TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Vektorová reprezentace slov a její aplikace

Bakalářská práce

Studijní program: B2646 Informační technologie
Studijní obor: Informační technologie

Autor práce: **Martin Halada**
Vedoucí práce: prof. Ing. Jan Nouza, CSc.
Ústav informačních technologií a elektroniky

Liberec 2021

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání bakalářské práce

Vektorová reprezentace slov a její aplikace

Jméno a příjmení: **Martin Halada**
Osobní číslo: M18000074
Studijní program: B2646 Informační technologie
Studijní obor: Informační technologie
Zadávající katedra: Ústav informačních technologií a elektroniky
Akademický rok: **2020/2021**

Zásady pro vypracování:

1. Cílem práce je prakticky se seznámit s vektorovou reprezentací slov založenou na metodě Word2Vec a její variantě FastText.
2. Prostřednictvím literatury se podrobně seznamte s uvedenými metodami a proveďte jejich vlastní (vhodně zdokumentovanou a komentovanou) implementaci, nejlépe v jazyce Python nebo C++.
3. Stáhněte si z internetu co největší objem českých textů (alespoň 1 GB, nejlépe z hlavních zpravodajských serverů), sestavte seznam nejčastějších slov nalézajících se v těchto textech, vytvořte z nich pracovní lexikon obsahující cca 200.000 nejčastějších slov a pro něj vypočítejte vektorovou reprezentaci.
4. S využitím vektorové reprezentace naimplementujte několik ukázkových úloh typu tvorba slovních analogií, vyhledávání podobných/protichůdných slov, analýza sentimentu (kladné/záporné/neutrální vyznění) textu, či zařazení článku do některé z vybraných tematických kategorií. U každé úlohy vždy vyhodnoťte úspěšnost použité metody na vámi připraveném testovacím setu.

Rozsah grafických prací:	Dle potřeby dokumentace
Rozsah pracovní zprávy:	30-40 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

[1] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S.Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. pp 3111–3119.
[2] Yoav Goldberg and Omer Levy 2014. word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. CoRR Vol. abs/1402.3722 (2014).
[3] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Herve Jegou, and Tomas Mikolov. 2016. Fasttext.zip: Compressing text classificationmodels. arXiv preprint arXiv:1612.03651.
[4] Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. Bagof tricks for efficient text classification. arXiv preprint arXiv:1607.01759, 2016

Vedoucí práce:	prof. Ing. Jan Nouza, CSc. Ústav informačních technologií a elektroniky
Datum zadání práce:	19. října 2020
Předpokládaný termín odevzdání:	17. května 2021

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

prof. Ing. Ondřej Novák, CSc.
vedoucí ústavu

V Liberci dne 19. října 2020

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

11. května 2021 Martin Halada

Vektorová reprezentace slov a její aplikace

Abstrakt

Bakalářská práce se zabývá vektorovou reprezentací slov založenou na metodě Word2Vec. V první kapitole představuje problematiku reprezentace slov pomocí vektorů, zmiňuje různé způsoby a varianty jejího učení a nastiňuje možnosti jejího praktického využití. Druhá kapitola se zabývá vlastní implementací zmíněné metody, uvádí postup přípravy dat, vytvoření pracovního lexikonu a trénování neuronové sítě. Třetí kapitola se věnuje pěti možným aplikacím, jako jsou hledání podobných resp. opačných slov, tvorba slovních analogií, analýza sentimentu a zařazování článků do vybraných kategorií. První tři aplikace pracují přímo s natrénovanými vektory, neboť využívají informace o pozicích slov ve vektorovém prostoru. Další dvě aplikace používají vektory jako vstup do neuronové sítě natrénované jako klasifikátor do vybraných tříd. U všech úloh byl experimentálně vyhodnocován vliv několika základních parametrů (velikost slovníku, dimenze vektorového prostoru a délka kontextového okna) na úspěšnost. Nejdůležitějším parametrem byla velikost pracovního lexikonu, menší roli mělo nastavení počtu slov v kontextovém okně.

