vrstvu Embedding navazovala vrstva Dense se čtyřmi skrytými jednotkami. Poslední vrstva Dense měla opět jednu skrytou jednotku. Celkový počet skrytých parametrů byl 20 649.

Obrázek 21 - Beta Summary 01

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 40, 4)             20000
_________________________________________________________________
flatten_1 (Flatten)          (None, 160)               0
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 644
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 5
=================================================================
Total params: 20,649
Trainable params: 20,649
Non-trainable params: 0
```

Na obrázku 22 je vidět, že validační křivka má rostoucí tendenci až do 7. epochy, kdy validační správnost začne klesat. Přidáním dalších skrytých parametrů bude zjištěno, zda jedná opravdu o hranici přeučení. Validační správnost dosahovala hodnoty 0,8715 při 7. epoše.

Obrázek 22 - Beta Accuracy 01



4.3.2 **Trénování 02**

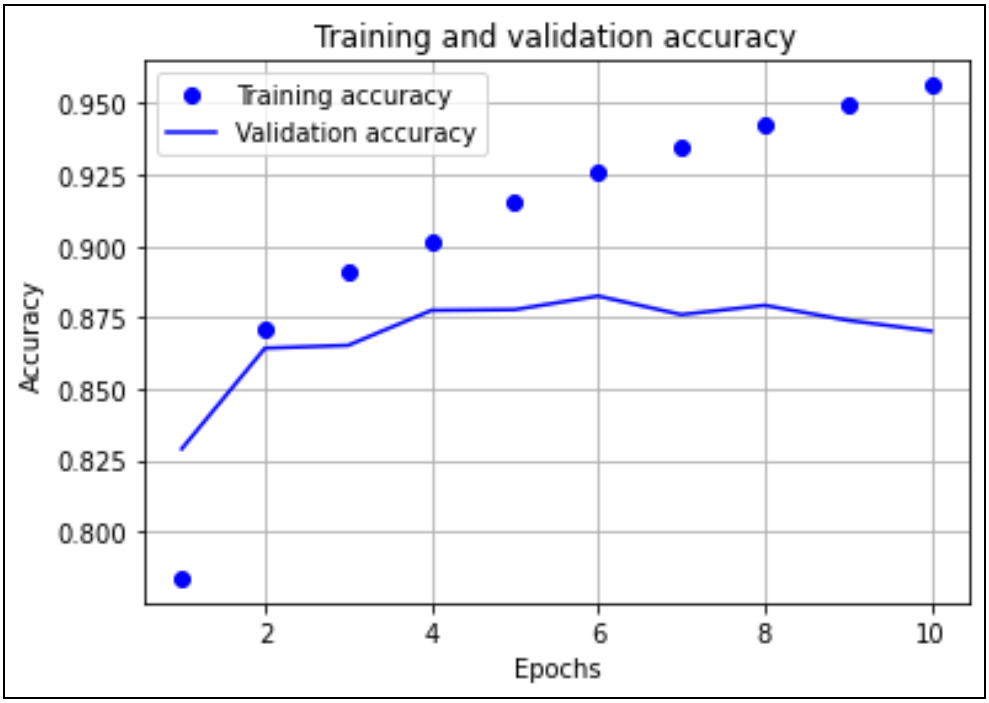
Ve druhém pokusu byl navýšen počet skrytých parametrů v první Dense vrstvě na osm, jak je vidět na obrázku 23. Tím se zvedl celkový počet parametrů na 42 577.

Obrázek 23 - Beta Summary 02

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 40, 8)             40000
_________________________________________________________________
flatten_2 (Flatten)          (None, 320)               0
_________________________________________________________________
dense_4 (Dense)              (None, 8)                 2568
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 9
=================================================================
Total params: 42,577
Trainable params: 42,577
Non-trainable params: 0
```

Na obrázku 24 je vidět, že se validační správnost modelu od 6. epochy nezlepšila. Bylo dosaženo obdobné hranice jako v minulém trénování, ale o jednu epochu dříve. Validační správnost nabyla hodnoty 0,8825 při 6. epoše.

Obrázek 24 - Beta Accuracy 02



Výsledný model Beta byl vybrán z trénování 02. Validační správnost modelu byla 88,3 %.

4.4 Model Gama

Model Gama byl postaven na konvolučních vrstvách. V kapitole *3.4.3.2 Konvoluční vrstvy* je uvedeno, že se tyto vrstvy velmi často používají pro obrazové vstupy. Přesto je možné při výběru správné dimenze vrstvy zpracovat sekvenční data, mezi které patří mimo jiné i text. Princip je velmi podobný jako u obrazového vstupu, jen se místo 2D oblastí z obrazových tenzorů vybere lokální 1D oblast části textu.

Pro tento typ modelu bylo nutné upravit vstupní data do podoby hustých slovních vektorů pomocí shlukování slov na základě popisu v kapitole *3.6.2*. Pro trénování modelu bylo k dispozici 20 000 textů, z toho důvodu byla zvolena metoda naučení se shlukování slov společně s hlavní úlohou pomocí vrstvy Embedding.

Pro zkompilování modelu byl vybrán optimalizátor RMSprop. Jako ztrátová funkce byla vybrána binary_crossentropy. A jako metrika byla zvolena správnost modelu accuracy.

Pro trénování modelu bylo zvoleno 15 epoch o velikosti dávky 128. Poměr validačních dat ku trénovacím datům je 0,2.

Znovu bylo postupováno pomocí obecné praktiky vytvoření modelu s nízkým počtem vrstev a skrytých jednotek a následným přidáváním skrytých vrstev a jednotek podle validačního skóre.

4.4.1 Trénování 01

Jak je vidět na obrázku 25, první pokus modelu Gama se skládal z jedné vrstvy Embedding, dvou jedno-dimenzionálních konvolučních vrstev Conv1D, dvou vrstev MaxPooling, s tím, že jedna byla globální, a poslední vrstvy Dense s jednou skrytou jednotkou.

V modelu Gama byly pro vrstvu Embedding zvoleny jako vstupní parametry:

- vstupní dimenze: 5000,
- výstupní vektor: 32,
- délka vstupní sekvence: 40.

Po vrstvě Embedding následovala první konvoluční vrstva Conv1D se 32 skrytými jednotkami a konvolučním oknem o velikosti 9 slov, které určuje, jak velkou subsekvenci se síť dokáže naučit.

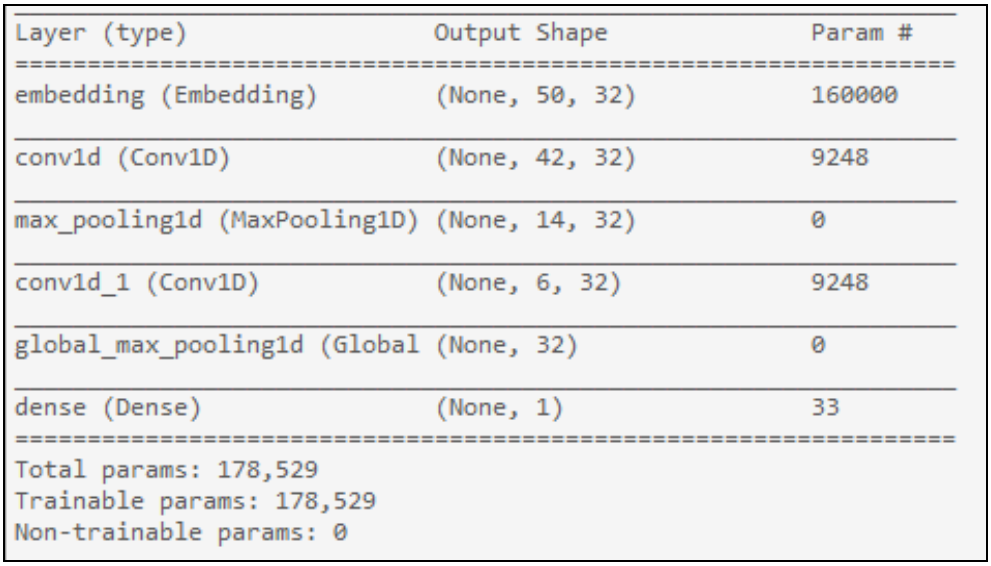
Následující skrytá vrstva MaxPooling1D zajišťovala sdružování podle maxima s nastaveným krokem 3.

