Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Klasifikace textových dokumentů pomocí neuronových sítí

Plzeň 2018 Petr Hlaváč

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 27. června 2018

Petr Hlaváč

Poděkování

Děkuji Ing. Ladislavu Lencovi, Ph.D. za jeho ochotu, čas a cenné připomínky k obsahu i zpracování. Dále bych chtěl také poděkovat organizaci MetaCentra za poskytnutí výpočetních prostředků, neboť tato práce vznikla za podpory projektů CERIT Scientific Cloud (LM2015085) a CESNET (LM2015042) financovaných z programu MŠMT Projekty velkých infrastruktur pro VaVaI.

Abstract

This Bechelor thesis focuses on the automatic text document classification with neural networks. The porpuse is to verify the effect of different text preprocessing types on quality of the classification. For text preprocessing we used tokanization, lemmatisation and stemming. Five neural network architectures were tested: multi-layer perceptron, convolutional neural networks, recurrent neural networks and their combinations. Testing was carried out on czech datataset CDCv2 and english RCV1-v2. Achieved results were compared with literature.

Abstrakt

Bakalářská práce se zabývá automatickou klasifikací textových dokumentů pomocí neuronových sítí. Cílem práce je zjistit vliv metody předzpracování dokumentů na výslednou kvalitu klasifikace. Použité metody předzpracování jsou tokenizace, lemmatizace a stemming. Celkem bylo testováno pět architektur neuronových sítí: vícevrstvý perceptron, konvoluční neuronová síť, rekurentní neuronová síť a jejich kombinace. Testování bylo provedeno na české databázi CDCv2 a anglické RCV1-v2. Dosažené výsledky byly následně porovnány s literaturou.

Obsah

1 Úvod **1**

2 Klasifikace dokumentů **2**
2.1 Binární . . . 2
2.2 Multi–class . . . 2
2.3 Multi–label . . . 2
2.4 Vyhodnocení úspěšnosti Multi–label klasifikace . . . 3
2.4.1 Statistické metody . . . 3

3 Předzpracování dokumentů **6**
3.1 Tokenizace . . . 6
3.2 Stemming . . . 6
3.3 Lemmatizace . . . 6
3.4 POS-tagging . . . 7
3.5 Nástroje pro předzpracování . . . 7
3.5.1 Universal Dependecies . . . 7
3.5.2 High Precision Stemmer . . . 9

4 Reprezentace dokumentů **10**
4.1 Bag of Words . . . 10
4.2 Integer Encoding . . . 11

5 Databáze **12**
5.1 CDCv2 . . . 12
5.2 RCV1v2 . . . 13
5.2.1 Topic codes . . . 13

6 Umělé neuronové sítě **15**
6.1 Biologický neuron . . . 15
6.2 Matematický model neuronu . . . 15
6.2.1 Perceptron . . . 16
6.3 Architektury neuronových sítí . . . 16
6.3.1 Fully-connected vrstva . . . 17
6.3.2 Konvoluční vrstva . . . 17
6.3.3 MaxPooling vrstva . . . 18
6.3.4 Rekurentní neuronové sítě . . . 19

7 Návrhy architektur **23**
7.1 Vícevrstvý perceptron 23
7.2 CNN 24
7.3 LSTM 24
7.4 RNN + CNN 25
7.5 CNN + RNN 26

8 Implementace neuronových sítí **27**
8.1 Keras 27
8.1.1 TensorFlow 27
8.2 Metacentrum 28
8.2.1 Přihlášení 28
8.2.2 Žádost o výpočetní úlohu 28
8.2.3 Úloha 30
8.2.4 Moduly 30
8.2.5 Instalace Kerasu 30

9 Dosažené výsledky **33**

10 Závěr **37**

Literatura **38**

A Uživatelská dokumentace **40**
A.1 Příprava dat 40
A.2 Architektury 42

B Seznam nevýznamových slov **44**

C Statistické výsledky **45**

1 Úvod

V dnešní době je počet textových dokumentů v digitální formě velmi vysoký. Aby bylo možné se v takto rozsáhlém množství rozumně orientovat je nutné dokumenty třídit do jednotlivých kategorií. Manuálně ovšem tato práce vyžaduje expertně zaměřené lidi a je velmi časově náročná. Z tohoto důvodu se stále více rozvíjí metody automatické klasifikace. K řešení této automatizace se ve velkém množství případů využívá metod strojového učení, které nám poskytují různé klasifikační algoritmy. Tato práce se ovšem zaměřuje pouze na jednu část a to v posledních letech velmi populární umělé neuronové sítě.

Samotný klasifikační algoritmus ale také není všemocný a výsledky jsou do velké míry ovlivněny kvalitou zvolených příznaků. K tomu, abychom je dokázali vhodně zvolit z nestrukturovaných textových dokumentů, je nutné dokumenty předzpracovávat. To většinou probíhá v několika krocích, kde základem je kvalitní tokenizace, následovaná převedením všech slov na buďto velká nebo malá písmena. Dalším častý krokem bývá filtrace interpunkčních znamének, nebo nahrazení číslovek slovy. Mezi hlavní metody předzpracování patří lemmatizace, neboli převod slova na jeho základní tvar a také stemming, kde nahrazujeme slova jejich kmenem nebo kořenem.

Cílem této práce je prostudovat metody pro klasifikaci textových dokumentů pomocí neuronových sítí, seznámit se s implementací těchto sítí v dostupných knihovnách, implementovat zvolené metody a ověřit jejich funkčnost na datech ve dvou rozdílných jazycích a to v českém a anglickém. Závěrem práce bude prostudování vlivu různých metod předzpracování textových dokumentů na zvolené metody klasifikace.

Čtenář by tedy po přečtení práce měl porozumět problému klasifikace textových dokumentů, jejich předzpracování a následně vhodné reprezentaci pro klasifikátor sestavený z umělých neuronových sítí. Zároveň by měl získat přehled o tom, jaký vliv má předzpracování na výsledky klasifikace.

2 Klasifikace dokumentů

Klasifikace je úloha, kdy máme předem známý počet kategorií a snažíme se určit, do které z nich dané pozorování patří. Kategorie se určují na základě trénovací množiny. Ta obsahuje pozorování u nichž jsou kategorie předem známé. Samotná klasifikace je poté prováděna klasifikátorem, který se skládá ze vstupního vektoru $\vec{x}$ obsahujícího příznakový popis objektu a výstupního identifikátoru třídy ω. Cílem této práce je klasifikace dokumentů. Vstupní množinou tedy budou textové dokumenty, které budeme následně klasifikovat do různých kategorii neboli témat. Vstup do klasifikátoru je tedy příznakový popis dokumentu d a klasifikátor zapíšeme jeho klasifikační funkcí $f(d)$, jejíž výstupem je identifikátor třídy ω. Klasifikační úlohu dále můžeme rozdělit na tři typy.

2.1 Binární

Binární klasifikace znamená, že máme pouze dvě kategorie, které se vzájemně vylučují. Tedy výstupní identifikátor pro funkci klasifikátor $f(d)$ je $\omega = x : x\epsilon\{0, 1\}$ Tento typ klasifikace se používá například při filtrování pošty, kde rozdělujeme zprávy do dvou kategorií a to běžná zpráva nebo spam.

2.2 Multi–class

Multi-class klasifikace vychází z binární s tím rozdílem, že už nemáme pouze dvě třídy, ale libovolný, předem stanovený počet n, kdy dokument řadíme právě a pouze do jedné kategorie. Pokud bychom tedy předchozí příklad upravili na klasifikaci do kategorií důležitá zpráva, běžná, nebo spam. Vznikla by nám klasifikace typu multi–class. Můžeme tedy říct, že výstupem klasifikační funkce $f(d)$ je $\omega = \vec{v} = (v_1, v_2, ..., v_n) : v\epsilon\{0, 1\}$, kde jednotlivé složky vektoru v označují kategorie dokumentu a pro multi–class klasifikaci platí, že $\forall v\epsilon\vec{v} : \exists!k\epsilon\{1, 2, ..., n\} : v_k = 1$.

2.3 Multi–label

Posledním typem je multi–label klasifikace. Vycházíme zde z typu multi-class, kde máme libovolný předem stanový počet tříd, ale dokument se může

nacházet ve více kategoriích zároveň. Do toho typu spadá právě klasifikování novinových článků podle témat, neboť v jednom článku může být témat hned několik. Opět je tedy výstupem klasifikační funkce $f(d)$ vektor $\omega = \vec{v} = (v_1, v_2, ..., v_n) : v\epsilon\{0, 1\}$. pro jehož prvky tentokrát platí, že neexistuje pouze jeden jednotkový ale může jich být několik $\forall v\epsilon\vec{v} : \exists k\epsilon\{1, 2, ..., n\} : v_k = 1$.

2.4 Vyhodnocení úspěšnosti Multi–label klasifikace

Multi–Label klasifikace se dá řešit dvěma způsoby. Buď to sadou binárních klasifikátorů, kde máme jeden klasifikátor pro každou kategorii nebo jedním klasifikátorem jehož výstup je vektor, u něhož každá z určovaných kategorií má přidělený svůj index. V případě, že dokument do kategorie patří je hodnota vektoru na indexu 1 v opačném případě 0. Pro daný index se tedy jedná o klasifikaci binární, kde buď se článek v kategorii nachází nebo nikoliv. Abychom určili kvalitu klasifikátoru musíme si určit všechny případy, které mohou při klasifikování nastat. Označení případů vychází z anglických slov kde T (True) a F (False) značí pravdivost a P (Positive) a N (Negative) určuje typ výsledku.

TP Klasifikátor správně určil, že se článek v kategorii nachází

TN Klasifikátor správně určil, že se článek v kategorii nenachází.

FP Klasifikátor chybně určil, že se článek v kategorii nachází.

FN Klasifikátor chybně určil, že se článek v kategorii nenachází.

Pro každý prvek vektoru je určen právě jeden případ a pro každý dokument získáme tolik případů kolik se snažíme určit kategorií. Výsledky klasifikace pro stejné kategorie následně sečteme a vyhodnotíme pomocí statistických metod.

2.4.1 Statistické metody

Přesnost (Accuracy) je poměr správně určených případů a celkového počtu všech klasifikovaných dokumentů.

$$acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

Úplnost (Sensitivity nebo také Recall) je poměr správně určených kladných případů a všech těch, které měly být klasifikovány jako kladné.

$$recall = \frac{TP}{TP + FN} \tag{2.2}$$

Přesnost (Precision) je poměr správně určených kladných klasifikací a klasifikátorem klasifikovaných jako kladných.

$$precision = \frac{TP}{TP + FP} \tag{2.3}$$

F–míra (F–measure nebo také F_1–score) je harmonický průměr úplnosti a přesnosti. F–Míra dosahuje hodnot od 0 do 1, kde 1 značí nejlepší výsledek a 0 nejhorší.

$$F_1 = \frac{2}{\frac{1}{recall} + \frac{1}{precision}} \tag{2.4}$$

Pro celkové vyhodnocení celé testovací sady se používají dva přístupy.

Micro-Average

Metoda spočívá v tom, že individuálně sečteme TP, TN, FP a FN pro všechny testované dokumenty a statistické metody poté aplikujeme nad celým systémem najednou. Při použití této metriky se tedy uvažuje, že všechny kategorie přispívají k výsledným statistickým metodám stejným dílem.

Macro-Average

Pro výpočet Macro-average je potřeba celou sadu testovacích dat rozdělit na menší části. V našem případě bude jedna část odpovídat výsledkům spadajícím do jedné kategorie. Pro tuto část vypočteme výsledné hodnocení pomocí metody Micro-average a tento postup opakujeme pro všechny zbývající kategorie dokumentů. Následně uděláme aritmetický průměr Precision a Recall všech kategorií a získané hodnoty dosadíme do vzorce 2.4 pro F–míru. Tento výpočet na rozdíl od Micro-average vypovídá spíše o tom, jak si klasifikátor vede v průměru přes všechny rozdělené části.