Klíčová slova: Vektorová reprezentace slov, Word2Vec, CBOW, skip-gram, negative sampling, neuronové sítě, aplikace, analýza sentimentu, analogie, kategorizace textů

Vector representation of words and its applications

Abstract

The bachelor thesis deals with vector representation of words based on Word2Vec method. The first chapter presents the issue of representation words using vectors, mentions different ways and variants of its learning and outlines the possibilities of its practical use. The second chapter deals with the actual implementation of the mentioned method, describes the procedure of data preparation, creation of a lexicon and training neural network. The third chapter deals with five possible applications, such as searching for similar and opposite words, creation of verbal analogies, sentiment analysis and classification of articles into selected categories. First three applications work directly with trained vectors, because they use information from word positions in vector space. The other two applications use vectors as input into a neural network trained as classifier into selected classes. On all tasks were tested the influence of several basic parameters (dictionary size, dimension of vector space and length of context window) on success. The most important parameter was the size of the dictionary, the setting of the number of words in the context window had a smaller role.

Keywords: Vector representation of words, Word2Vec, CBOW, skip-gram, negative sampling, neural networks, applications, sentiment analysis, analogy, text categorisation

Obsah

Úvod **12**

1 Vektorová reprezentace slov **13**

1.1 Praktické využití . . . 14
1.2 Word2Vec . . . 14
1.2.1 CBOW . . . 16
1.2.2 Skip-gram . . . 17
1.2.3 Hierarchická softmax funkce . . . 19
1.2.4 Negative sampling . . . 19
1.2.5 Skip-gram with negative sampling . . . 20
1.3 Varianty Word2Vec . . . 21
1.3.1 FastText . . . 21
1.3.2 GloVe . . . 22
1.3.3 Doc2Vec . . . 22

2 Implementace metody Word2Vec **23**

2.1 Příprava dat . . . 23
2.2 Vytvoření pracovního lexikonu . . . 27
2.3 Trénování neuronové sítě . . . 31

3 Praktické aplikace **35**

3.1 Podobná slova . . . 35
3.2 Protichůdná slova . . . 37
3.3 Analogie . . . 39
3.4 Analýza sentimentu . . . 41
3.5 Kategorizace . . . 45

Závěr **49**

Literatura **52**

A Přílohy **56**

Seznam tabulek

2.1 Počet stažených dat ze zpravodajských portálů 23
2.2 Statistika stažených dat 24
2.3 Nejčastěji se vyskytující slova v trénovacím souboru 30
2.4 Nejčastěji se vyskytující významová slova 30
2.5 Závislost OOV rate na velikosti pracovního lexikonu 31
2.6 Natrénované modely neuronové sítě 34

3.1 Podobná slova, porovnání různých modelů 36
3.2 Opačná slova, porovnání různých modelů 38
3.3 Analogie, porovnání různých modelů 40
3.4 Statistika dat pro analýzu sentimentu 41
3.5 Porovnání úspěšnosti - analýza sentimentu 44
3.6 Porovnání ztrátové funkce - analýza sentimentu 45
3.7 Statistika dat pro zařazení článku do kategorie 45
3.8 Porovnání úspěšnosti - kategorie článků 48
3.9 Porovnání ztrátové funkce - kategorie článků 48

Seznam obrázků

1.1 Vektorový prostor o dimenzi 2 15
1.2 Vizualizace kontextového okna architektury CBOW o délce 5 16
1.3 Schéma architektury CBOW 17
1.4 Vizualizace kontextového okna architektury skip-gram o délce 5 . . . 18
1.5 Schéma architektury skip-gram 18
1.6 Binární strom využívaný hierarchickou softmax funkcí 19
1.7 Schéma metody FastText 22

3.1 Rozdělení dat pro analýzu sentimentu 42
3.2 Analýza sentimentu - úspěšnost nejlepšího modelu 43
3.3 Analýza sentimentu - ztrátová funkce nejlepšího modelu 44
3.4 Kategorie článků - úspěšnost nejlepšího modelu 47
3.5 Kategorie článků - ztrátová funkce nejlepšího modelu 47