Po sdružovací vrstvě opět následovala konvoluční vrstva Conv1D se 32 skrytými jednotkami a konvolučním oknem o velikosti 9 slov.

Předposlední vrstva GlobalMaxPoolig1D opět zajišťovala sdružování podle maxima, dále se starala i o zploštění výstupního vektoru tak, aby ho mohla poslední vrstva Dense zpracovat.

Celkový počet skrytých parametrů v celé neuronové síti byl 178 529.

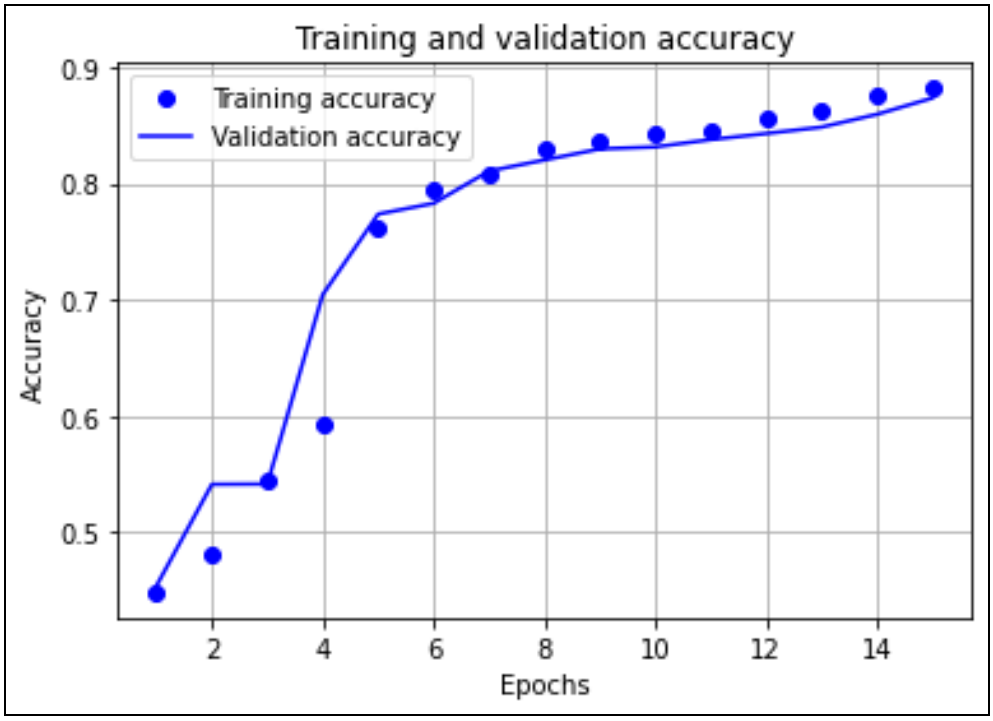
Obrázek 25 - Gama Summary 01



Na obrázku 26 je zobrazena trénovací a validační správnost modelu. Tečkovaná křivka opět zobrazuje trénovací správnost a plná křivka validační správnost.

Křivka validační správnosti je po 15. epoše stále rostoucí, model má tedy z trénovacích dat stále užitek a může se naučit více nezávislých vzorů. Proto je možné přidat více skrytých parametrů. Validační správnost dosahovala hodnoty 0,8733 při 15. epoše, a to je znovu více než referenční hodnota 0,5 dosažená náhodným klasifikátorem.

Obrázek 26 - Gama Accuracy 01



4.4.2 **Trénování 02**

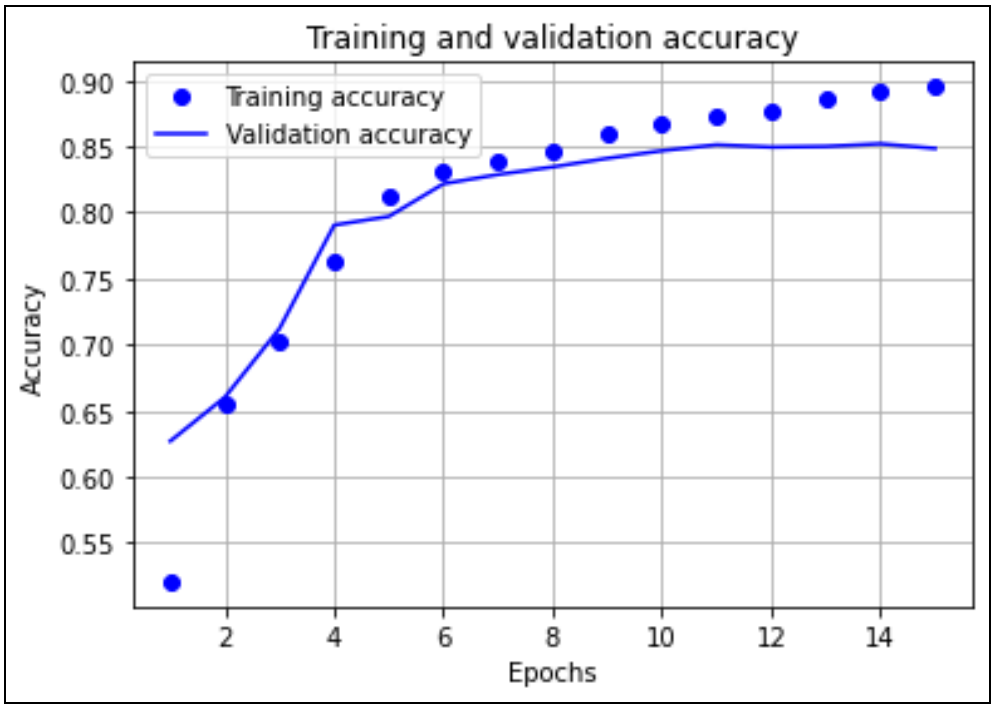
Ve druhém pokusu byl navýšen počet skrytých parametrů v konvolučních vrstvách na 64, jak je vidět na obrázku 27. Tím se zvedl celkový počet parametrů na 393 921.