Cross–validation

Křížová validace (Cross–validation) je postup, který se používá převážně pro databáze obsahující nízký počet testovacích vzorků. Celou databázi rozdělíme na k částí, ze kterých následně pouze jednu část použijeme jako testovací data a zbytek jako trénovací. Pro takto natrénovaný klasifikátor provedeme statistické výpočty. Poté obměníme část s testovacími daty za jednu

z trénovacích a natrénujeme nový klasifikátor a takto pokračujeme dále. Ve výsledku tedy získáme k klasifikátorů, kde pro každý máme jiná trénovací a testovací data. Vyhodnocení klasifikace je poté aritmetický průměr statistických výpočtů všech klasifikátorů.

3 Předzpracování dokumentů

Dokumenty se před klasifikací upravují ve snaze dosáhnout lepších klasifikačních výsledků. Běžně se pro předzpracování dokumentů používají metody popsané v této kapitole.

3.1 Tokenizace

Tokenizace je způsob předzpracování, kdy sekvence po sobě jdoucích znaků označíme jako tokeny. V našem případě se jedná o tokenizaci dokumentů. Procházíme tedy dokumenty po sekvencích znaků, které reprezentují náš token. Správné určení tokenů ale vždy není zcela jasné. Tokeny by měly co nejlépe zachovávat původní stavbu věty. Problém tedy nastává v tom, jak od sebe různé sekvence rozeznat. Naivním přístupem je použít pro oddělení bílé znaky. Poté se nám ale může stát, že například víceslovné názvy, které bychom chtěli zachovat jako jeden token budou rozděleny. Další problém také nastává u interpunkčních znamének. Většinou bývají nesprávně zařazena v předchozím tokenu, místo aby měla svůj vlastní.

3.2 Stemming

Stematizace (Stemming) souvisí s nahrazováním původních slov v dokumentu jejich kmeny nebo v ideálním případě kořeny. Stematizace je realizována odsekáváním předpon a koncovek slov. Předzpracování tímto stylem se používá pro zredukování tvarů zapsání jednoho slova. Celý tento postup je velmi závislý na jazyku dokumentu. V češtině se pro základní postup používá seznam možných koncovek a jejich odstraňování.

3.3 Lemmatizace

V dokumentech se velmi často nachází více tvarů stejného slova. Lemmatizace je proces, kdy tyto tvary nahrazujeme takzvanými lemmaty neboli slovníkovými tvary. Proces je velmi blízký Stemmingu, neboť v některých případech se lemma a kořen slova shodují. Ovšem existují také případy, kdy jsou zcela odlišné. Hlavním cílem lemmatizace je stejně jako u Stemmingu eliminace více tvarů stejného slova.

3.4 POS-tagging

POS (part of speech) tagging slouží k přiřazování odpovídající jazykové kategorie slovům. V češtině se jako tato kategorie používají právě slovní druhy. Pro jejich správné určení je ovšem důležité správně porozumění kontextu slova použitého ve větě. Toho je ovšem strojově velmi těžké docílit. Předzpracování touto metodou nám umožňuje klasifikovat dokumenty na základě stavby jejich vět. Také nám přidává upřesňující informaci o slovech nacházejících se v dokumentech.

3.5 Nástroje pro předzpracování

Nástrojů pro předzpracování dokumentů existuje mnoho. Naším cílem je ale naleznout takové, které nejsou zaměřené pouze na jeden jazyk. Použití rozdílných nástrojů by mohlo vést ke vnesení nepřesností při porovnávání vlivu předzpracování na daný jazyk.

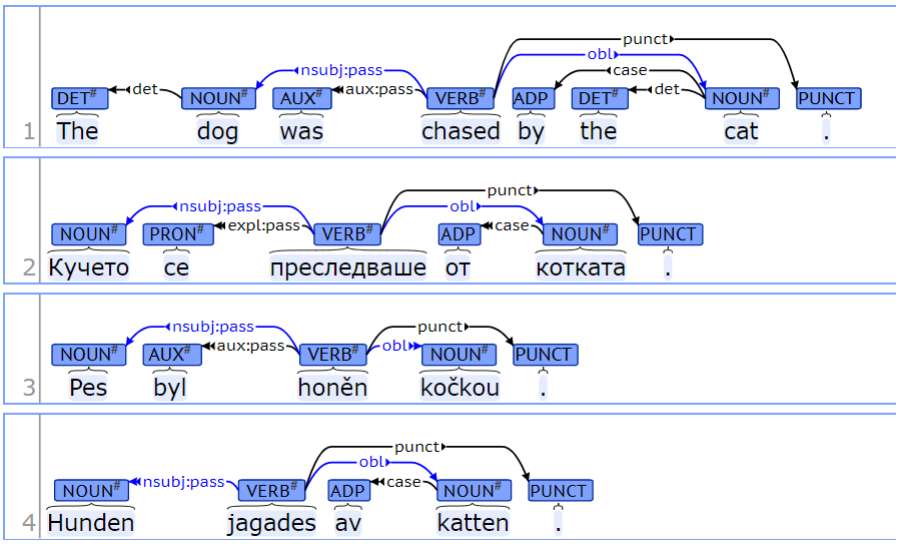
3.5.1 Universal Dependecies

Universal dependecies (dále jen UD) je projekt, který se zaměřuje na vytváření konzistentních databází pro porovnání stavby vět v textech z mnoha různých jazyků. Cílem projektu je usnadnit vývoj vícejazyčných analyzátorů a analyzovat výzkum z hlediska jazykové typologie. Na tento projekt jsou kladeny následující požadavky. UD musí být správné z hlediska jazykové analýzy pro jednotlivé jazyky. Musí být také vhodné pro rychlou a důslednou anotaci prováděnou člověkem a jejich formát má být snadno počítačově zpracovatelný.

UDPipe

UDPipe [21] je výtvorem Institutu formální a aplikované Lingvistiky na Univerzitě Karlově. Jejich cílem bylo vytvořit jednoduchý program pro tokenizaci, lemmatizaci a POS-taggingu textových dokumentů v různých světových jazycích. Vstupem programu jsou textové dokumenty bez jakéhokoliv předzpracování a výstup jsou soubory typu CONNLL-U, což je typický formát právě pro UD.

Zdrojové soubory aplikace jsou dostupné na GitHubu [1] a po splnění požadavků uvedených na [2] je možné program sestavit spouštěním *make* v src složce zdrojového kódu.



Obrázek 3.1: Příklad rozboru věty pro anglický, bulharský, český a švédský jazyk.

```
# sent_id = 1
# text = They buy and sell books.
1   They     they    PRON    PRP    Case=Nom|Number=Plur              2   nsubj   2:nsubj|4:nsubj   _
2   buy      buy     VERB    VBP    Number=Plur|Person=3|Tense=Pres   0   root    0:root            _
3   and      and     CONJ    CC     _                                 4   cc      4:cc              _
4   sell     sell    VERB    VBP    Number=Plur|Person=3|Tense=Pres   2   conj    0:root|2:conj     _
5   books    book    NOUN    NNS    Number=Plur                       2   obj     2:obj|4:obj       SpaceAfter=No
6   .        .       PUNCT   .      _                                 2   punct   2:punct           _

# sent_id = 2
# text = I have no clue.
1   I        I       PRON    PRP    Case=Nom|Number=Sing|Person=1     2   nsubj   _   _
2   have     have    VERB    VBP    Number=Sing|Person=1|Tense=Pres   0   root    _   _
3   no       no      DET     DT     PronType=Neg                      4   det     _   _
4   clue     clue    NOUN    NN     Number=Sing                       2   obj     _   SpaceAfter=No
5   .        .       PUNCT   .      _                                 2   punct   _   _
```

Obrázek 3.2: CONLL-U format.

Spuštění UDPipe:

```
udpipe --tokenize --tag --output=connl
   path_to_udpipe_model [text documents]
```

Listing 3.1: Vzorový příkaz pro spušení UDPipe.

V případě malého množství dokumentů je možné využít webovou službu [3], její hlavní nevýhodou je ovšem nutnost zpracovávat dokumenty po jednom, což při předzpracování velkého datasetu může trvat v řádech dnů.

3.5.2 High Precision Stemmer

High Precision Stemmer [5] je vícejazyčný nástroj pro stematizaci vytvořený na principu strojového učení bez učitele. K dispozici jsou natrénované modely pro 11 jazyků včetně českého a anglického. Tento nástroj byl vytvořen na Katedře informatiky a výpočetní techniky, Fakulty aplikovaných věd, Západočeské univerzity v Plzni. Implementace stemmeru je k dispozici v Javě a pouze pro nekomerční účely. Aplikace pro stemming vyžaduje, již tokenizovaná data v UTF-8 kódování.

Stemmer je distribuován ve formě knihovny. Po jejím importu je možná provést stemming následujícím způsobem.

```
1 //maximum length of suffix that can be stripped off
2 int maxSuffixLength = 3;
3 //load model
4 Stemmer stemmer =
     StemmerBuilder.loadStemmer("model.bin",
     maxSuffixLength);
5 //stemming
6 String stem = stemmer.getClass("stolem");
```

Listing 3.2: Vzorové využití knihovny HPS.

4 Reprezentace dokumentů

Neuronové sítě nemohou pracovat s textem přímo. Text musí být převeden na číselné vektory, které mají předem fixně stanovenou délku. Pro řešení tohoto problému existují různé přístupy.

4.1 Bag of Words

Bag of Words je základní a velmi naivní přístup. Z dokumentů je vytvořen slovník, který obsahuje předem stanovený počet slov. Pořadí slova ve slovníku určuje také zároveň pozici daného slova v příznakovém vektoru. Dokument je tedy ve výsledku reprezentován vektorem fixní délky o velikosti předem vytvořeného slovníku a číselná hodnota na každé pozici vektoru počet výskytů slova v dokument.

Například máme dva krátké dokumenty.

> *Honza se rád díva na televizi. Marie televizi nesleduje.*
>
> *Honza se také rád dívá na fotbal.*

Slovník vytvoříme z unikátních slov nacházejících se v dokumentech.

"Honza"	*"na"*	*"také"*
"se"	*"televizi"*	*"fotbal"*
"rád"	*"Marie"*	
"díva"	*"nesleduje"*	

Z následujícího slovníku by poté vektory pro jednotlivé dokumenty byly následující

> *První dokument:* $[1, 1, 1, 1, 1, 2, 1, 1, 0, 0]$
>
> *Druhý dokument:* $[1, 1, 1, 1, 1, 0, 0, 0, 1, 1]$

Použitím tohoto přístupu se ovšem vzdáváme informace o tom na jaké pozici se dané slovo v dokumentu nachází, co mu předcházelo nebo co bude následovat, přicházíme tedy úplně o jeho kontext.

4.2 Integer Encoding

Integer encododing se naopak snaží zachovat pozice slov v dokumentu. Opět zkonstruujeme slovník unikátních slov. Tentokrát je ale každému slovu přidělen unikátní klíč. Ten odpovídá číslu pozice, na které se slovo ve slovníku nachází. Příznakový vektor tentokrát odpovídá délce textového dokumentu. V dokumentu poté postupujeme token po tokenu a přiřazujeme jim hodnoty klíčů ze slovníku. Tento přístup má dva problémy. Délky vektorů nejsou fixní. Je tedy nutné předem stanovit přesnou délku vektoru a následně kratší dokumenty doplnit speciálně vyhrazenou hodnotou. Ta se většinou volí jako 0. Delší dokumenty musíme naopak zkrátit. Tokeny přesahující limitní délku se odstraňují. Druhým problémem je, že slovník nemusí obsahovat slovo nalezené ve větě. Libovolně se tedy zvolí ještě jedna unikátní hodnota reprezentující právě tato slova.

Pro vytvoření slovníku využijeme věty z předchozího příkladu.

1. "Honza"
2. "se"
3. "rád"
4. "díva"
5. "na"
6. "televizi"
7. "Marie"
8. "nesleduje"
9. "také"
10. "fotbal

Jelikož máme krátké dokumenty. Stanovíme si pevnou délku vektoru podle nejdelšího z dokumentů. Příznakové vektory poté budou vypadat takto.