Seznam pseudokódů

2.1 Kód pro stažení odkazů z iDNES.cz . . . 25
2.2 Kód pro stažení textů z iDNES.cz . . . 26
2.3 Kód pro spojení souborů . . . 26
2.4 Kód pro vyfiltrování souboru . . . 27
2.5 Struktura pro uložení slova . . . 27
2.6 Metoda pro vytvoření slovníku . . . 28
2.7 Metoda pro přidávání slov do slovníku . . . 28
2.8 Metoda pro zmenšení slovníku . . . 29
2.9 Metoda pro uložení slovníku . . . 29
2.10 Příprava neuronové sítě . . . 32
2.11 Pseudokód pro trénování sítě . . . 33
3.1 Algoritmus hledání podobných slov . . . 35
3.2 Algoritmus hledání protichůdných slov . . . 37
3.3 Algoritmus tvorby slovních analogií . . . 39

Seznam zkratek

CBOW	Continuous Bag of Words
GloVe	Global vectors
LSTM	Long short-term memory
NCE	Noise Contrastive Estimation
NLP	Natural language processing
OOV	Out-Of-Vocabulary
RNN	Recurrent neural network
SGD	Stochastic gradient descent

Úvod

Strojové učení a neuronové sítě, za posledních deset let, zažily rychlý vývoj. Tyto zajímavé a rostoucí obory jsou důležité a užitečné při řešení celé řady problémů. Jednou z těchto oblastí je zpracování přirozeného jazyka, kterou také často nalezneme pod označením NLP (z angličtiny Natural language processing).

Její poznatky a techniky jsou využitelné zejména v počítačové lingvistice, protože dokážou částečně porozumět přirozenému jazyku. Umožňují například analyzovat, případně i generovat texty a slova. Větší podniky mohou využívat tyto techniky například pro vyhodnocování rozsáhlých dotazníků spokojenosti a jiných dokumentů, ve kterých jim jejich zákazníci poskytují zpětnou vazbu. I běžný uživatel internetu se také může setkat s NLP, například při vyhledávání dokumentů a webových stránek s podobným obsahem, nebo například při sledování videí, kde je možné si zapnout automaticky vygenerované titulky.

Cílem této práce je seznámit se s vektorovou reprezentací slov založenou na metodě Word2Vec, případně její novější variantě FastText. Pomocí této techniky, která reprezentuje slova unikátním vektorem, vytvořit vlastní implementaci ve vybraném programovacím jazyce. Vzhledem k potřebné rychlosti zpracování výpočtů je doporučeno využít jazyk C++ nebo Python. Práce si dále klade za cíl stáhnout z internetu rozsáhlý objem českých textů ze zpravodajských webů. Z těchto dat následně sestavit slovník, který bude obsahovat nejčastěji se vyskytující slova ze stažených dat a s pomocí takto vytvořeného pracovního lexikonu vypočítat vektorovou reprezentaci pro tato slova.

Ve třetí části je cílem si vyzkoušet implementaci řady ukázkových aplikací a demonstrovat využití vektorové reprezentace slov. Mezi uvedené aplikace můžeme zařadit vyhledávání podobných a protichůdných slov, dále tvorbu slovních analogií, analýzu sentimentu a zařazování textů do několika kategorií. Nezbytnou součástí bakalářské práce je, pro výše zmíněné úlohy, připravit testovací dataset a tyto aplikace řádně otestovat. Pro některé aplikace, jako je analýza sentimentu a kategorizace textů, bude třeba ještě stáhnout a připravit nová trénovací data, ve kterých budou jednotlivé věty správně označkovány.

Vzhledem k tomu, že se tento druh technologií rychle vyvíjí, neexistují v českém jazyce zatím všechny termíny, které souvisejí s danou problematikou. Proto jsou v práci uváděny i anglické výrazy, které jsou ale dále pro pochopení významu textu lépe vysvětleny.