Obrázek 27 - Gama Summary 02

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 40, 64)            320000
_________________________________________________________________
conv1d (Conv1D)              (None, 32, 64)            36928
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 10, 64)            0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 2, 64)             36928
_________________________________________________________________
global_max_pooling1d (Global (None, 64)                0
_________________________________________________________________
dense_10 (Dense)             (None, 1)                 65
=================================================================
Total params: 393,921
Trainable params: 393,921
Non-trainable params: 0
```

Na obrázku 28 je vidět, že se validační správnost modelu od 11. epochy nelepší. Bylo dosaženo hranice toho, co se z trénovacích dat model může naučit s tímto nastavením sítě. Validační správnost dosahovala hodnoty 0,8512 při 11. epoše.

Obrázek 28 - Gama Accuracy 02



4.4.3 **Trénování 03**

Ve třetím pokusu byl snížen krok skryté vrstvy MaxPooling1D na hodnotu 2, jak je vidět na obrázku 29. Tím model zůstal s celkovým počtem 393 921 parametrů.

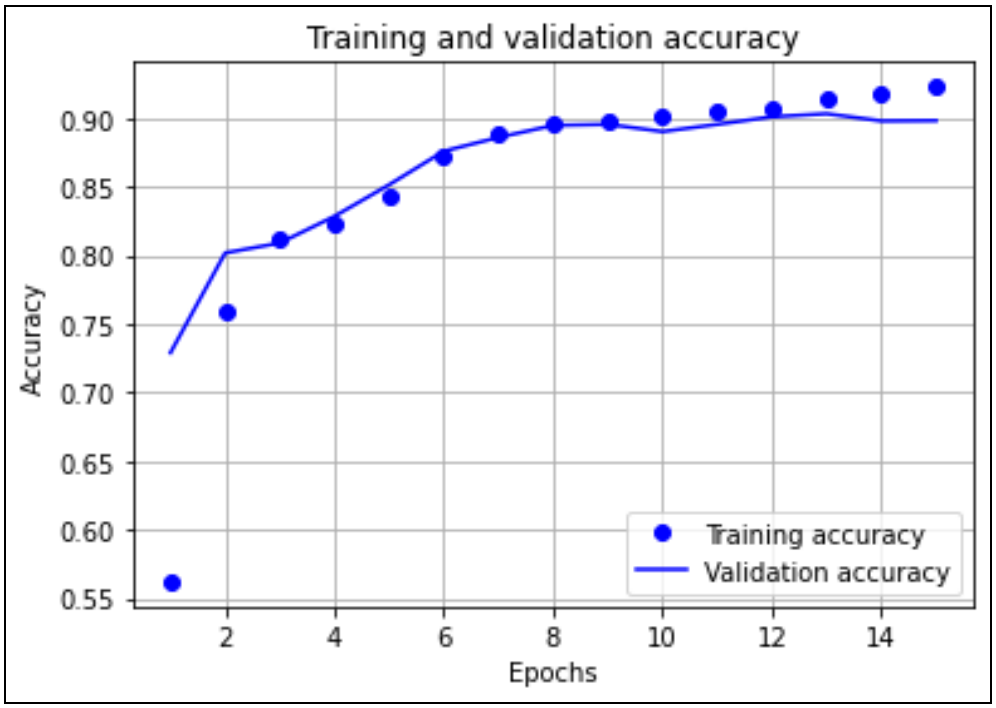
Obrázek 29 - Gama Summary 03

Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	(None, 40, 64)	320000
conv1d_2 (Conv1D)	(None, 32, 64)	36928
max_pooling1d_1 (MaxPooling1	(None, 16, 64)	0
conv1d_3 (Conv1D)	(None, 8, 64)	36928
global_max_pooling1d_1 (Glob	(None, 64)	0
dense_11 (Dense)	(None, 1)	65

Total params: 393,921
Trainable params: 393,921
Non-trainable params: 0

Na obrázku 30 je vidět, že se validační správnost modelu do 6. epochy zvýšila rychleji než při trénování 02. Poté se růst zpomalil, i tak model dosáhl správnosti 0,9032 při 13. epoše a předčil předchozí trénování.

Obrázek 30 - Gama Accuracy 03



Výsledný model Gama byl vybrán z trénování 03. Validační správnost modelu byla 90,3 %.

4.5 Model Delta

Základním kamenem modelu Delta se stala vrstva LSTM, která spadá pod rekurentní neuronové sítě popsané v kapitole *3.4.3.3*.

Zpětná vazba jednotlivých neuronů ve vrstvě LSTM umožňuje modelu postupně navazovat spojení na dřívější vstupy v jedné sekvenci, a zpracovávat tak sekvenční data odlišným způsobem od dříve použitých vrstev.

Opět bylo nutné pro tento typ modelu upravit vstupní data do podoby hustých slovních vektorů pomocí shlukování slov na základě popisu v kapitole *3.6.2*. Pro trénování modelu bylo k dispozici 20 000 textů, z toho důvodu byla zvolena metoda naučení se shlukování slov společně s hlavní úlohou pomocí vrstvy Embedding.

Pro zkompilování modelu byl vybrán optimalizátor RMSprop. Jako ztrátová funkce byla vybrána binary_crossentropy. A jako metrika byla zvolena správnost modelu accuracy.

Pro trénování modelu bylo zvoleno 10 epoch o velikosti dávky 128. Poměr validačních dat ku trénovacím datům je 0,2.

Znovu bylo postupováno pomocí obecné praktiky vytvoření modelu s nízkým počtem vrstev a skrytých jednotek a následným přidáváním skrytých vrstev a jednotek podle validačního skóre.

4.5.1 **Trénování 01**

Jak je vidět na obrázku 31, první pokus modelu Delta se skládal z jedné vrstvy Embedding, navazující vrstvy LSTM a poslední vrstvy Dense s jednou skrytou jednotkou.

V modelu Delta byly pro vrstvu Embedding zvoleny jako vstupní parametry:

- vstupní dimenze: 5000,
- výstupní vektor: 4,
- délka vstupní sekvence: 40.

Po vrstvě Embedding následovala LSTM vrstva se čtyřmi skrytými jednotkami. Na vrstvu LSTM navazovala vrstva Dense s jednou skrytou jednotkou.

Celkový počet skrytých parametrů v celé neuronové síti byl 20 149.

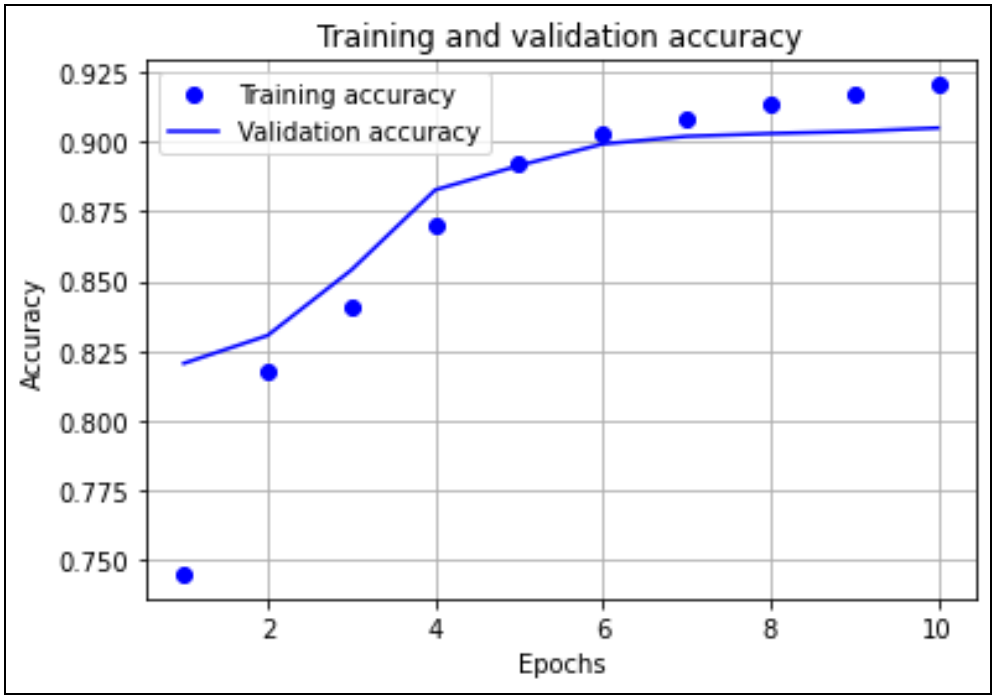
Obrázek 31 - Delta Summary 01

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_9 (Embedding)      (None, 40, 4)             20000
_________________________________________________________________
lstm_2 (LSTM)                (None, 4)                 144
_________________________________________________________________
dense_25 (Dense)             (None, 1)                 5
=================================================================
Total params: 20,149
Trainable params: 20,149
Non-trainable params: 0
```

Na obrázku 32 je zobrazena trénovací a validační správnost modelu. Tečkovaná křivka opět zobrazuje trénovací správnost a plná křivka validační správnost.

Křivka validační správnosti je po 10. epoše stále rostoucí, model má tedy z trénovacích dat stále užitek a může se naučit více nezávislých vzorů. Proto je možné přidat více skrytých parametrů. Validační správnost dosahovala hodnoty 0,9011 při 10. epoše, což je opět více než referenční hodnota 0,5 dosažená náhodným klasifikátorem.

Obrázek 32 - Delta Accuracy 01



4.5.2 **Trénování 02**

Ve druhém pokusu byl navýšen počet skrytých parametrů v LSTM vrstvě na 16, jak je vidět na obrázku 33. Tím se zvedl celkový počet parametrů na 82 129.