První dokument: $[1, 2, 3, 4, 5, 6, 1, 6, 8]$

Druhý dokument: $[1, 2, 9, 3, 4, 5, 10, 0, 0]$

5 Databáze

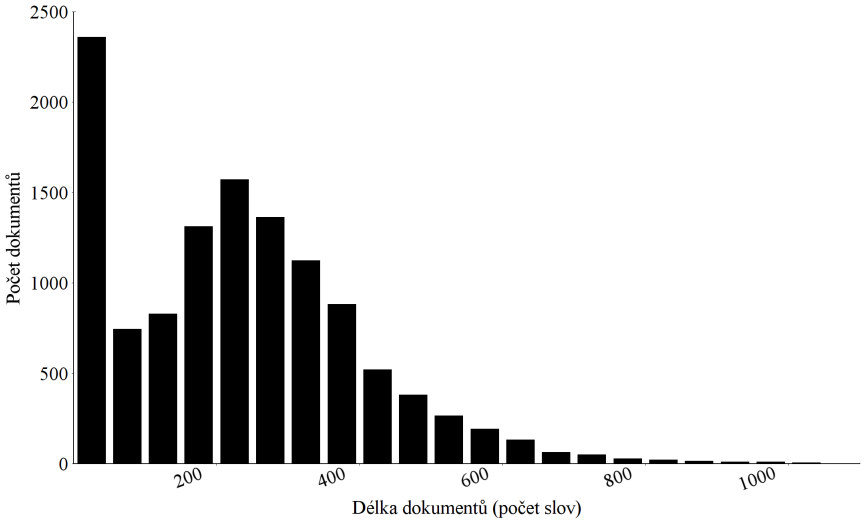
Pro experimenty byly využity dvě databáze. Jedna česká poskytnutá ČTK a druhá anglická od americké zpravodajské agentury Reuters. Obě databáze jsou zdarma dostupné pouze pro vědecké účely.

5.1 CDCv2

Česká databáze se nazýva Czech Text Document Corpus v2 [11]. Celkem obsahuje 11955 ručně klasifikovaných novinových článků, které jsou řazeny do 60 kategorií. Všechny tyto dokumenty jsou zároveň dostupné ve třech typech předzpracování a to v tokenizované verzi, lematizované verzi a ve formě POS-taggů. Předzpracování bylo uděláno plně automaticky pomocí nástroje UDPipe. V tabulce 5.1 jsou ukázány statistické údaje o korpusu. Tyto údaje jsou mírně odlišné od literatury [11] z důvodu filtrace interpunkčních znamének v průběhu předzpracování dokumentů. Na obrázků 5.1 je histogram, z kterého lze vyčíst rozložení jednotlivých délek dokumentů v korpusu.

Prvky	Počet prvků	Prvky	Počet prvků
Dokumenty	11 955	Unikátní slova	145 541
Kategorie	60	Unikátní lemma	72 274
Slova	2 981 392	Unikátní stem	70 499

Tabulka 5.1: Statistický rozbor CDCv2.



Obrázek 5.1: Rozdělení dokumentů dle délky v CDCv2.

5.2 RCV1v2

Americká databáze nese název Reuters Corpus Volume I [13]. Obsahuje přes 800000 manuálně klasifikovaných novinových článků. Články jsou uloženy ve formátu XML a obsahují titulek, datum a místo vydání, text článku, a klasifikované kategorie.

5.2.1 Topic codes

Kategorie topic codes byly vybrány, aby vystihovaly hlavní význam článku. Narozdíl od českého korpusu byly rozděleny do čtyřech hierarchických skupin: CCAT (Corporate/Industrial), ECAT (Economics), GCAT (Government/Social) a MCAT (Markets). Celkově bylo v databázi definováno 126 kategorií. Ovšem použito jich bylo pouze 103. Kategorie jsou řazeny hierarchicky do stromu například C311 (Domestic markets) je potomkem C31 (Markets/Marketing), která je podkategorií CCAT (Corporate/Industrial).

Trénovací množina

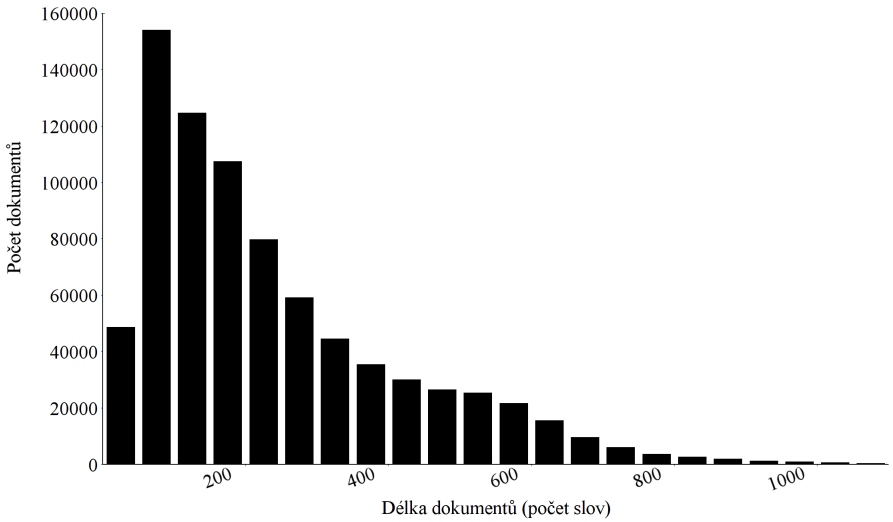
Databáze má doporučenou chronologicky zvolenou trénovací množinu v intervalu ID od 2286 do 26150. Jedná se o 23149 dokumentů publikovaných v rozmezí 11 dnů roku 1996. Tato trénovací množina obsahuje, u skoro každé kategorie Topic codes, alespoň dva nebo více článků. Výjimku tvoří pouze dvě kategorie, která zde nejsou obsaženy. Vzhledem k absenci trénovacích dat je v literatuře [13] doporučeno nezahrnovat tyto kategorie do vyhodnocení. Dostáváme se tedy na celkové číslo 101 kategorií. S takto zvolenou trénovací množinou nám poté zbývá 781265 dokumentů do množiny testovací.

Statistický rozbor

V tabulce 5.2 jsou ukázány statistické informace o korpusu RCV1v2, ze kterých vyplývá, že na rozdíl od českého korpusu má lemmatizace na počet unikátních slov mnohem menší vliv, zatímco stemming si zde vede lépe. Z histogramu na obrázku 5.2 je poté vidět, že korpusu dominují dokumenty s počtem slov od 50 po 200.

Prvky	Počet prvků	Prvky	Počet prvků
Dokumenty	806 791	Unikátní slova	434 471
Kategorie	126	Unikátní lemma	423 437
Slova	198 021 893	Unikátní stem	330 815

Tabulka 5.2: Statistický rozbor RCV1v2.



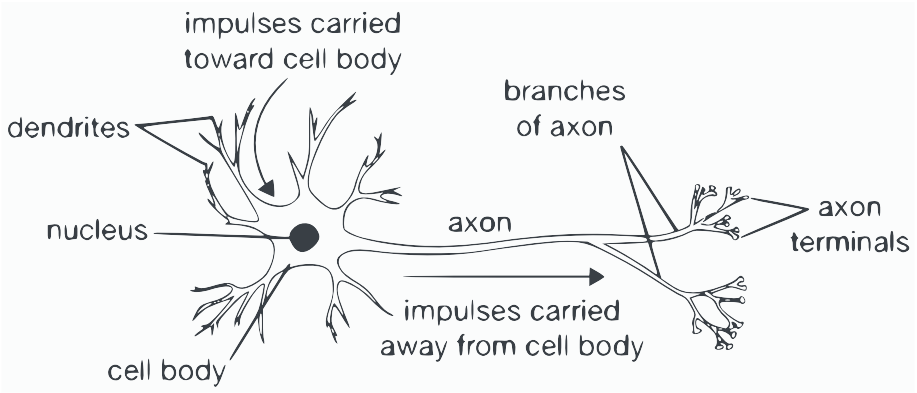
Obrázek 5.2: Rozdělení dokumentů dle délky v RCV1v2.

6 Umělé neuronové sítě

Koncept umělých neuronových sítí původně vznikal s cílem vytvořit funkční model biologických neuronových systémů. Postupem času se ale zjistilo, že tento koncept, z velké části biologickými systémy inspirován, dosahuje dobrých výsledků v úlohách zabývajících se strojovým učením.

6.1 Biologický neuron

Základní výpočetní jednotkou mozku je neuron. V lidském mozku lze naleznout přibližně 86 bilionů neuronů. Ke komunikaci mezi neurony dochází pomocí synapsí. Těch se v nervovém systému nachází přibližně 10^4 až 10^5. Vstupem do každého neuronu jsou dendrity a výstupem je jediný axon, který se nakonec rozvětví a připojí pomocí synapsí na dendrity ostatních neuronů.

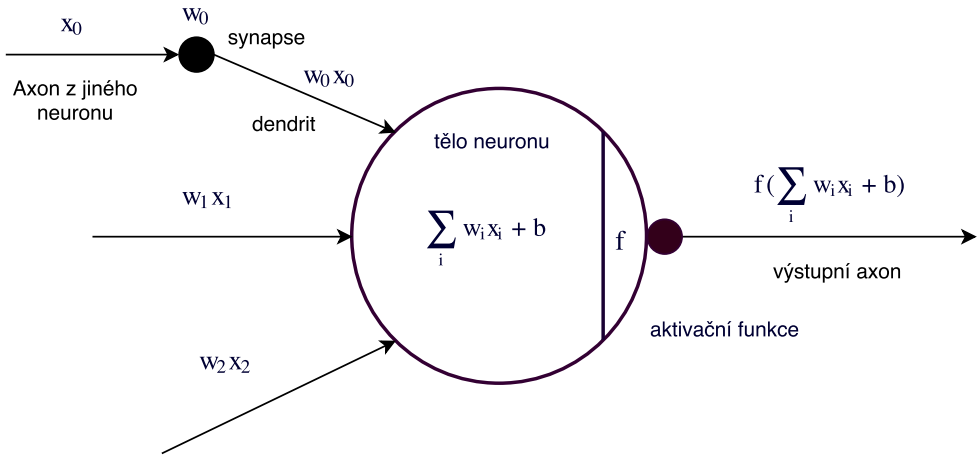


Obrázek 6.1: Biologický neuron [4].

6.2 Matematický model neuronu

V matematickém modelu zobrazeném na níže uvedeném obrázku signál (x_0) putuje axonem, kde násobně reaguje ($x_0 w_0$) v závislosti na síle synapse (w_0) s dendritem následujícího neuronu. Myšlenka je taková, že síly synapse (hodnoty w) jsou trénovatelné a řídí to, jaký vliv má neuron na jeho následovníka. V základním modelu vedou všechny denrity do těla neuronu, kde jsou sečteny. V případě, že přesáhnou určitou hranici, může neuron vyslat signál po axonu výstupním. V matematickém modelu předpokládáme, že nezáleží na

načasování signálů, ale na jejich frekvenci. Hodnota této frekvence je v modelu simulována aktivační funkcí (f). Tuto funkci je možné volit a často se využívá například sigmoidální funkce, neboť vezme sumu vstupních hodnot do neuronu a výslednou hodnotu namapuje na interval $< 0, 1 >$.



Obrázek 6.2: Matematický model neuronu.

6.2.1 Perceptron

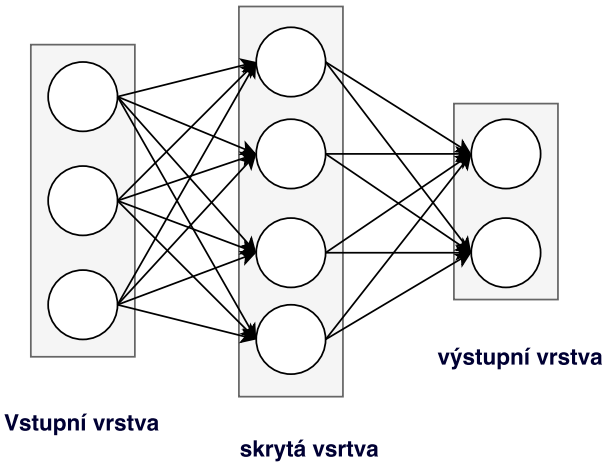
Nejjednodušším modelem umělé neuronové sítě je právě perceptron, tento model je složen ze základního matematického modelu neuronu. Perceptron ze své podstaty umí řešit pouze lineárně separovatelné úlohy. Z čehož vyplývá, že jeho nejlepší využití je pro binární klasifikátor.