1 Vektorová reprezentace slov

Vektorová reprezentace slov [1], nebo také vnoření slov (z anglického výrazu word embedding), je postup číselného vyjádření slov, které se využívá pro aplikace zpracování přirozeného jazyka (NLP).

Vektorová reprezentace je řada technik [2], kde jsou jednotlivá slova vyjádřena jako vektor s hodnotou v předdefinovaném vektorovém prostoru. Vektory slov jsou naučeny z různých textů, kde je brán zřetel na použití slov a jejich závislosti. To má za následek, že slova s podobným významem mají i podobnou reprezentaci. Vektory mají různě velké dimenze, často desítky až stovky hodnot. Vyšší dimenze dokážou zachytit více vztahů mezi slovy, ale vyžadují více dat pro učení. Existuje řada implementací, například GloVe (Global Vectors), Word2Vec, FastText a další varianty.

Jádrem této práce bude se seznámit s metodou Word2Vec, která byla publikována v roce 2013 týmem vědců [3] v čele s Tomášem Mikolovem.

Tomáš Mikolov, který je původem z České republiky, se kromě této metody podílel i na jejich variantách, konkrétně novější techniku FastText [4] a další metody. FastText je nadstavbou původní metody, doplněná o některé nové poznatky - viz kapitola 1.3 zaměřená na varianty Word2Vec.

Při zpracování textů je nutné vyřešit problematiku samotného převádění slov na číselné hodnoty, neboli vektorizaci. Zde existuje několik strategií jak tento problém vyřešit.

První možností [5] je reprezentovat slova pomocí kódu 1 z n. To znamená, že každé slovo bude vyjádřeno unikátním vektorem, který bude obsahovat na všech pozicích nulu a na pozici indexu tohoto slova ze slovníku bude hodnota jedna. Tato strategie je ovšem velice neefektivní, protože počet dimenzí vektoru bude roven počtu slov ve slovníku. Navíc tento vektor obsahuje, již zmíněné, samé nuly kromě jedné hodnoty (řídký vektor) a tím se plýtvá paměťovým prostorem.

Druhou možností [5] je zakódovat každé slovo unikátním číslem. Například podle indexů slov uložených ve slovníku. Ačkoliv je tento přístup efektivnější, protože neplýtvá pamětí, má řadu nevýhod. Toto kódování nezachycuje žádné vztahy mezi ostatními slovy. Druhým problémem je náročnost interpretace pro model. Protože zde neexistují vazby mezi slovy, tak klasifikátor nemá možnost naučit matice vah.

Třetí, nejpoužívanější, možností je vektorizace slov [5] (word embedding). Jedná se o efektivní způsob reprezentace, podobná slova jsou kódována podobným způsobem. Oproti výše uvedeným metodám toto kódování neprovádíme manuálně [6], protože to jsou trénované parametry (váhy). Váhy se učí během fáze trénování a hodnoty vektorů jsou reálná čísla.

1.1 Praktické využití

Vektorová reprezentace slov nachází využití pro celou řadu aplikací [7], zejména v oblasti zpracování přirozeného jazyka (NLP). Používá se jednak jako vstup do modelů strojového učení, kdy je cílem zjistit číselné vyjádření slov a natrénovat model, a poté jako reprezentace nebo vizualizování různých vzorů podle použití v textech na kterých byly vektory natrénovány.

V oblasti zpracování přirozeného jazyka je často nutné využívat vektory pro syntaktickou analýzu [9] a následné jazykové modelování. Mezi některé praktické aplikace vektorové reprezentace slov lze zařadit například automatický překlad mezi 2 různými jazyky. Například Google Translator [10] spolu se službou Microsoft Translator jsou příklady NLP modelů, které pomáhají při překladu. Dále se NLP zabývá například analyzováním vět ve smyslu gramatiky a syntaxe a aplikace následně upozorňují uživatele na případné problémy v jejich textech. Dále slouží pro experimentování při tvorbě slovních analogií. Jiným příkladem využití může být například postupné predikování slov do vět. Tato služba bývá implementována ve vyhledávačích nebo v emailových klientech.