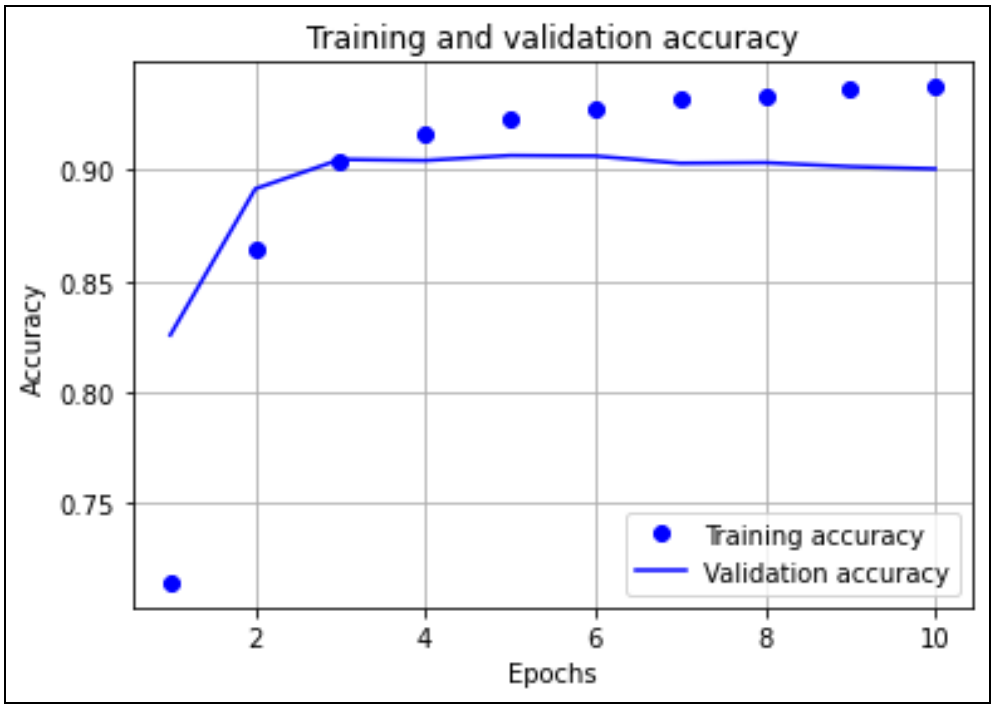
Obrázek 33 - Delta Summary 02

Layer (type)	Output Shape	Param #
embedding_10 (Embedding)	(None, 40, 16)	80000
lstm_3 (LSTM)	(None, 16)	2112
dense_26 (Dense)	(None, 1)	17

Total params: 82,129
Trainable params: 82,129
Non-trainable params: 0

Na obrázku 34 je vidět, že se validační správnost modelu zvýšila velmi rychle a hranice 0,9 byla překročena již ve 3. epoše. Maxima bylo dosaženo při 5. epoše s hodnotou 0.9065, poté začala správnost klesat.

Obrázek 34 - Delta Accuracy 02



Výsledný model Delta byl vybrán z trénování 02. Validační správnost modelu byla 90,7 %.

5 Výsledky a diskuse

Jednotlivé modely byly trénovány, validovány a testovány na různých množinách dat. Nemohlo tak dojít k propuštění informací z učení modelu, které by ovlivnilo testování. Konkrétní počet záznamů v každé množině je uveden v tabulce 4.

Množina	Počet
Trénovací data	16 000
Validační data	4 000
Testovací data	2 986
Celkem	22 986

Tabulka 4 - Přehled počtu záznamů datových množin

5.1 Srovnání modelů při učení

Učení modelů probíhalo na trénovacích a validačních datech. Tabulka 5 zobrazuje přehled parametrů jednotlivých modelů, mezi které patří například doba zpracování dávky, doba zpracování epochy nebo celkový počet parametrů v neuronové síti.

	Alfa	Beta	Gama	Delta
Doba zpracování dávky [ms]	28	25	24	10
Doba zpracování epochy [s]	4	3	3	1

Epocha s nejvyšší správností	4.	6.	13.	5.
Nejvyšší dosažená správnost	0,9060	0,8825	0,9032	0,9065
Celkový počet parametrů v síti	40 089	42 577	393 921	82 129

Tabulka 5 - Přehled parametrů získaných při trénování modelů

Z tabulky 5 vyplývá, že nejrychleji dokázal tréninková a validační data zpracovat model Delta, který dokázal jednu dávku o velikosti 128 novinových titulků zpracovat v průměru do 10 ms.

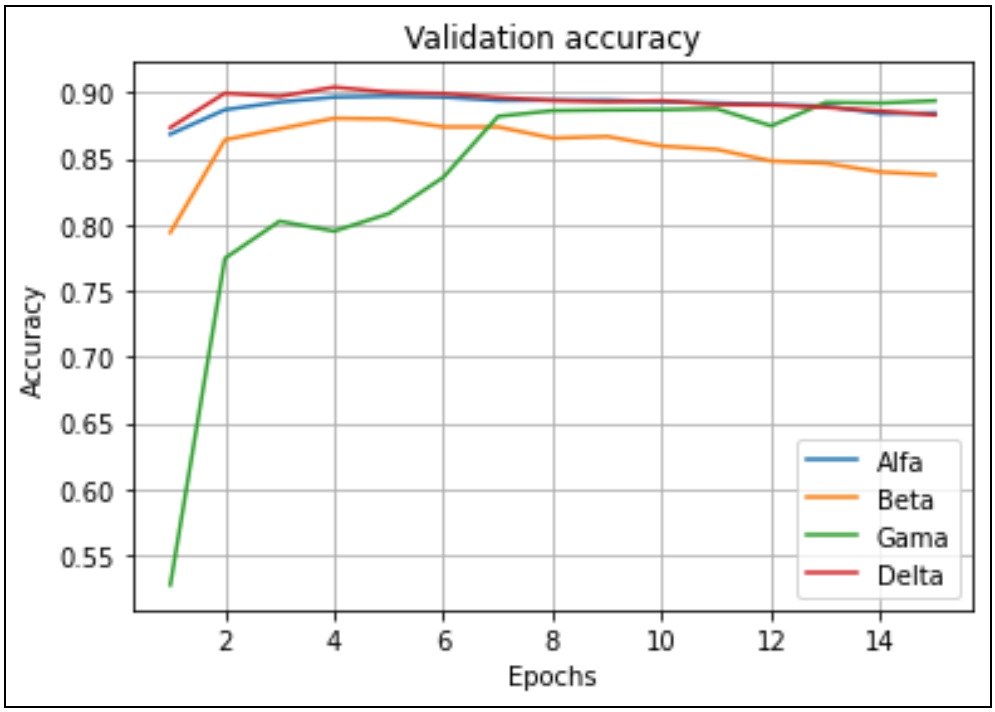
Nejvyšší dosažené správnosti dosahovaly modely Alfa, Beta a Delta kolem 5. epochy, modely se tedy učily podobným tempem. Model Gama se učil výrazně pomaleji a maximální správnosti dosahoval až okolo 13. epochy.

Obrázek 35 zobrazuje vývoj validační správnosti modelů. Z obrázku je patrné, že modely Alfa, Beta a Delta mají tendenci se rychle učit, ale také velmi rychle dosáhnou hranice přeučení a jejich validační správnost se začne zhoršovat. Model Gama se učí pomaleji a své hranice dosáhne až o několik epoch později.

Nejvyšší dosažená správnost modelů Alfa, Gama a Delta se pohybovala lehce nad 90 %. Model Beta za ostatními modely mírně zaostával se správností mírně nad 88 %.

K naučení se jednotlivých vzorů potřeboval nejvíce parametrů v neuronové síti model Gama, a to necelých 400 000. Naopak nejméně parametrů potřebovaly modely Alfa a Beta.

Obrázek 35 - Srovnání validační správnosti modelů



5.2 Srovnání modelů při testování

Testování modelů probíhalo na testovacích datech. Tabulka 6 zobrazuje přehled jednotlivých metrik odvozených z matice záměn popsané v kapitole *3.5.3*.

	Alfa	**Beta**	**Gama**	**Delta**
Accuracy	0,9060	0,8714	0,8891	0,8969
Precision	0,9118	0,8886	0,8801	0,9170
Recall	0,9157	0,8801	0,9281	0,8960
Specificity	0,8987	0,8501	0,9021	0,8728
F1 Score	0,9138	0,8843	0,9035	0,9064

Tabulka 6 - Přehled vypočítaných metrik získaných při testování

Vypočítané metriky vyšly pro modely Alfa, Gama a Delta bez větších rozdílů. S nepatrným rozdílem dosáhl nejvyšší správnosti model Alfa, který nedokázal správně ohodnotit 289 příkladů z celé testovací množiny.