6.3 Architektury neuronových sítí

Neuronové sítě jsou modelovány jako směsice neuronů, které tvoří acyklický graf. To znamená, že výstup některých neuronů může být použit jako vstup jiných. Cykly v grafech nejsou povoleny, neboť by to znamenalo smyčku v grafu při dopředném průchodu. Aby byl graf sítě přehledný, neurony jsou často řazeny do jednotlivých vrstev.

6.3.1 Fully-connected vrstva

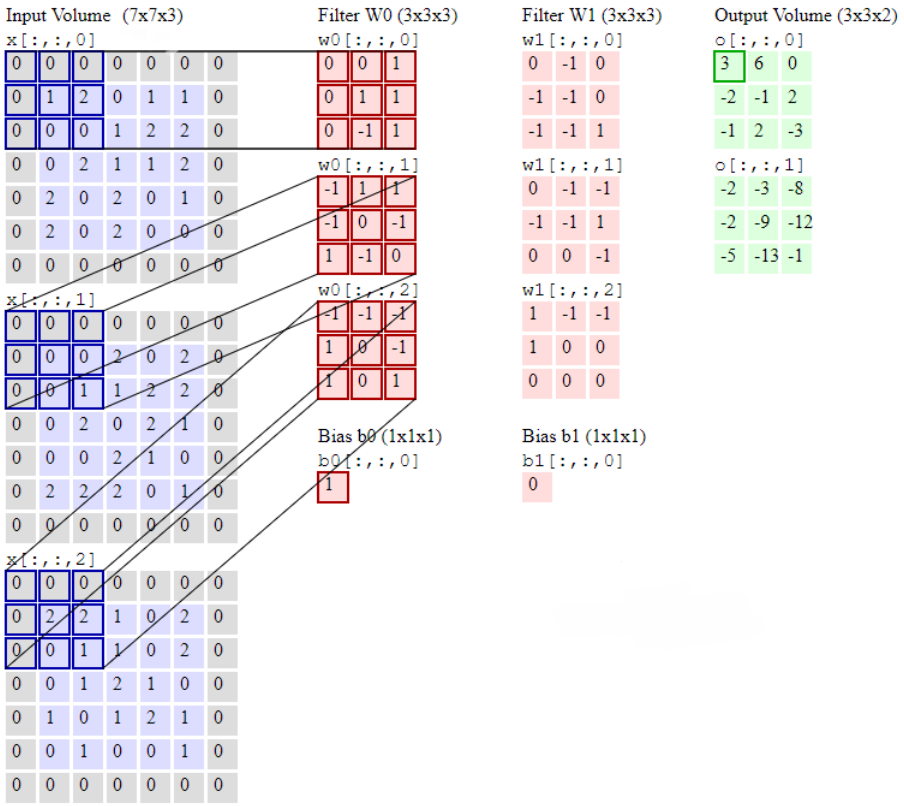
Tato vrstva je navržena tak, že neurony v jedné vrstvě mezi sebou nesdílejí žádné vazby, avšak každý neuron jedné vrstvy má vazbu na všechny neurony vrstvy následující viz obrázek.



Obrázek 6.3: Fully-connected vrstva.

6.3.2 Konvoluční vrstva

Konvoluční vrstva [4] se skládá ze sady trénovatelných filtrů. Filtry mají obvykle malé rozměry, ale aplikují se přes celou hloubku vstupních dat. Vstupními daty je tedy třírozměrná matice, kde třetí rozměr je označován právě jako hloubka. Typicky se toto vyskytuje u obrázků, kde data pro různé barvy RGB jsou uložena v rozdílných vrstvách matice.



Obrázek 6.4: Princip konvoluční vrstvy.[4]

Postup aplikace filtru na vstupní data z obrázku 6.4

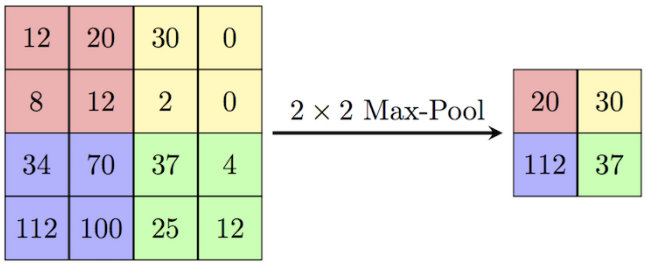
$$
\begin{aligned}
&D1 = 0 \cdot 0 + 0 \cdot 0 + 0 \cdot 1 + 0 \cdot 0 + 1 \cdot 1 + 2 \cdot 1 + 0 \cdot 0 + 0 \cdot (-1) + 0 \cdot 1 \\
&D2 = 0 \cdot (-1) + 0 \cdot 1 + 0 \cdot 1 + 0 \cdot (-1) + 0 \cdot 0 + 0 \cdot (-1) + 0 \cdot 1 + 0 \cdot (-1) \\
&+1 \cdot 0 \\
&D3 = 0 \cdot (-1) + 0 \cdot (-1) + 0 \cdot (-1) + 0 \cdot 1 + 2 \cdot 0 + 2 \cdot (-1) + 0 \cdot 1 + 0 \cdot 0 \\
&+1 \cdot 1 \\
&OV[0][0] = D1 + D2 + D3 + b0 = 3 + 0 - 1 + 1 = 3
\end{aligned}
\tag{6.1}
$$

Počet filtrů v konvoluční vrstvě je volitelný. Vrstva se trénuje pomocí změn hodnot bias a jednotlivých prvků filtrů.

6.3.3 MaxPooling vrstva

Slouží k redukci parametrů předávaných neuronovou sítí. Zároveň se používá k zmenšení vlivu overfittingu neboli přeučení, což si můžeme vysvětlit tím, že klasifikátor je příliš závislý na vstupních datech a není schopen generalizovat. Tato vrstva není trénovatelná. Jejímy parametry jsou velikost pole ve kterém se hledá maximum a jeho posun. Na obrázku 6.5 je zobrazen vliv

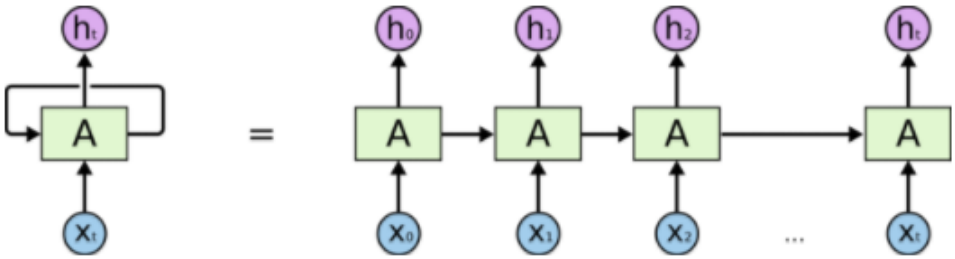
Maxpooling o velikosti 2x2 a posunu 2.



Obrázek 6.5: Funkce MaxPooling vrstvy.[4]

6.3.4 Rekurentní neuronové sítě

Pro pochopení konkrétního slova v textu je nutné znát souvislosti ze slov předchozích. Stejně jako v případě, že bychom chtěli určit jaká událost probíhá v každém momentě nějakého filmu. Pro řešení těchto problémů je nutné si uchovávat nějakou informaci, která může ovlivnit ty následující. A právě na tento problém se specializují rekurentní neuronové sítě [18]. Jejich architektura je specifická tím, že mají smyčky. Díky tomu se zároveň zdá, že vypadají komplikovaně, avšak tyto smyčky je možné snadno rozepsat do snadnější pochopitelné reprezentace.

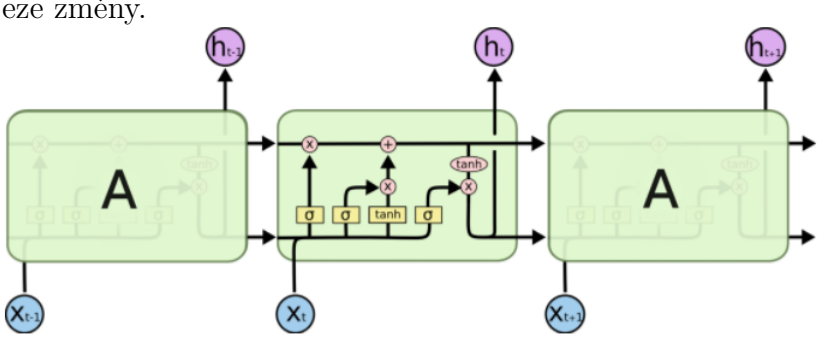


Obrázek 6.6: Rozepsaná rekurentní síť.[18]

U přenášení předchozích znalostní pro ovlivnění té následující ale dochází k několika problémům. Jedním z nich je vzdálenost dvou bloků, které se navzájem ovlivňují. V případech kdy je tato vzdálenost malá, jsou rekurentní sítě schopné se tuto spojitost naučit. V opačném případě i přes to, že člověk by dokázal stanovit vhodné nastavení jednotlivých částí sítě, tak běžné učící techniky selhávají.

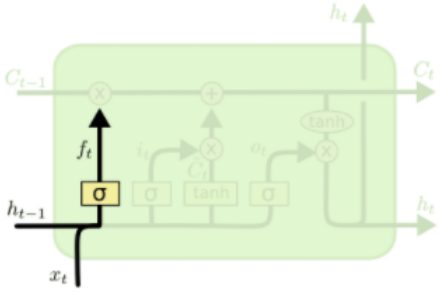
Long Short Term Memory

Long Short Term Memory (dále jen LSTM) jsou druhem rekurentních sítí, který byl navržen tak, aby byl schopen se učit právě dlouhodobé souvislosti. Na obrázku 6.7 je vyobrazena LSTM buňka. Její hlavní částí je horizontální čára vedoucí vrchem diagramu, která prochází celou sérií buněk. Tato čára se nazývá stav buňky. Vstupní informace tudy prochází pouze s malými lineárními změnami. Je tedy velmi snadné, aby touto částí prošla kompletně informace beze změny.



Obrázek 6.7: LSTM buňky.[18]

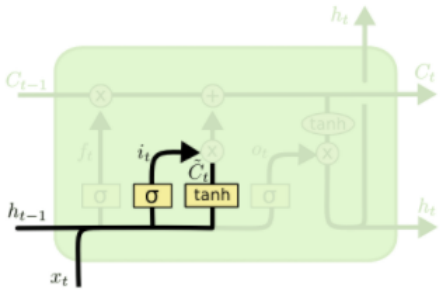
Prvním krokem LSTM vrstvy je rozhodnout, kterou informaci ze stavu buňky už nebudeme dále potřebovat a máme ji zapomenout. To rozhodnutí závisí na vrstvě se sigmoidální aktivační funkcí, která je také nazývana jako "zapomínací brána". Tato vrstva mapuje interval hodnot od <0,1>, kde 1 znamená ponechat a 0 kompletně zapomenout.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Obrázek 6.8: Forget gate layer.[18]

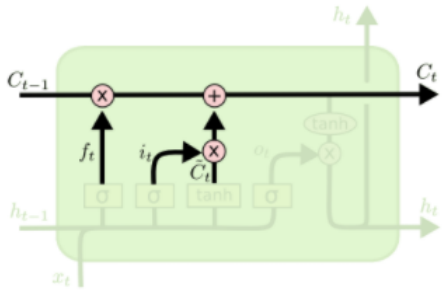
Dalším krokem je rozhodnutí jaká informace bude uchována ve stavu buňky. Tento proces se skládá ze dvou částí. První je opět vrstva se sigmoidální aktivační funkcí, která bude tentokrát rozhodovat, které hodnoty se mají upravit. Tato vrstva se také nazývá "vstupní brána". Ta je následována další vrstvou s aktivační funkcí tanh. Ta má na starosti sestavit vektor nových kandidátů, kteří by mohli být přidáni k aktuálnímu stavu.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Obrázek 6.9: Input and update gate layer.[18]

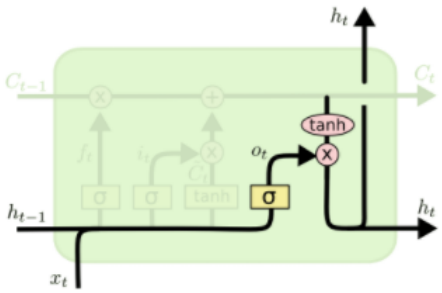
Po těchto dvou krocích je načase aktualizovat stav buňky na nově získaný v předchozích dvou krocích. Prvně vynásobíme aktuální stav f_t, věcmi které jsme se rozhodli zapomenout. A poté přičteme $i_t \cdot \tilde{C}_t$, což reprezentuje nové kandidáty, vynásobené hodnotou, která říká jak moc se mají jednotlivé hodnoty v každém stavu upravit.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Obrázek 6.10: Aktualizace stavu buňky.[18]

Posledním krokem je určení výstupních dat z buňky. Výstup bude vycházet z aktuálního vstupu buňky avšak bude ještě dále filtrován. Opět bude použita funkce sigmoid, která tentokrát bude rozhodovat o tom jaká data budou na výstupu. Stav buňky projde přes vrstvu s aktivační funkcí tanh, která namapuje hodnoty na interval $< -1, 1 >$. Poté je vynásobíme s výstupem vrstvy sigmoid, abychom dále posílali pouze hodnoty, které mají nějakou důležitou informaci.