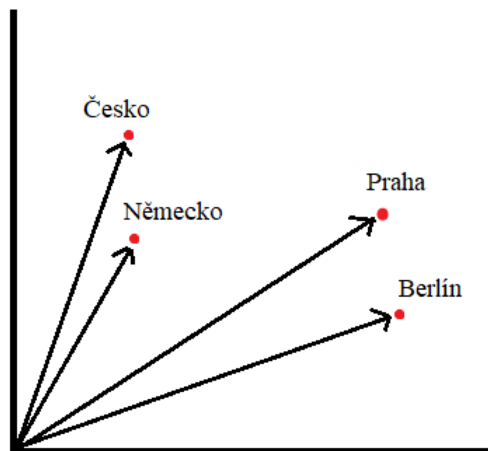
Velké firmy často potřebují analyzovat různé dotazníky, recenze a další rozsáhlé dokumenty, za účelem vyhodnocení spokojenosti zákazníků. Tato oblast bývá označována jako analýza sentimentu. Mezi nejznámější programy, které tento princip využívají, lze například zařadit Hubspot's Service Hub [8]. Uživatelé se setkávají s programy, které využívají vektorovou reprezentaci slov, například při zapnutí automaticky vygenerovaných titulků u videí na některých platformách. Vyhledávače umožňují uživatelům využít rozšířené vyhledávání a možnost najít a setřídit další dokumenty/weby podle podobnosti. Užitečnou vlastností některých webových portálů jsou doporučovací systémy pro audiovizuální materiály.

1.2 Word2Vec

Word2Vec je označení pro soubor technik [11], případně model, který je určen pro vytvoření vektorové reprezentace slov, často označované termínem vnoření slov. V anglické literatuře se zde setkáváme s označením „word embedding". Vnořování slov obecně využívá myšlenku, že každé slovo lze reprezentovat pomocí unikátního vektoru, jehož hodnoty popisují informace o tom, jak je slovo odlišné od jiných - viz obr. č. 1.1.

Pro následné určení podobnosti mezi vektory vypočítáme kosinovou podobnost [12]. Ta nezohledňuje velikosti, úhly vystihují vektory s velkým počtem dimenzí lépe než vzdálenosti. Čím je tento úhel menší, tím je podobnost vyšší:

$$cos(\theta) = \frac{A \cdot B}{||A|| \cdot ||B||} = \frac{\sum\limits_{i=1}^{n} AiBi}{\sqrt{\sum\limits_{i=1}^{n} Ai^2}\sqrt{\sum\limits_{i=1}^{n} Bi^2}} \tag{1.1}$$



Obrázek 1.1: Vektorový prostor o dimenzi 2

Kosinové podobnosti se často mylně říká kosinová vzdálenost. Vzdálenost je funkce, která by měla splňovat trojúhelníkovou nerovnost, symetričnost a její hodnota by měla být větší nebo rovna nule. Proto je vhodné ji nazývat termínem kosinová míra [13], nebo kosinová podobnost. Mezi hlavní výhody této míry patří, že je normalizovaná mezi hodnotami nula a jedna. Jako druhou přednost můžeme považovat, že při vykreslování kosinová míra zachycuje směr.

Využívání vektorů s velkým počtem dimenzí je výhodné pro uložení většího množství informací [11], ovšem tato interpretace je pro uživatele příliš abstraktní a při vysokém počtu dimenzí je obtížné vektory vizualizovat. Výhodou je ale to, že kosinová podobnost platí pro jakýkoliv dimenzionální prostor.

Vektorovou reprezentaci slova získáme tak, že zjišťujeme, jaká jiná slova se s ním poblíž často vyskytují [11]. Celý proces lze tedy shrnout do několika bodů. Nejprve je důležité stáhnout rozsáhlý soubor textových dat, nejlépe složený ze článků z různých oblastí. Je ideální si vybrat takovou oblast, která bude blízká budoucím aplikacím. Nebývá zde příliš vhodné se zaměřovat pouze na jednu oblast, například stahovat články jen ze sportovní rubriky, protože slovní zásoba obsažena v těchto textech nebude tak bohatá a bude úzce vázaná jen na tuto oblast. Nejlepším řešením je tedy vybrat rovnoměrně data z různých celků. Dále je potřeba si zvolit délku kontextového okna, s nímž procházíme veškerý trénovací soubor. Toto okno („sliding window") nám generuje dataset (trénovací páry), který využijeme dále pro trénování modelu.