Obdobně jako při validaci dosahoval model Beta mírně horších výsledků než zbývající tři modely, přestože špatně vyhodnotil pouze o 57 příkladů více než model Alfa.

Pokud srovnáme všechny naměřené hodnoty získané z učení a testovaní modelů, zjistíme, že i když všechny modely mají obdobné výsledky, nejlépe vycházel model Alfa, který dosahoval nejlepších jak při učení, tak i při testování.

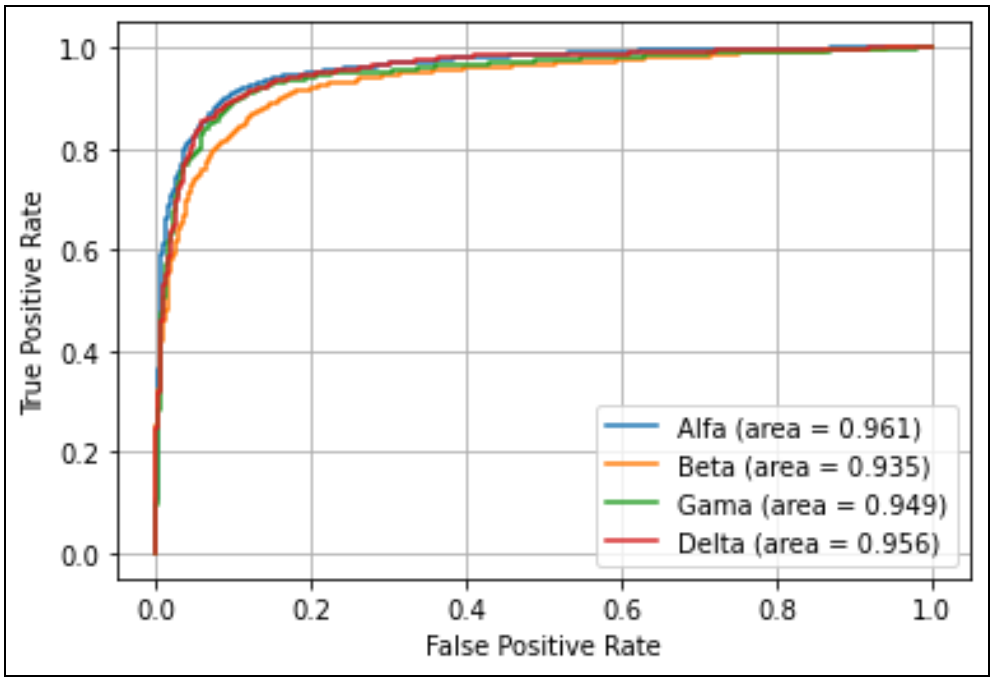
Tento závěr platí pouze pro mezní práh přednastavený na hodnotu 0,5. Takto nastavená hodnota zajišťuje rovnoměrné rozřazení příkladů do kategorií podle vypočítané pravděpodobnosti, kdy počet špatně identifikovaných nepravdivých článků je přibližně stejný jako počet špatně identifikovaných pravdivých článků.

Pokud by bylo cílem úlohy identifikovat co největší skupinu nepravdivých článků i za cenu zvýšení nepřesnosti při identifikaci pravdivých článků, model Delta by byl vhodnější.

Na obrázku 36 jsou zobrazeny ROC křivky jednotlivých modelů. Osy *x* a *y* zobrazeného grafu představují procentuální míry špatně a správně identifikovaných nepravdivých článků při různých hodnotách mezního prahu.

Model Delta má při vyšší míře správně identifikovaných nepravdivých článků mírně nižší míru špatně identifikovaných nepravdivých článků než ostatní modely.

Obrázek 36 - Srovnání ROC křivek modelů



5.3 Nedostatky řešení

Mezi největší nevýhody neuronových sítí patří práce s omezenými daty. Pokud by model zpracovával příklad, u kterého by nedokázal detekovat vzory naučené při trénování, nedokázal by ho správně klasifikovat.

Tato nevýhoda se dá částečně eliminovat postupným doučováním modelu, kdy se již za provozu model doučí nové vzory z nové, postupně rozšířené, tréninkové množiny. Touto metodou je provozovatel modelu schopen zajistit, aby model neopakoval stejné chyby na podobných datech.

6 Závěr

Hlavním cílem této diplomové práce bylo prokázat efektivitu moderních metod analýzy textu založenou na použití algoritmů hlubokého učení. K tomu bylo zapotřebí vybrat správné neuronové vrstvy a s jejich pomocí vytvořit neuronové sítě, které by při úloze binární klasifikace úspěšně překonaly referenční hodnotu stanovenou náhodným klasifikátorem. Hlavní cíl byl díky splnění obou dílčích cílů úspěšně naplněn.

V praktické části byla řešena úloha binární klasifikace na datové množině pravdivých a nepravdivých novinových článků. Pro její vyřešení byly vytvořeny čtyři různé modely Alfa, Beta, Gama a Delta využívající neuronové sítě a metody hlubokého učení. Všechny vytvořené modely dosahovaly při učení validační správnosti nad 85 % při klasifikaci, zda se jedná o pravdivý, nebo nepravdivý novinový článek.

V následující části diplomové práce byly porovnány výsledky všech zmíněných modelů dosažené na testovacích datech. Ke komparaci byly použity metriky odvozené z matice záměn, doba učení modelu, rychlost zpracování epochy nebo celkový počet parametrů v neuronové síti.

Pro výběr nejlepšího modelu byly všechny tyto zmíněné parametry velmi důležité, protože testovací správnost získaná z metrik vyšla pro všechny modely velmi podobně. Nakonec po zohlednění ostatních parametrů modelů bylo zjištěno, že danou úlohu dokáže nejlépe vyhodnotit model Alfa, využívající hustě propojené neuronové sítě s kódováním vstupních textů pomocí metody 1-z-n.

Dále byla ve výsledcích zmíněna diskuse nad využitím modelu se zvýšenou přesností klasifikace nepravdivých novinových článků za cenu snížení přesnosti při klasifikaci pravdivých článků. Pro tuto aplikaci by se nejlépe hodil model Delta, využívající rekurentní neuronovou sít s kódováním vstupních textů do hustých slovních vektorů.

7 Seznam použitých zdrojů

1. Song, D. A Short History of Deepfakes. Medium. https://medium.com/@songda/a-short-history-of-deepfakes-604ac7be6016 (accessed March 14, 2021)
2. Markoff, J. Google Cars Drive Themselves, in Traffic. The New York Times. https://www.nytimes.com/2010/10/10/science/10google.html (accessed March 14, 2021)
3. Walch, K. Why The Race For AI Dominance Is More Global Than You Think. Forbes. https://www.forbes.com/sites/cognitiveworld/2020/02/09/why-the-race-for-ai-dominance-is-more-global-than-you-think/?sh=52026296121f (accessed March 14, 2021)
4. Chatrná, I. S hledáním léku na covid-19 pomáhá brněnským vědcům umělá inteligence, software si vyvíjejí sami. iRozhlas. https://www.irozhlas.cz/veda-technologie/brno-vakcina-masarykova-univerzita-covid_2007301632_tkr (accessed March 14, 2021).
5. What is artificial intelligence—AI. ORACLE. https://www.oracle.com/artificial-intelligence/what-is-ai/ (accessed March 14, 2021)
6. Machine learning—defined. ORACLE. https://www.oracle.com/data-science/machine-learning/what-is-machine-learning/ (accessed March 14, 2021)
7. Artificial Intelligence and Machine Learning: Policy Paper, 2017. Internet Society. https://www.internetsociety.org/resources/doc/2017/artificial-intelligence-and-machine-learning-policy-paper/?gclid=CjwKCAjwnK36BRBVEiwAsMT8WAPLRBpNVUsAhWf3UA5V2O-oxAoQ0zGCEDjoa1HCwO6lx-2fygmjixoCOCgQAvD_BwE (accessed March 14, 2021)
8. Brownlee, J. Supervised and Unsupervised Machine Learning Algorithms, 2020. Machine Learning Mastery. https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/ (accessed March 14, 2021)
9. Koumakis, L. Deep learning models in genomics; are we there yet?, 2020. ScienceDirect. https://www.sciencedirect.com/science/article/pii/S2001037020303068 (accessed March 14, 2021).
10. Durčák, P. Neuronové sítě a princip jejich fungování, 2017. NaPočítači.cz. https://www.napocitaci.cz/33/neuronove-site-a-princip-jejich-fungovani-uniqueidgOkE4NvrWuNY54vrLeM670eFNQh552VdDDulZX7UDBY/ (accessed Jan 01, 2021).
11. Chollet, F. Deep learning v jazyku Python: knihovny Keras, Tensorflow. Přeložil Rudolf PECINOVSKÝ. Praha: Grada Publishing, 2019. Knihovna programátora (Grada). ISBN 978-80-247-3100-1
12. Patidar, P. Tensors — Representation of Data In Neural Networks, 2019. Medium. https://medium.com/mlait/tensors-representation-of-data-in-neural-networks-bbe8a711b93b (accessed Jan 01, 2021).
13. Brownlee, J. How Do Convolutional Layers Work in Deep Learning Neural Networks?, 2020. Machine Learning Mastery.