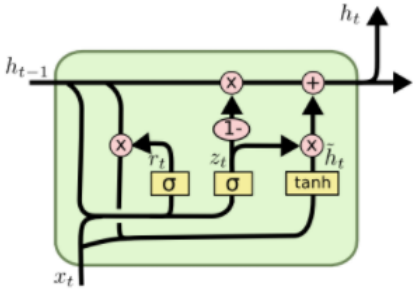


$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Obrázek 6.11: Výstup buňky.[18]

Gated Reccurent Unit

Je jednou z mnoha variací vycházejících z LSTM buňky. Její hlavní změnou je, že kombinuje "zapomínací"a "vstupní vrstvu brány"a zároveň také spojuje stav buňky a její vstupní informaci. Reprezentace GRU buňky je zobrazena na 6.12 i s vzorci nutnými pro výpočet jednotlivých kroků.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Obrázek 6.12: Gated Recurrent Unit.[18]

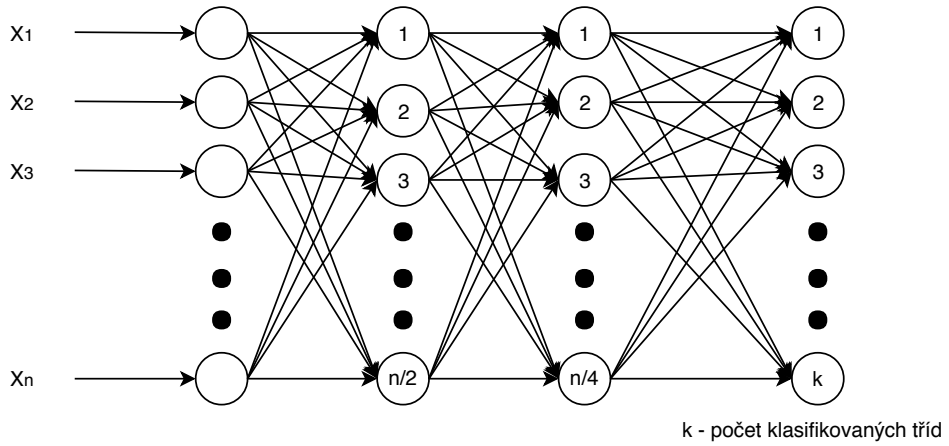
Výsledkem těchto úprav vznikla buňka, která je jednodušší pro výpočet i trénování. To má za následek, že její popularita stoupá a je velmi často používána.

7 Návrhy architektur

Neuronové sítě se skládají z vrstev, které je možno různě kombinovat. Existuje tedy mnoho různých způsobů, jak sestavit úspěšný klasifikátor pro řešení zadaného problému. Jednu věc mají ale všechny zmíněné architektury společnou a to je výstupní vrstva, kde výstupem je vektor obsahující reálná čísla. Abychom dostali námi chtěné výsledky binární klasifikace pro jednotlivé kategorie, je nutné provést prahování. Určíme tedy hodnotu prahu p, kde všechny hodnoty menší než p budou rovny 0 a větší než p jedničce. Volba prahu má na úspěšnost klasifikace velký vliv a najít optimální hodnotu není jednoduché. Pro účely této práce byla hodnota zvolena empiricky pro CNN architekturu $p = 0,05$, pro ostatní architektury se vstupem integer encoding $p = 0,2$ a pro MLP architektur $p = 0,5$. Hodnoty byly stanoveny ve snaze vybalancovat statistické metody precission a recall. Jako aktivační funkce výstupní vrstvy pro všechny architektury byla volena sigmoida.

7.1 Vícevrstvý perceptron

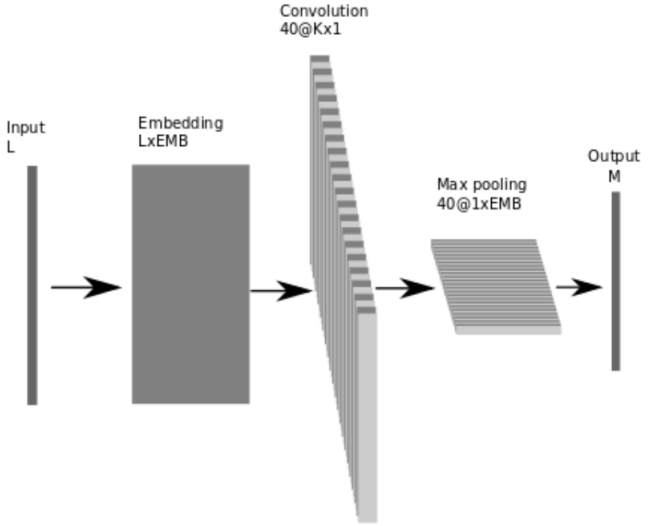
Vícevrstvý perceptron (dále jen MLP) je jednou z nejzákladnějších architektur. Je složena pouze z Fully-connected vrstev. Počet neuronů ve skrytých vrstvách se řídí jednoduchým pravidlem a to takovým, že každá má polovinu buněk než předcházející. Vstupní vrstva byla stanovena na 10000 neuronů z důvodu paměťové náročnosti a počet kategorií k odpovídá počtu neuronů vrstvy výstupní.



Obrázek 7.1: Více vrstvý perceptron.[18]

7.2 CNN

Architektura konvoluční neuronové sítě byla převzata z vědeckého článku [12]. Vstupem do této architektury je vektor délky L, zpracovaný pomocí integer encoding. Na volbě L závisí paměťová náročnost při trénování sítě, je tedy vhodné volit tuto hodnotu s ohledem na rozložení délky klasifikovaných dokumentů a dostupnou paměť. Podle těchto dvou kriterií byla zvolena hodnota $L = 400$. Druhá vrstva se nazývá Embedding layer. Ta vytváří pro každé vstupní slovo vektorovou reprezentaci stanovené délky EMB. Dokument je poté reprezentován maticí, kde L je počet řádků matice a EMB sloupců. Třetí vrstva je konvoluční. V této vrstě je použito N_c konvolučních filtrů o velikosti $K \times 1$. Provádí se tedy 1D konvoluce přes jednu pozici v embedding vektoru a K vstupních slov. Následující vrstva provádí max pooling o velikost $L - K + 1$, jejímž výstupem $N_c 1 \times EMB$ vektorů. Tyto vektory jsou následně spojeny do jednoho a poslány do výstupní vrstvy obsahující M buněk, kde M je počet klasifikovaných kategorií.

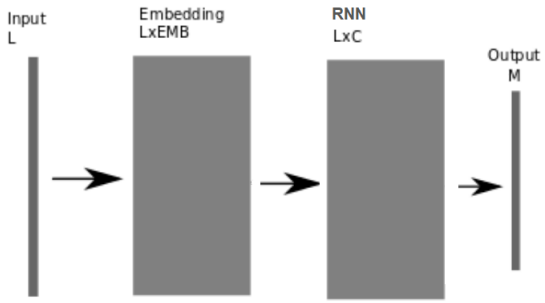


Obrázek 7.2: Architektura konvoluční neuronové sítě.[18]

7.3 LSTM

Rekurentní neuronové sítě se běžně používají pro klasifikaci krátkých textových dokumentů. Pro delší dokumenty byla navržena architektura jejíž, první dvě vrstvy jsou stejné jako u CNN architektury. Ve třetí vrstvě je ovšem matice $L \times EMB$ vstupem pro LSTM vrstvu. Ta nám poté vrací výstup pro každou vstupní sekvenci, neboli řádek matice $L \times EMB$ a tím získáváme výstupní matici $L \times C$ kde C je počet buněk LSTM vrstvy. Matice poté slouží jako vstup pro výstupní fully-connected vrstu a získáváme

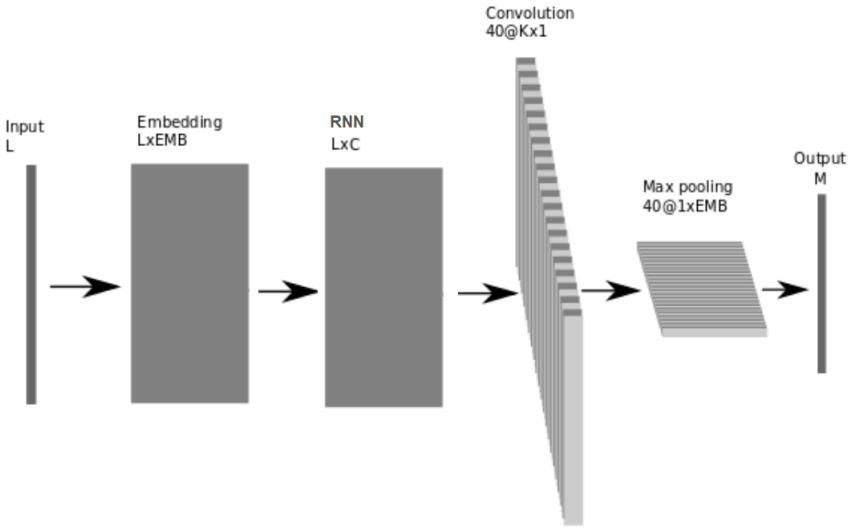
výstup o velikosti M, což je počet klasifikovaných kategorií.



Obrázek 7.3: Architektura konvoluční neuronové sítě.[18]

7.4 RNN + CNN

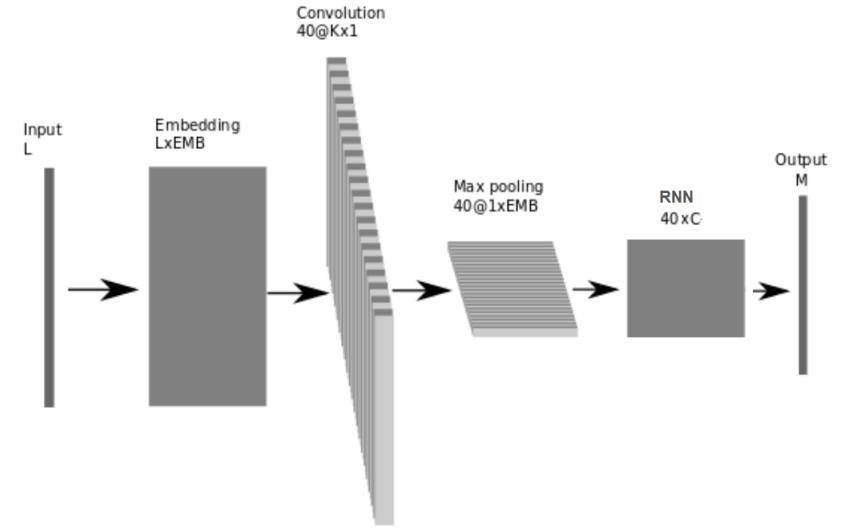
Vzhledem k vlastnosti dobrého uchování souvislostí vrstvy LSTM, bylo této skutečnosti využito při návrhu RNN + CNN architektury. Kdy rekurentní síť se stará o převod matice $L \times EMB$ na $L \times C$ od čehož si slibujeme vytvoření návaznosti mezi jednotlivými textovými reprezentacemi. Matice $L \times C$ poté slouží jako vstupní vrstva pro konvoluční neuronovou síť. U této architektury opět platí, že vstupem je vektor velikosti L předzpracovaný pomocí integer encoding a výstup má velikost M, která je rovna počtu klasifikovaných kategorií. Za RNN vsrtvu, vzhledem ke stejným vstupním parametrům, lze použít jak LSTM, tak GRU.



Obrázek 7.4: Architektura konvoluční neuronové sítě.[18]

7.5 CNN + RNN

Rekurentní vrstvu je však možné použít i pro zlepšení příznaků získaných z konvoluční vrstvy pomocí max poolingu. Vezmeme tedy CNN architekturu, kterou zakončíme max poolingem s výstupem $N_c 1 \times EMB$, kde N_c je počet použitých konvolučních filtr. Vektor získaný max poolingem bude rozdělen na sekvence po jednotlivých řádcích $N_c \times EMB$. Pro každou sekvenci získáme výstup z rekurentní vrstvy. Tyto výstupy následně vytvoří matici $N_c \times C$, kde C je počet LSTM nebo GRU buněk. Matice je už poté pouze klasifikována na počet kategorií M výstupní fully-connected vrstvou.



Obrázek 7.5: Architektura konvoluční neuronové sítě.[18]

8 Implementace neuronových sítí

Využívání neuronových sítí ve strojovém učení se stalo velmi populárním. Je tedy k dispozici velký výběr knihoven v různých programovacích jazycích. Ve velké oblibě je při implementaci jazyk C++ a Python. Nechybí ovšem ani implementace v Matlabu nebo Javě. Na vyvíjení knihoven se podílejí velké firmy jako například Google s knihovnou TensorFlow [22], Microsoft s Microsoft Cognitive Toolkit (CNTK) [7] nebo Facebook a jeho knihovna PyTorch [20] nebo Caffe2 [6]. Existuje ale také velké množství knihoven, které vznikly jako open source projekt například Theano [23]. Výběr je tedy velmi bohatý, proto začaly vznikat projekty, kde je použito rozhraní, ve kterém je možné napsat jeden kód a spustit ho poté s různými knihovnami na pozadí. Momentálně nejpoužívanější rozhraní tohoto typu je Keras [10].

8.1 Keras

Keras [10] je vysokoúrovňové rozhraní pro neuronové sítě v jazyku Python. Jeho hlavní výhodou je, že se nestará o implementaci neuronových sítí jako takovou, ale vytváří přehledné a dobře dokumentované rozhraní pro snadné navrhování a testování různých architektur neuronových sítí. Implementaci poté obstarávají momentálně 3 dostupné knihovny Theano [23], TensorFlow [22] a CNTK [7]. Všechny tyto implementace mají podporu pro konvoluční neuronové sítě a zároveň také podporují urychlení výpočtu počítáním na grafické kartě. Keras navíc na vývoji spolupracuje s výrobci výše zmíněných knihoven a dále k jeho práci přispívají i firmy jako NVIDIA a Apple. Na jeho internetových stránkách [10], lze nalézt také podrobný návod jak ho nainstalovat, nastavit, a také několik zkušebních příkladů.

8.1.1 TensorFlow

TensorFlow byl původně vyvíjen týmem Google pro firemní použití, ovšem v roce 2015 byl vydán pod Apache 2.0 open source licencí. Výpočetně složité funkce jsou napsány v programovacím jazyce C++. Pro urychlení výpočtů je také možné využít funkcí naprogramovaných v jazyku CUDA pro výpočet na grafické kartě. Všechny tyto funkce jsou obaleny pro využití v jazyku Python.

CUDA

Computed Unified Device Architecture neboli CUDA [8] je hardwarová a softwarová architektura, která umožňuje na vybraných GPU spouštět programy napsané v jazycích C/C++, FORTRAN nebo programy postavené na technologiích OpenCL. Použití této architektury je omezeno pouze na grafické akcelerátory společnosti NVIDIA. Přehled o tom jaké GPU CUDU podporují a v jaké míře lze nalézt na stránkách firmy NVIDIA [16].

8.2 Metacentrum

Projekt Metacentrum vznikl v roce 1996, od roku 1999 je součástí aktivit sdružení CESNET. Od svého vzniku se zabývá budováním národního Gridu. Součástí MetaCentra je virtuální organizace MetaCentrum VO, která je otevřená všem akademickým pracovníkům, zaměstnancům a studentům vědeckovýzkumných institucí v České Republice. Vzhledem k vypočetní náročnosti konvolučních sítí budeme právě těchto služeb využívat.

8.2.1 Přihlášení

Pro využití služeb je nutné podat přihlášku, která je většinou schválena do druhého dne. U institucí, které využívají Českou akademickou federaci identit eduID.cz, což je většina univerzit v České Republice, je přihláška schválena mnohem rychleji. Pro podání přihlášky je nutné znát údaje pro přístup ke svému univerzitnímu účtu, který je MetaCentrem ověřen a data z něj jsou poté v přihlášce použita.

8.2.2 Žádost o výpočetní úlohu

Pro přihlášení na servery MetaCentra je potřeba terminál s podporou protokolu SSH. Uživatelům systému Windows je doporučeno používat PuTTY [19] a pro správu dat program s FTP protokolem WinSCP [24]. Oba programy jsou šířeny pod open-source licencí. V PuTTY terminálu se poté údaji z dřívější registrace přihlásíme na některý z čelních(frontedových) uzlů. Seznam frontendů je k dispozici na wiki [9].

Parametry úlohy

Na čelním uzlu si zažádáme o úlohu pomocí příkazu qsub, jenž je následován řadou parametrů, které jsou odděleny přepínačem -l.

```
qsub
-l select=1:ncpus=2:ngpus=1:gpu_cap=cuda35:mem=5gb
-l walltime=1:00:00
-l option1
-l option2
script.sh
```

Listing 8.1: Vzorová specifikace parametrů příkazu qsub.

Parametr select slouží pro volbu zdrojů, které vyžadujeme. Můžeme zde určit počet procesorů, grafických karet a velikost RAM. Dále zde můžeme zvolit užší výběr výpočetních strojů podle specifických vlastností. Parametr gpu_cap=cuda35 omezuje výběr na stroje které podporují výpočetní instrukce CUDY na úrovni 3 viz odkaz [16]. Instrukce třetí úrovně, jsou potřeba, pro počítání na GPU v prostředí Keras. V případě požadavku o GPU přidáváme na konec parametr -q gpu.

Dalším parametrem je walltime, tento parametr definuje maximální dobu trvání úlohy a zapisován je ve formátu hh:mm:ss.

Dávkové úlohy

Dávkové úlohy jsou neinteraktivní typy úloh. Pro jejich běh je potřeba připravit spouštěcí skript, odeslat úlohu, čekat na její dokončení a poté si prohlédnout výsledky. Pro tvorbu skriptu se používá Bash. V hlavičce dávkového souboru lze také definovat všechny potřebné parametry pro úlohu.

Interaktivní úlohy

Interaktivní úloha je speciální typ úlohy dávkové, kde je možné přímo pracovat s přidělenými zdroji místo používání dávkových skriptů. O interaktivní úlohu se žádá pomocí přepínače -I viz následující příklad.

```
qsub
-I
-l select=1:ncpus=2:ngpus=1:gpu_cap=cuda35:mem=5gb
-l walltime=1:00:00
```

Listing 8.2: Vzorová žádost o interaktivní úlohu.

Výhodou interaktivních úloh je podpora GUI. Tento typ úlohy je také velmi užitečný pro ověření korektnosti dávkových souborů. V případě, že se v dávkové úloze nachází chyba, zjistíte to nejdříve až po přiražení úlohy, což může trvat i několik hodin. Lepší je tedy úlohu odladit v interaktivním módu a až poté jí pustit jako dávkovou. Nevýhodou interaktivních úloh je, že v případě odpojení od internetu přijdete i o vaši úlohu. To samé platí i při ukončení terminálu.

8.2.3 Úloha

Při získání úlohy je uživateli přidělen výpočetní server. Ovšem předtím než je možné spustit jakýkoliv kód, je důležité si vše správně nastavit. MetaCentrum přiděluje při získání úlohy všem jednotné prostředí pouze se základní funkčností. Aplikace je poté možné přidat ve formě modulů, jejichž kompletní seznam lze nalézt na wiki [15].

8.2.4 Moduly

Moduly obsahují jednotlivé programy například Python nebo Matlab. Jsou zde i balíčky s knihovnami právě pro Python. Ty ale bohužel většinou neobsahují aktuální verze softwaru, případně v nich některé knihovny chybí. Moduly je možné také vytvářet a poté zažádat MetaCentrum o jejich zveřejnění, což se vyplatí hlavně u týmových projektů. V našem případě je ale tento postup zbytečný a spokojíme se pouze s moduly základními.

8.2.5 Instalace Kerasu

Jelikož MetaCentrum nedovoluje instalovat uživatelům vlastní knihovny na jejich serveru z bezpečnostních a konfliktních důvodů využijeme modulu virtualenv v Pythonu. Tento modul nám vytvoří izolované Pythonovské prostředí, do kterého už pomocí Python package systému (dále jen pip) můžeme instalovat potřebné knihovny.

Postup

Instalace není výpočetně náročná, avšak neměla by se provádět na čelním uzlu, ale v interaktivní úloze. Při přihlášení na čelní uzel se váš domovský adresář nachází na místě, kam byste měli ukládat svá data, toto místo je ve skriptu nahrazeno názvem STORAGE. Pro instalaci izolovaného prostředí využijeme jeden z modulů poskytovaný MetaCentrem, python34-modules-gcc, neboť obsahuje již nainstalovaný pip. Založíme složku do které budeme

prostředí vytvářet. Pomocí pipu nainstalujeme virtualenv, který se bude nacházet v podadresáři adresáře keras. Poté přidáme do systémové proměnné PYTHONPATH složku s nově nacházejícím se Pythonem. Zavoláme knihovnu virtualenv a vytvoříme nové izolované prostředí v adresáři keras. Odpojíme modul s přidanými knihovnami pro Python a přidáme nový, ve kterém se nastaví pouze samotný interpreter jazyka Python. Poté se s jeho využitím přepneme do nově vytvořeného prostředí. Přepíšeme cesty nastavené modulem na nově vzniklé cesty z důvodu, aby nevznikaly kolize knihoven interpreteru. Poté již stačí pouze použít pip k nainstalování potřebných knihoven.

```
1  module add python34-modules-gcc
2
3  DIR=*STORAGE*/keras
4  mkdir -p $DIR
5  cd $DIR
6
7  pip3 install virtualenv --root ./virtualenv
       --process-dependency-links --ignore-installed
8
9  $DIR/virtualenv/software/python-3.4.1/gcc/bin/virtualenv
       keras
10
11 source $DIR/keras/bin/activate
12
13 PYTHONUSERBASE=$DIR/keras/
14 export PATH=$PYTHONUSERBASE/bin:$PATH
15 export PYTHONPATH=$PYTHONUSERBASE/lib/python3.4/site
16 -packages:$PYTHONPATH
17
18 pip install tensorflow-gpu
19 pip install keras
20 pip install h5py
```

Tímto skriptem je instalace Kerasu a Tensorflow dokončena. Nyní už můžeme zažádat o úlohu. Po jejím přiřazení přidáme moduly potřebné pro počítání na grafické kartě tedy CUDU verze 8.0 a CUDNN verze 6.0. Pro přidání CUDNN je potřeba být zaregistrován v NVIDIA Developer Program [17] a tuto registraci potvrdit na stránce MetaCentra [14]. Poté opět přepíšeme cesty modulu interpreteru Pythonu na cesty našeho prostředí. Nyní se už pouze stačí přepnout do složky s našimi skripty a pomocí příkazu python

je spouštět.

```
DIR=*STORAGE*

module add python-3.4.1-gcc cuda-8.0 cudnn-6.0


source $DIR/keras/bin/activate

PYTHONUSERBASE=$DIR/keras2017/
export PATH=$PYTHONUSERBASE/bin:$PATH
export PYTHONPATH=$PYTHONUSERBASE/lib/python3.4/site
-packages:$PYTHONPATH

python *NAME OF SCRIPT*
```

9 Dosažené výsledky

Prvně byly vyhodnoceny sady výsledků pro předzpracovaná data korpusů CDCv2(60) a RCV1-v2(101), kde číslo udané v závorce znázorňuje počet klasifikovaných kategorií. Vzhledem k tomu, že korpus RCV1-v2 obsahuje také kategorie s nízkým zastoupením, nebylo by vhodné je pro multi-label klasifikaci z korpusu CDCv2 odstraňovat, jak je dáno v literatuře [11]. Abychom ale nepřišli o možnost porovnání dosažených výsledků s literaturou, byly také vytvořeny klasifikátory pro CDCv2 (37), kde se vyhodnocuje pouze nejčastějších 37 kategorií.