https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/ (accessed Jan 01, 2021).
14. Sumit, S. A Comprehensive Guide to Convolutional Neural Networks, 2018. towards data science. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (accessed Jan 01, 2021).
15. Chug, A. Introduction to Long Short Term Memory, 2019. GeeksforGeeks. https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/ (accessed Jan 01, 2021)
16. Soner, Y. Activation Functions in Neural Networks, 2020. Towards Data Science. https://towardsdatascience.com/activation-functions-in-neural-networks-eb8c1ba565f8 (accessed Jan 01, 2021)
17. Sagar, S. Activation Functions in Neural Networks, 2017. Towards Data Science. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6 (accessed Jan 01, 2021)
18. Brownlee, J. Your First Deep Learning Project in Python with Keras Step-By-Step, 2019. Machine Learning Mastery. https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/ (accessed Jan 01, 2021)
19. Brownlee, J. How to Choose Loss Functions When Training Deep Learning Neural Networks, 2019. Machine Learning Mastery. https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/ (accessed Jan 01, 2021)
20. Minaee, S. 20 Popular Machine Learning Metrics, 2019. Towards Data Science. https://towardsdatascience.com/20-popular-machine-learning-metrics-part-1-classification-regression-evaluation-metrics-1ca3e282a2ce (accessed Jan 01, 2021)
21. Brownlee, J. A Gentle Introduction to Threshold-Moving for Imbalanced Classification, 2020. Machine Learning Mastery. https://machinelearningmastery.com/threshold-moving-for-imbalanced-classification/ (accessed Jan 01, 2021)
22. Kód 1 Z N. PLC AUTOMATIZACE. http://plc-automatizace.cz/knihovna/data/kodovani/1-N-code.htm (accessed Jan 01, 2021)
23. Karani, D. Introduction to Word Embedding and Word2Vec, 2018. Towards Data Science. https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa (accessed Jan 01, 2021)
24. Serder, Y. What is TensorFlow? The machine learning library explained, 2019. InfoWorld. https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html (accessed Jan 01, 2021)
25. Heller, M. What is Keras? The deep neural network API explained, 2019. InfoWorld. https://www.infoworld.com/article/3336192/what-is-keras-the-deep-neural-network-api-explained.html (accessed Jan 01, 2021)
26. Cakl, O. Dezinformace, fake-news, bulvární zpráva, 2019. Transparency Internacional. https://www.transparency.cz/dezinformace-fake-news-bulvarni-zprava/ (accessed Jan 01, 2021)
27. Galarnyk, M. Install Python (Anaconda) on Windows, 2020. Medium. https://medium.com/@GalarnykMichael/install-python-anaconda-on-windows-2020-f8e188f9a63d (accessed Jan 01, 2021)

8 Přílohy

V této části diplomové práce jsou uvedeny přílohy, mezi které patří instalace distribuce Anaconda, instalace rozšiřujících modulů a zdrojové kódy modelů hlubokého učení.

8.1 Instalace distribuce Anaconda

Anaconda je distribuce programovacího jazyka Python, která obsahuje spoustu nástrojů, mezi něž mimo jiné patří Jupyter Notebook nebo vývojové prostředí Spyder.

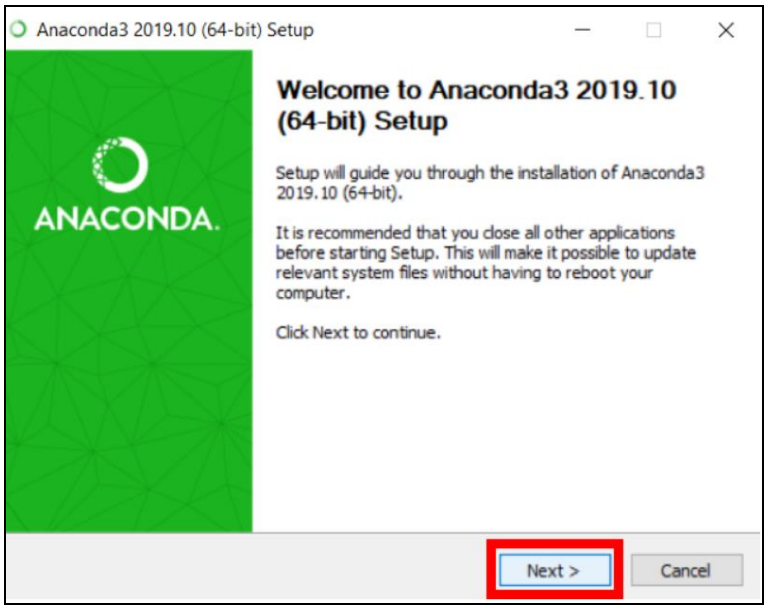
V této části kapitoly je popsána instalace Anacondy na operační systém Windows 10.

1) Prvním krokem pro nainstalování distribuce anakonda je jít na oficiální stránky distribuce: https://www.anaconda.com/products/individual a požadovanou verzi stáhnout.

2) Dalším krokem je spustit stažený soubor jako správce.

3) Po spuštění souboru by se mělo objevit okno průvodce instalací distribuce Anaconda. Klikněte na tlačítko „Next“.

Obrázek 37 - Instalace Anaconda



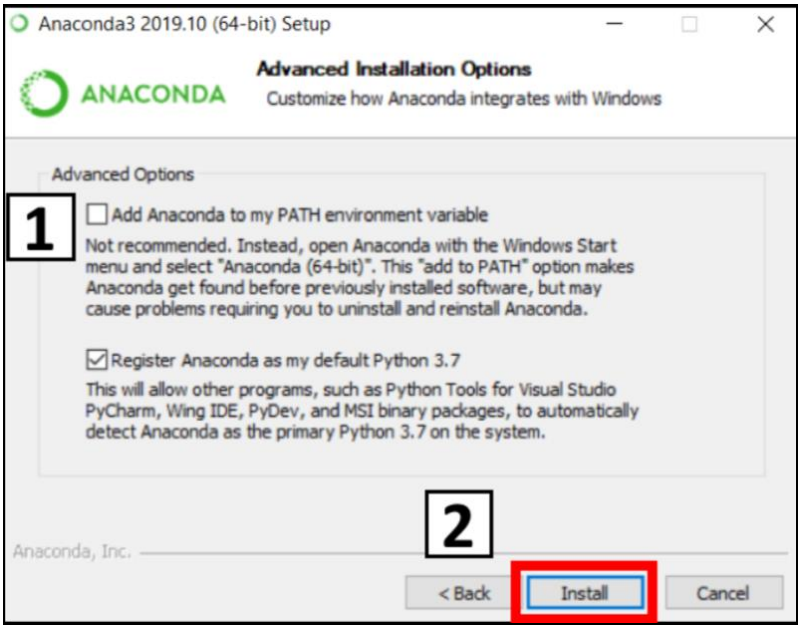
4) Zobrazí se okno licenčními podmínkami. Po jejich přečtení klikněte na „I Agree“.