V tabulce 9.1 vidíme výsledky pro předzpracování pouhou tokenizací textu. Pro české dokumenty dosáhla nejlepších výsledků architektura CNN, která předčila i kombinované architektury rekurentních a konvolučních neuronových sítí, velmi obdobně si také vedla architektura MLP, která dosahavala u nejlepších výsledků pro data anglická. Zajímavé je také porovnání pouze pro reprezentaci pomocí integer encoding. Zatímco u českých dat dosahovala lepších výsledků architektura CNN, pro anglická se projevily jako lepší kombinované architektury konkrétně tedy GRU + CNN .

Tabulka výsledků 9.2 pro lemmatizovaná data nám ukazuje, že pro architektury používající integer encoding nastalo ve většině případů pouze drobné zlepšení jinak byly výsledky přibližně stejné. Ovšem úspěšnost architektury MLP se zvýšila znatelně. Jako příčinu lze považovat to, že lemmatizace snížila počet unikátních slov zhruba na polovinu, je tedy velice pravděpodobné, že předzpracování BOW pokrylo větší rozsah slov v dokumentech. U anglických dokumentů takto výraznou změnu pozorovat nemůžeme. Jedním z důvodů může být to, že lemmatizace nebyla při redukci unikátních slov tak efektivní.

Výsledky stemmingu podle tabulky 9.3 dosahovaly také lepších výsledků než pouhá tokanizace ale zároveň můžeme pozorovat mírné zhoršení oproti lemmatizaci. To může být zapříčiněno tím, že se redukce unikátních slov dotkla také těch, která byla klíčová pro rozlišení některých kategorií.

Celkově nejlepších výsledků dosáhla architektura vícevrstvého perceptronu jak pro české, tak i anglické dokumenty. Vzhledem k její úspěšnosti bylo navržena snaha o následné vylepšení těchto výsledků a to pomocí filtrování takzvaný nevýznamových slov. To jsou slova, která nemají vliv na určení kategorie. Jedná se převážně o spojky a zájmena. Volba těchto slov ovšem velmi záleží na typu kategorií, které chceme klasifikovat. Seznam použitých stop slov je uveden v příloze. Výsledky tohoto experimentu se nacházejí

Architektura	CDCV2 (37)		CDCv2 (60)		RCV1-v2 (101)	
	$F_{1\mu}$	F_{1m}	$F_{1\mu}$	F_{1m}	$F_{1\mu}$	F_{1m}
MLP	**0.834**	**0.835**	**0.876**	**0.800**	**0.810**	**0.620**
CNN	**0.837**	**0.836**	**0.877**	**0.802**	0.755	0.560
LSTM	0.802	0.792	0.851	0.740	0.788	0.508
CNN + LSTM	0.821	0.817	0.859	0.771	0.777	0.552
CNN + GRU	0.832	0.831	0.874	0.799	0.787	0.563
LSTM + CNN	0.779	0.777	0.807	0.706	0.751	0.480
GRU + CNN	0.825	0.822	0.868	0.780	0.797	0.587

Tabulka 9.1: Výsledky jednotlivých architektur pro textové dokumenty předzpracované pouhou **tokenizací**.

Architektura	CDCV2 (37)		CDCv2 (60)		RCV1-v2 (101)	
	$F_{1\mu}$	F_{1m}	$F_{1\mu}$	F_{1m}	$F_{1\mu}$	F_{1m}
MLP	**0.853**	**0.855**	**0.849**	**0.823**	**0.813**	**0.625**
CNN	0.842	0.844	0.838	0.806	0.751	0.550
LSTM	0.814	0.809	0.806	0.742	0.782	0.505
CNN + LSTM	0.822	0.822	0.817	0.755	0.776	0.541
CNN + GRU	0.829	0.830	0.825	0.793	0.785	0.571
LSTM + CNN	0.781	0.781	0773	0.732	0.769	0.532
GRU + CNN	0.831	0.831	0.827	0.780	0.800	0.581

Tabulka 9.2: Výsledky jednotlivých architektur pro textové dokumenty předzpracované **lemmatizací**.

Architektura	CDCV2 (37)		CDCv2 (60)		RCV1-v2 (101)	
	$F_{1\mu}$	F_{1m}	$F_{1\mu}$	F_{1m}	$F_{1\mu}$	F_{1m}
MLP	**0.852**	**0.854**	**0.848**	**0.819**	**0.811**	**0.629**
CNN	0.842	0.844	0.838	0.806	0.756	0.563
LSTM	0.809	0.805	0.801	0.748	0.782	0.510
CNN + LSTM	0.821	0.817	0.815	0.771	0.776	0.546
CNN + GRU	0.834	0.835	0.830	0.798	0.784	0.559
LSTM + CNN	0.779	0.777	0.772	0.726	0.765	0.522
GRU + CNN	0.829	0.832	0.824	0.780	0.792	0.575

Tabulka 9.3: Výsledky jednotlivých architektur pro textové dokumenty předzpracované **stemmingem**.

v tabulce 9.4. Z výsledků vyplývá, že tato filtrace byla efektivní při zlepšení klasifikace u předzpracování tokenizací. Nepatrného vylepšení došlo také při vyhodnocení pro lemmatizaci. Naopak u stemmingu dopadly výsledky hůře.

Metoda předzpracování	**CDCv2 (37) filtrace nevýznamových slov**		**CDCv2 (37)**	
	$F_{1\mu}$	F_{1m}	$F_{1\mu}$	F_{1m}
Tokenizace	0.838	0.839	0.834	0.835
Lemmatizace	**0.854**	**0.855**	**0.853**	**0.855**
Stematizace	0.849	0.853	0.852	0.854

Tabulka 9.4: Vysledky filtrace nevýznamových slov.

Porovnáním dosažených výsledků pro český a anglický jazyk můžeme pozorovat, že pro český jazyk bylo dosaženo lepších klasifikačních výsledků. Zde je ale potřeba vzít v úvahu rozdíly mezi korpusy. Anglický korpus má o 41 víc kategorií, což se značně projeví na hodnotách macro-averaged F–míry a zároveň obsahuje 781265 dokumentů určených pro testování. Zatímco u jazyka českého máme celkově k dispozici textových dokumentů pouze 11955. Dalším zajímavým rozdílem je dominance architektury CNN nad složitějšími návrhy architektur. U anglických dat totiž k takovému jevu nedochází. Můžeme také říci, že metody předzpracování mají pro oba jazyky stejný vliv. Předzpracováním pomocí lemmatizace bylo v obou jazycích dosaženo nejlepších výsledků. Následovala metoda stemmingu a nejhůře dopadla obyčejná tokenizace. Toto tvrzení vyvrací pouze výsledky CDCv2 (60) z tabulky 9.1, kde tokenizovaná verze dosáhla naopak výsledků značně převyšujících ostatní. Zde je ale potřeba uvést, že u předzpracovaného textu pomocí lemmatizace a stemmingu byly naopak vyšší honoty macro-averaged F–míry. Tento výsledek je pravděpodobně zapříčiněn nevhodnou volbou prahu p, která činí výsledky těžko porovnatelnými.

Celkovým zhodnocením výsledků klasifikace je porovnání s literaturou, u korpusu CDCv2 nám k tomu slouží tabulka 9.5. Nejlepších výsledků dosáhla architektura MLP s předzpracováním pomocí lemmatizace a filtrováním nevýznamových slov. Pro testovanou architekturu CNN dopadly výsledky
o něco hůře. U anglických dat a korpusu RCV1-v2 bylo dle literatury [13] dosaženo nejlepších výsledků při klasifikace pouze 101 kategorií typu Topics klasifikátorem Support Vector Machine s micro-averaged F–mírou 0.816 a macro-averaged 0.619. Při klasifikaci pomocí neuronových sítí bylo architekturou MLP dosaženo výsledku 0.813 micro-averaged F–míry a 0.629 macro-averaged. Celkově bylo tedy dosaženo horších výsledků celkové klasi-

fikace, ale klasifikátor si průměrně vedl lépe pro určení všech typů kategorií. Poměr těchto hodnot je velmi ovlivněn právě volbou prahu p.

Architektura	**CDCv2 (37) dosažené $F_{1\mu}$**	**CDCv2 (37) publikované $F_{1\mu}$**
Vícevrstvý pereceptron (MLP)	0.854	0.839
CNN	0.842	0.847

Tabulka 9.5: Porovnání dosažených výsledků na korpusu CDCv2 s literaturou.

10 Závěr

Tato práce se věnovala automatické klasifikaci textových dokumentů pomocí neuronových sítí. Cílem tohoto experimentu bylo prostudovat vliv různých metod předzpracování na výsledky klasifikace.

Prvním úkolem bylo se seznámit s knihovnami pro implementaci neuronových sítí. Na vybranou knihovnu bylo kladeno několik kriterií. Hlavním kriteriem, vzhledem k výpočetní náročnosti úlohy, byla možnost využití této knihovny na gridové infrastruktuře MetaCentra. Této podmínce vyhovovala nejlépe knihovna Keras napsána pro jazyk Python. Jedním z důvodů této volby je také dostupná, velmi přehledná a dobře strukturovaná dokumentace této knihovny.

Dále bylo potřeba najít vhodné nástroje pro lemmatizaci a stematizaci dokumentů. Jelikož záměrem práce je získané výsledky ověřit pro český i anglický jazyk, bylo nutné najít takové nástroje, které budou mít podporu pro oba. Proto byly vybrány programy UDPipe, který realizuje tokenizaci s lemmatizaci a následně High Precision Stemmer pro použitý stemming. Tyto nástroje byly použity pro předzpracování dokumentů z datových korpusů CDCv2 a RCV1v2.

S připravenými daty bylo potřeba najít, navrhnout, nebo přizpůsobit vhodné architektury neuronových sítí pro řešení problému multi–label klasifikace. Ty bylo potřeba následně implementovat, natrénovat a vyhodnotit dosažené výsledky pro jednotlivé typy předzpracování.

Dosažené výsledky byly následně porovnány s literaturou, kde pro český korpus bylo dosaženo lepších celkových výsledků použitím lemmatizace, filtrace nevýznamových slov a architektury MLP. Pro jazyk anglický byla opět nejúspěšnější architektura MLP, která dosáhla výsledků pro micro-averaged F–míru sice horších o $0,003$, ale pro macro-averaged naopak lepších o $0,01$.

Budoucí vylepšení by mohlo přinést důkladné prostudování vlivu volby prahu p na výsledky klasifikace. Jeho volba totiž velmi ovlivňuje poměr micro a macro-average statistických metod. Také by pro reprezentaci pomocí BOW mohla být navržena metoda filtrování nevýznamových slov na základě POS-taggingu.

Literatura

[1] *UDpipe github* [online]. Dostupné z: https://github.com/ufal/udpipe.

[2] *UDpipe* [online]. Dostupné z: https://github.com/ufal/udpipehttp://ufal.mff.cuni.cz/udpipe/install.

[3] *UDpipe online* [online]. Dostupné z: http://lindat.mff.cuni.cz/services/udpipe/run.php.

[4] *Convolutional Neural Networks for Visual Rocognition* [online]. Dostupné z: http://cs231n.github.io/convolutional-networks/#overview.

[5] Brychcín, T. – Konopík, M. HPS: High precision stemmer. *Information Processing & Management*. January 2015, 51, 1, s. 68 – 91. ISSN 0306-4573. doi: http://dx.doi.org/10.1016/j.ipm.2014.08.006. Dostupné z: http://www.sciencedirect.com/science/article/pii/S0306457314000843.

[6] *Caffe2* [online]. Google, 2018. [cit. 2018/01/17]. Dostupné z: https://caffe2.ai/.

[7] *Microsoft Cognitive Toolkit (CNTK)* [online]. Microsoft, 2018. [cit. 2018/01/17]. Dostupné z: https://github.com/Microsoft/CNTK.

[8] *NVIDIA CUDA* [online]. NVIDIA. [cit. 2018/01/17]. Dostupné z: http://docs.nvidia.com/cuda/index.html#axzz3wagwJWX1.

[9] *Frontendové uzly MetaCentrum* [online]. MetaCentrum. [cit. 2018/01/17]. Dostupné z: https://wiki.metacentrum.cz/wiki/%C4%8Celn%C3%AD_uzel.

[10] *Keras* [online]. Keras, 2018. [cit. 2018/01/17]. Dostupné z: https://keras.io/.

[11] Král, P. – Lenc, L. Czech Text Document Corpus v 2.0. 2018.

[12] Král, P. – Lenc, L. Deep Neural Networks for Czech Multi-label Document Classification. 2017.

[13] Lewis, D. et al. RCV1: A New Benchmark Collection for Text Categorization Research. *The Journal of Machine Learning Research*. 1 2004, 5, 12, s. 361 – 397. ISSN 1532-4435. Dostupné z: https://dl.acm.org/citation.cfm?id=1005345.

[14] *MetaCentrum* [online]. CESNET. [cit. 2018/01/18]. Dostupné z: https://perun.metacentrum.cz/meta/registrar/?locale=cs&vo=meta&group=lic_cudnn.

[15] *MetaCentrumModules* [online]. MetaCentrum. [cit. 2018/01/18]. Dostupné z: https://wiki.metacentrum.cz/wiki/Kategorie:Applications.

[16] *NVIDIA CUDA GPUS* [online]. NVIDIA. [cit. 2018/01/17]. Dostupné z: https://developer.nvidia.com/cuda-gpus.

[17] *Nvidia Developer Program* [online]. NVIDIA. [cit. 2018/01/18]. Dostupné z: https://developer.nvidia.com/rdp/assets/cudnn-65-eula-asset.

[18] Olah, C. *Understanding LSTM Networks* [online]. Dostupné z: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[19] *PuTTY* [online]. PuTTY. [cit. 2018/01/17]. Dostupné z: http://www.putty.org/.

[20] *PyTorch* [online]. Facebook. [cit. 2018/01/17]. Dostupné z: http://pytorch.org/.

[21] Straka, M. – Straková, J. Tokenizing,POS Tagging, Lemmatizing and Parsing UD 2.0 with UDPipe. 2017.

[22] *TensorFlow* [online]. Google, 2018. [cit. 2018/01/17]. Dostupné z: https://github.com/tensorflow/tensorflow.

[23] *Theano* [online]. Theano, 2018. [cit. 2018/01/17]. Dostupné z: http://deeplearning.net/software/theano/.

[24] *WinSCP* [online]. WinSCP.net. [cit. 2018/01/17]. Dostupné z: https://winscp.net/eng/docs/lang:cs.

A Uživatelská dokumentace

Přiložené programy potřebují ke spuštění mít nainstalovaný Python ve verzi 3.6.4 obsahující moduly Keras (2.1.5), Tensorflow-gpu (1.6.0) a h5py(2.7.1). Pro jejich instalaci na vlastním stroji je možné postupovat dle návodu popsaného v sekci 8.2.5. Vzhledem k vydávání nových verzí příslušných knihoven bude s velkou pravděpodobnostní nutné pomocí pipu specifikovat požadovanou verzi knihovny. Skripty byly vytvořeny výhradně pro počítání na grafické kartě s využitím prostředků MetaCentra, v případě snahy spustit tyto skripty na vlastním stroji, je třeba vlastnit grafickou kartu s podporou výpočetních instrukci CUDY úrovně alespoň tři a mít nainstalovanou CUDU (8.0) a výpočetní knihovnu CUDNN verze (6.0). Minimální požadovaná velikost paměti grafické karty jsou 2GB. Zároveň je také třeba splňovat požadavek na velikost paměti RAM, a to 16GB. Skripty jsou rozděleny do dvou kategorií.

A.1 Příprava dat

Příprava dat zajišťuje převod textových dokumentů korpusů CDCv2 a RCV1v2 na vstupní data pro neuronové sítě. Vstupem je tedy adresář s textovými soubory a výstupem je soubor ve formátu h5. Pro tvorbu vstupních dat reprezentovaných pomocí BOW a integer encoding slouží tyto skripty.

CreateDatasetCDCv2BOW – vytváří reprezentaci BOW pro korpus CDCv2

CreateDatasetCDCv2IntEnco – vytváří reprezentaci integer encoding pro korpus CDCv2

CreateDatasetRCV1v2BOWtraining – vytváří reprezentaci BOW trénovacích dat pro korpus RCV1v2

CreateDatasetRCV1v2BOWtestX – vytváří reprezentaci BOW trénovacích dat pro korpus RCV1v2

CreateDatasetRCV1v2Inteenco – vytváří reprezentaci integer encoding pro korpus RCV1v2

Vzhledem k velkému počtu dokumentů korpusu RCV1v2 jsou připraveny 4 programy pro tvorbu testovacích dat reprezentace BOW, kde každý z nich

zpracovává pouze 200000 souborů. I přes toto omezení dosahují soubory h5 velikosti 8GB. Pro vytvoření všech souborů s reprezentací BOW korpusu RCV1v2 je nutné mít vyhrazeno minimálně 31GB volného místa.

Jednotlivé korpusy a způsoby reprezentace mají vytvořeny konfigurační soubory.

CDCv2BOWconfig – konfigurační soubor pro reprezentaci BOW korpusu CDCv2

CDCv2IntEncoconfig – konfigurační soubor pro reprezentaci integer encoding korpusu CDCv2

RCV1v2BOWconfing – konfigurační soubor pro reprezentaci BOW korpusu RC1v2

RCV1v2BOWcoconfig – konfigurační soubor pro reprezentaci integer encoding korpusu RCV1v2

V těchto souborech se nastavují základní parametry pro tvorbu reprezentací.

PATH_DATA – Cesta k textovým dokumentům

PATH_VALIDATION_DATA – Cesta k textovým dokumentům určeným pouze pro validaci

OUT_FILE – Cesta na kterou bude uložen výstupní soubor

DOCUMENT_FORMAT – Formát dokumentu (tok, lemma, stem)

CROSS_VALIDATION_SPLIT – Specifikace počtu částí pro metodu křížové validace

CROSS_VALIDATION_START – Specifikace části použité pro testování.

BAG_OF_WORDS_SIZE – Velikost slovníku pro metodu BOW

VOCABULARY_SIZE – Velikost slovníku pro integer encoding

FINAL_VECTOR_LENGTH – Délka příznakového vektoru pro integer encoding.

STOPWORDS – Použití filtrace nevýznamových slov (True/False)

CATEGORIES – Specifikace kategorií, které mají být při klasifikaci použity.

FILE_NAME – Jméno výstupního souboru s daty

Defaultně je nastavena cesta k datům do složky Databaze. Na CD ale databáze z důvodu jejich velikosti nejsou přiloženy, je tedy nutné si tyto databáze opatřit dle postupu z literatury [11, 13] a poté je do složky zkopírovat. Předpokládaný formát názvu textových dokumentu je $ID\_CAT_1\_CAT_2...\_CAT_x$, kde CAT označuje kategorie, ve kterých se dokumenty nachází.

Po specifikaci parametrů v konfiguračních souborech, doplnění databází a splnění požadavků na knihovny a výpočetní prostředky je možná programy spouštět v příkazové řádce pomocí příkazu python následovaného názvem skriptu.

A.2 Architektury

Složka architektury obsahuje programy s implementací pro natrénování a vyhodnocení neuronové sítě. Nastavení vstupních a výstupních parametrů sítí lze opět specifikovat v konfiguračních souborech.

BOWconfig – konfigurační soubor pro reprezentaci BOW

IntEncoconfig – konfigurační soubor pro reprezentaci integer encoding

Soubory obsahují následující sadu parametrů.

PATH_DATA – Cesta k datům ve formátu h5.

DATA_NAME – Jméno souboru s daty.

TRAIN_DATA_NAME – Jméno h5, která obsahuje pouze trénovací data. Nutné v případě reprezentace BOW u RCV1v2

TEST_X_DATA_NAME – Jméno h5, která obsahuje testovací data. Zde je pro reprezentaci BOW nutné zadat všechny 4 soubory.

OUT_FILE – Cesta kde bude uložen natrénovaný model sítě.

BAG_OF_WORDS_SIZE – Velikost slovníku pro metodu BOW

VOCABULARY_SIZE – Velikost slovníku pro integer encoding

FINAL_VECTOR_LENGTH – Délka příznakového vektoru pro integer encoding.

CATEGORIES – Specifikace kategorií, které mají být při klasifikaci použity.

Po specifikaci parametrů v konfiguračních souborech je opět programy možné spouštět v příkazové řádce za pomoci příkazu python následovaného názvem skriptu.

B Seznam nevýznamových slov

Zde je uveden seznam slov použitých při filtraci nevýznamových slov z dokumentů.

'v', 'a', 'dnes', 'cz', 'tímto', 'budeš', 'budem', 'byli', 'jseš', 'můj', 'svým', 'ta', 'tomto', 'tohle', 'tyto', 'jej', 'zda', 'proč', 'máte', 'tato', 'kam','tohoto', 'kdo', 'kteří', 'mi', 'nám', 'tom', 'tomuto', 'mít', 'nic', 'proto', 'kterou', 'byla', 'toho', 'protože', 'asi', 'ho', 'naši', 'napište', 'což', 'tím', 'takže', 'svých', 'jeji', 'svými', 'jste', 'aj', 'tu', 'tedy', 'této', 'bylo', 'kde', 'ke', 'právě', 'ji', 'nad', 'nejsou', 'čí', 'pod', 'těma', 'mezi', 'přes', 'ty', 'pak', 'vám', 'ani', 'když', 'však', 'ne', 'jsem', 'tento', 'článku', 'články', 'aby', 'jsme', 'před', 'ptá', 'jejich', 'byl', 'ještě', 'až', 'bez', 'také', 'pouze', 'první', 'vaše', 'která', 'náš', 'nový', 'tipy', 'pokud', 'může', 'strana', 'jeho', 'své', 'jiné', 'zprávy', 'nové', 'není', 'vás', 'jen', 'podle', 'zde', 'článek', 'už', 'email', 'být', 'více', 'bude', 'již', 'než', 'který', 'by', 'které', 'co', 'nebo', 'ten', 'tak', 'má', 'při', 'od', 'po', 'jsou', 'jak', 'další', 'ale', 'si', 've', 'to', 'jako', 'za', 'zpět', 'ze', 'do', 'pro', 'je', 'na','se', 'z', 'o', 'že', 'i', 's'

C Statistické výsledky

Vzhledem k velkému množství výsledků získaných pro jednotlivé architektury a metody předzpracování byly ve výsledcích uvedeny pouze hodnoty F–míry. Z toho důvodu byl na CD přidán soubor, který obsahuje veškeré statistické informace o natrénovaných klasifikátorech použitých v kapitole Dosažené výsledky. Soubor je ve formátu excelovských tabulek, kde každý list nese název vyhodnocené architektury.