5) Na následující obrazovce je nutné vybrat pro koho je instalace určena, jestli jen pro aktuálního uživatele, nebo pro všechny. Po vybrání klikněte na tlačítko „Next“.

6) Na další obrazovce je nutné vybrat, kam má být distribuce nainstalována. Po výběru umístění klikněte na tlačítko „Next“.

7) Na následující obrazovce je nutné se rozhodnout, zda chcete přidat distribuci Anaconda do položky PATH vašeho operačního systému a zda chcete registrovat Anacondu jako výchozí Python distribuci.

Obrázek 38 - Nastavení PATH Anaconda



Po zaškrtnutí vybraných políček klikněte na tlačítko „Install“.

8) Na další obrazovce probíhá instalace distribuce, po dokončení klikněte na tlačítko „Next“.

9) Další obrazovka nabízí možnost nainstalovat vývojové prostředí PyCharm po kliknutí na uvedený odkaz. Klikněte na tlačítko „Next“.

10) Poslední obrazovka nabízí možnost dozvědět se více o Anaconda Cloud nebo jak s distribucí Anaconda začít. Klikněte na tlačítko „Finish“ [27]

8.2 Instalace použitých modulů

Distribuce Anaconda obsahuje veliké množství již nainstalovaných knihoven. Přesto mezi předinstalované knihovny následující moduly nepatří, a proto se musí doinstalovat ručně.

8.2.1 Instalace knihovny TensorFlow

1) Prvním krokem pro nainstalování knihovny TensorFlow je otevření programu Anaconda Command Prompt.

2) Pro nainstalování aktuální CPU knihovny TensorFlow je nutné zadat příkaz „*conda create -n tensorflow*“.

Následně v programu lze ke knihovně přistupovat přes příkaz *import*.

8.2.2 Instalace knihovny Keras

1) Prvním krokem pro nainstalování knihovny Keras je otevření programu Anaconda Command Prompt.

2) Pro nainstalování aktuální knihovny Keras je nutné zadat příkaz „*install -c anakonda keras*“.

Následně v programu lze ke knihovně přistupovat přes příkaz *import*.

8.3 **Zdrojový kód modelu Alfa**

```
# -*- coding: utf-8 -*-

import numpy as np
import pandas as pd

cesta = "C:\\Users\\fake_news\\"

df_train = pd.read_csv(cesta + "train.csv")
df_test = pd.read_csv(cesta + "test.csv")
df_submit = pd.read_csv(cesta + "submit.csv")
df_test_sub = pd.concat([df_test,df_submit], axis=1)
df_test_sub = df_test_sub .drop("idS", axis = 1)

df_all = pd.concat([df_train,df_test_sub])

#%%

df_all = df_all.drop(["author", "text"], axis = 1)

df_all.dropna(inplace=True)

df_all = df_all.sample(frac=1)

X = df_all.drop(["id", "label"], axis = 1)
Y = df_all["label"]

count = df_all.label.value_counts()

df_all.reset_index(inplace = True, drop = True)
X.reset_index(inplace = True, drop = True)
Y.reset_index(inplace = True, drop = True)
#%%
from keras.preprocessing.text import Tokenizer

tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(X['title'])

one_hot_results = tokenizer.texts_to_matrix(X['title'], mode="binary")

x_all = one_hot_results
y_all = np.array(Y)

x_train = x_all[:20000]
y_train = y_all[:20000]

x_test = x_all[20000:]
y_test = y_all[20000:]
```

```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop

model = Sequential()
model.add(layers.Dense(8, activation="relu", input_shape=(5000,)))
model.add(layers.Dense(8, activation="relu"))
model.add(layers.Dense(1, activation="sigmoid"))

model.summary()

#%%

model.compile(optimizer=RMSprop(lr=1e-3),
              loss="binary_crossentropy",
              metrics=["acc"])

history = model.fit(x_train, y_train,
                    epochs=4,
                    batch_size=128,
                    validation_split=0.2)

import matplotlib.pyplot as plt

history_dict = history.history
loss_values = history_dict["loss"]
val_loss_values = history_dict["val_loss"]

epochs = range(1, len(loss_values) + 1)

plt.plot(epochs, loss_values, "ro", label="Training loss")
plt.plot(epochs,val_loss_values, "r", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.grid()
plt.show()

plt.clf()

acc_values = history_dict["acc"]
val_acc_valuesA = history_dict["val_acc"]

epochs = range(1, len(acc_values) + 1)

plt.plot(epochs, acc_values, "bo", label="Training accuracy")
plt.plot(epochs,val_acc_valuesA, "b", label="Validation accuracy")
```

```
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid()
plt.show()
plt.clf()

#%%

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score

y_pred = model.predict_classes(x_test)

print(confusion_matrix(y_test, y_pred))

print('accuracy = %0.6f' % accuracy_score(y_test,y_pred))
print('recall = %0.6f' % recall_score(y_test,y_pred))
print('precision = %0.6f' % precision_score(y_test,y_pred))
print('F1 = %0.6f' % f1_score(y_test,y_pred))

#%%
from sklearn.metrics import roc_curve, auc
from numpy import sqrt, argmax
y_pred = model.predict(x_test).ravel()

fprA, tprA, tresholdsA = roc_curve(y_test, y_pred)
aucA = auc(fprA, tprA)

plt.plot(fprA, tprA, label='ROC (area = %0.3f)' % aucA)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()
plt.clf()

#%%
results = model.evaluate(x_test, y_test)
```

8.4 Zdrojový kód modelu Beta

```
# -*- coding: utf-8 -*-

import numpy as np
import pandas as pd
from keras.preprocessing.sequence import pad_sequences

cesta = "C:\\Users\\fake_news\\"

df_train = pd.read_csv(cesta + "train.csv")
df_test = pd.read_csv(cesta + "test.csv")
df_submit = pd.read_csv(cesta + "submit.csv")
df_test_sub = pd.concat([df_test,df_submit], axis=1)
df_test_sub = df_test_sub .drop("idS", axis = 1)

df_all = pd.concat([df_train,df_test_sub])

#%%

df_all = df_all.drop(["author", "text"], axis = 1)

df_all.dropna(inplace=True)

df_all = df_all.sample(frac=1)

X = df_all.drop(["id", "label"], axis = 1)
Y = df_all["label"]

count = df_all.label.value_counts()

df_all.reset_index(inplace = True, drop = True)
X.reset_index(inplace = True, drop = True)
Y.reset_index(inplace = True, drop = True)
#%%
from keras.preprocessing.text import Tokenizer

max_features = 5000
max_len = 40

tokenizer = Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(X['title'])

sequences = tokenizer.texts_to_sequences(X['title'])

padding = pad_sequences(sequences,maxlen=max_len,padding = "pre")

x_all = np.array(padding)
```

```
y_all = np.array(Y)

x_train = x_all[:20000]
y_train = y_all[:20000]

x_test = x_all[20000:]
y_test = y_all[20000:]


from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop

model = Sequential()
model.add(layers.Embedding(max_features, 8, input_length=max_len))
model.add(layers.Flatten())
model.add(layers.Dense(8, activation="relu"))
model.add(layers.Dense(8, activation="relu"))
model.add(layers.Dense(1, activation="sigmoid"))

model.summary()

#%%
import tensorflow as tf

model.compile(optimizer=RMSprop(lr=1e-3),
              loss="binary_crossentropy",
              metrics=["acc"])

history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.2)

import matplotlib.pyplot as plt

history_dict = history.history
loss_values = history_dict["loss"]
val_loss_values = history_dict["val_loss"]

epochs = range(1, len(loss_values) + 1)

plt.plot(epochs, loss_values, "ro", label="Training loss")
plt.plot(epochs,val_loss_values, "r", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.grid()
```

```
plt.show()

plt.clf()

acc_values = history_dict["acc"]
val_acc_valuesB = history_dict["val_acc"]

#%%
val_acc_valuesB2 = []
for prvek in val_acc_valuesB:
    val_acc_valuesB2.append(prvek - 0.02)

#%%

epochs = range(1, len(acc_values) + 1)

plt.plot(epochs, acc_values, "bo", label="Training accuracy")
plt.plot(epochs,val_acc_valuesB2, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid()
plt.show()

#%%
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score

y_pred = model.predict_classes(x_test)

print(confusion_matrix(y_test, y_pred))

print('accuracy = %0.6f' % accuracy_score(y_test,y_pred))
print('recall = %0.6f' % recall_score(y_test,y_pred))
print('precision = %0.6f' % precision_score(y_test,y_pred))
print('F1 = %0.6f' % f1_score(y_test,y_pred))

#%%
from sklearn.metrics import roc_curve, auc
from numpy import sqrt, argmax
y_pred = model.predict(x_test).ravel()

fprB, tprB, tresholdsB = roc_curve(y_test, y_pred)
aucB = auc(fprB, tprB)
```

```
plt.plot(fprB, tprB, label='ROC (area = %0.3f)' % aucB)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()
plt.clf()

#%%
results = model.evaluate(x_test, y_test)
```

8.5 Zdrojový kód modelu Gama

```
# -*- coding: utf-8 -*-

import numpy as np
import pandas as pd
from keras.preprocessing.sequence import pad_sequences

cesta = "C:\\Users\\fake_news\\"

df_train = pd.read_csv(cesta + "train.csv")
df_test = pd.read_csv(cesta + "test.csv")
df_submit = pd.read_csv(cesta + "submit.csv")
df_test_sub = pd.concat([df_test,df_submit], axis=1)
df_test_sub = df_test_sub .drop("idS", axis = 1)

df_all = pd.concat([df_train,df_test_sub])

#%%

df_all = df_all.drop(["author", "text"], axis = 1)

df_all.dropna(inplace=True)

df_all = df_all.sample(frac=1)

X = df_all.drop(["id", "label"], axis = 1)
Y = df_all["label"]

count = df_all.label.value_counts()

df_all.reset_index(inplace = True, drop = True)
X.reset_index(inplace = True, drop = True)
Y.reset_index(inplace = True, drop = True)
#%%
from keras.preprocessing.text import Tokenizer

max_features = 5000
max_len = 40

tokenizer = Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(X['title'])

sequences = tokenizer.texts_to_sequences(X['title'])

padding = pad_sequences(sequences,maxlen=max_len,padding = "pre")

x_all = np.array(padding)
```

```
y_all = np.array(Y)

x_train = x_all[:20000]
y_train = y_all[:20000]

x_test = x_all[20000:]
y_test = y_all[20000:]

from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop

model = Sequential()
model.add(layers.Embedding(max_features, 16, input_length=max_len))
model.add(layers.Conv1D(16, 9, activation="relu"))
model.add(layers.MaxPooling1D(2))
model.add(layers.Conv1D(16, 9, activation="relu"))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(1))

model.summary()
#%%
model.compile(optimizer=RMSprop(lr=1e-4),
              loss="binary_crossentropy",
              metrics=["acc"])

history = model.fit(x_train, y_train,
                    epochs=15,
                    batch_size=128,
                    validation_split=0.2)

import matplotlib.pyplot as plt

history_dict = history.history
loss_values = history_dict["loss"]
val_loss_values = history_dict["val_loss"]

epochs = range(1, len(loss_values) + 1)

plt.plot(epochs, loss_values, "ro", label="Training loss")
plt.plot(epochs,val_loss_values, "r", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.grid()
plt.show()

plt.clf()
```

```
acc_values = history_dict["acc"]
val_acc_valuesC = history_dict["val_acc"]

epochs = range(1, len(acc_values) + 1)

plt.plot(epochs, acc_values, "bo", label="Training accuracy")
plt.plot(epochs,val_acc_valuesC, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid()
plt.show()

#%%
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score

y_pred = model.predict_classes(x_test)

print(confusion_matrix(y_test, y_pred))

print('accuracy = %0.6f' % accuracy_score(y_test,y_pred))
print('recall = %0.6f' % recall_score(y_test,y_pred))
print('precision = %0.6f' % precision_score(y_test,y_pred))
print('F1 = %0.6f' % f1_score(y_test,y_pred))

#%%
from sklearn.metrics import roc_curve, auc
from numpy import sqrt, argmax
y_pred = model.predict(x_test).ravel()

fprC, tprC, tresholdsC = roc_curve(y_test, y_pred)
aucC = auc(fprC, tprC)

plt.plot(fprC, tprC, label='ROC (area = %0.3f)' % aucC)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()
plt.clf()
#%%
results = model.evaluate(x_test, y_test)
```

8.6 Zdrojový kód modelu Delta

```
# -*- coding: utf-8 -*-

import numpy as np
import pandas as pd
from tensorflow import keras
from keras.preprocessing.sequence import pad_sequences

cesta = "C:\\Users\\fake_news\\"

df_train = pd.read_csv(cesta + "train.csv")
df_test = pd.read_csv(cesta + "test.csv")
df_submit = pd.read_csv(cesta + "submit.csv")
df_test_sub = pd.concat([df_test,df_submit], axis=1)
df_test_sub = df_test_sub .drop("idS", axis = 1)

df_all = pd.concat([df_train,df_test_sub])

#%%

df_all = df_all.drop(["author", "text"], axis = 1)

df_all.dropna(inplace=True)

df_all = df_all.sample(frac=1)

X = df_all.drop(["id", "label"], axis = 1)
Y = df_all["label"]

count = df_all.label.value_counts()

df_all.reset_index(inplace = True, drop = True)
X.reset_index(inplace = True, drop = True)
Y.reset_index(inplace = True, drop = True)

#%%

from keras.preprocessing.text import Tokenizer

max_features = 5000
embedding_vector_features = 10
maxlen=40

tokenizer = Tokenizer(num_words=max_features)
tokenizer.fit_on_texts(X['title'])

sequences = tokenizer.texts_to_sequences(X['title'])
```

```
padding = pad_sequences(sequences,maxlen=maxlen,padding = "pre")

x_all = np.array(padding)
y_all = np.array(Y)

x_train = x_all[:20000]
y_train = y_all[:20000]

x_test = x_all[20000:]
y_test = y_all[20000:]


model = keras.Sequential([
    keras.layers.Embedding(5000, embedding_vector_features, input_length = maxlen),
    keras.layers.LSTM(embedding_vector_features),
    keras.layers.Dense(1, activation = 'sigmoid')
  ])

model.summary()
#%%
from keras.optimizers import RMSprop

model.compile(loss = 'binary_crossentropy',
            optimizer = RMSprop(lr=1e-3),
            metrics = ['acc'])

history = model.fit(x_train, y_train,
                  epochs=10,
                  batch_size=128,
                  validation_split=0.2)

import matplotlib.pyplot as plt

history_dict = history.history
loss_values = history_dict["loss"]
val_loss_values = history_dict["val_loss"]

epochs = range(1, len(loss_values) + 1)

plt.plot(epochs, loss_values, "ro", label="Training loss")
plt.plot(epochs,val_loss_values, "r", label="Validation loss")
plt.title("Training and validation loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.grid()
plt.show()

plt.clf()
```

```
acc_values = history_dict["acc"]
val_acc_valuesD = history_dict["val_acc"]

epochs = range(1, len(acc_values) + 1)

plt.plot(epochs, acc_values, "bo", label="Training accuracy")
plt.plot(epochs,val_acc_valuesD, "b", label="Validation accuracy")
plt.title("Training and validation accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid()
plt.show()

#%%

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score

y_pred = model.predict_classes(x_test)

print(confusion_matrix(y_test, y_pred))

print('accuracy = %0.3f' % accuracy_score(y_test,y_pred))
print('recall = %0.3f' % recall_score(y_test,y_pred))
print('precision = %0.3f' % precision_score(y_test,y_pred))
print('F1 = %0.3f' % f1_score(y_test,y_pred))

#%%
from sklearn.metrics import roc_curve, auc
from numpy import sqrt, argmax
y_pred = model.predict(x_test).ravel()

fprD, tprD, tresholdsD = roc_curve(y_test, y_pred)
aucD = auc(fprD, tprD)

plt.plot(fprD, tprD, label='ROC (area = %0.3f)' % aucD)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()
plt.clf()
#%%
results = model.evaluate(x_test, y_test)
```