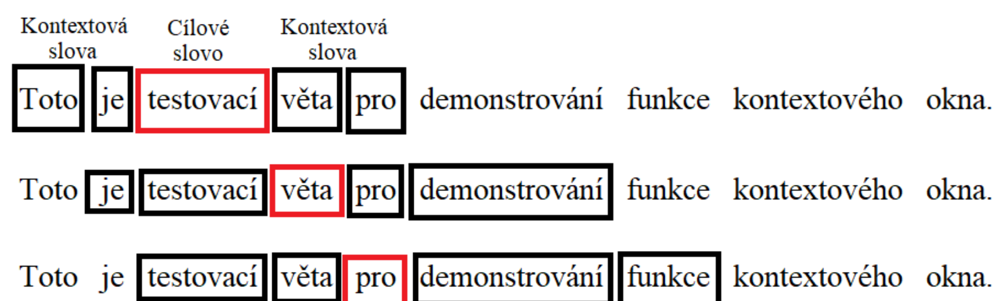
S kontextovým oknem [11] lze pracovat dvěma způsoby. Ty jsou popsány v následujících podkapitolách (CBOW a skip-gram). Délka kontextového okna je jeden z hyperparametrů procesu trénování a jeho nastavení ovlivňuje výsledek. Obecně je doporučováno využívat délku okna v rozmezí 2-15 slov pro aplikace, u nichž jsou příbuzná slova vzájemně zaměnitelná. Například slova dobrý a špatný, protože jsou často používána v podobném kontextu. Naopak větší délku okna 15-50 slov je dobré využít v případě, že je v aplikacích důležité vyjádřit příbuznost slov například jako synonyma.

Model neuronové sítě metody Word2Vec se skládá ze tří vrstev [18] (vstupní, jedné skryté, výstupní vrstvy). Vstupní vrstva se skládá z tolika neuronů, kolik je slov ve slovníku. Skrytá vrstva obsahuje méně neuronů, než vstupní vrstva. Pro velká data to bývá zpravidla pár stovek neuronů (nejčastěji 100-300). Zároveň velikost skryté vrstvy vyjadřuje počet dimenzí výsledných vektorů. Poslední, třetí, je vrstva výstupní s aktivační funkcí softmax. Mezi těmito vrstvami jsou matice vah. Matice vah mezi vstupní a skrytou vrstvou po dokončení procesu trénování obsahuje výsledné vektory slov, které dále využíváme pro další aplikace. Matice vah mezi skrytou a výstupní vrstvou slouží jako kontextová matice a používá se při trénování. Po natrénování se již tato kontextová matice neukládá.

Výpočet Word2Vec lze urychlit řadou technik [14]. Nejdůležitější je vybrání vhodné architektury. CBOW je nejrychlejší, zatímco skip-gram je pomalejší, ale je vhodnější pro zachování méně častých slov. Dále je důležitý výběr algoritmu. Hierarchická softmax se hodí pro méně častá slova, zatímco negative sampling pro často frekventovaná slova. Další optimalizační metoda využívá podvzorkování (subsampling) častých slov. To umožní opět zrychlení výpočtu.

1.2.1 CBOW

První architekturou metody Word2Vec je přístup označovaný zkratkou CBOW (Continuous Bag of Words) [20], který vznikl z původního BOW (Bag of Words, v překladu „batoh slov") rozšířením do prostoru spojitých čísel. Architektura má za cíl predikovat cílové slovo na základě okolních slov v jeho sousedství [22]. To znamená, že s pomocí několika slov nalevo a napravo v kontextovém okně chceme vypočítat prostřední slovo - viz obr. č. 1.2.



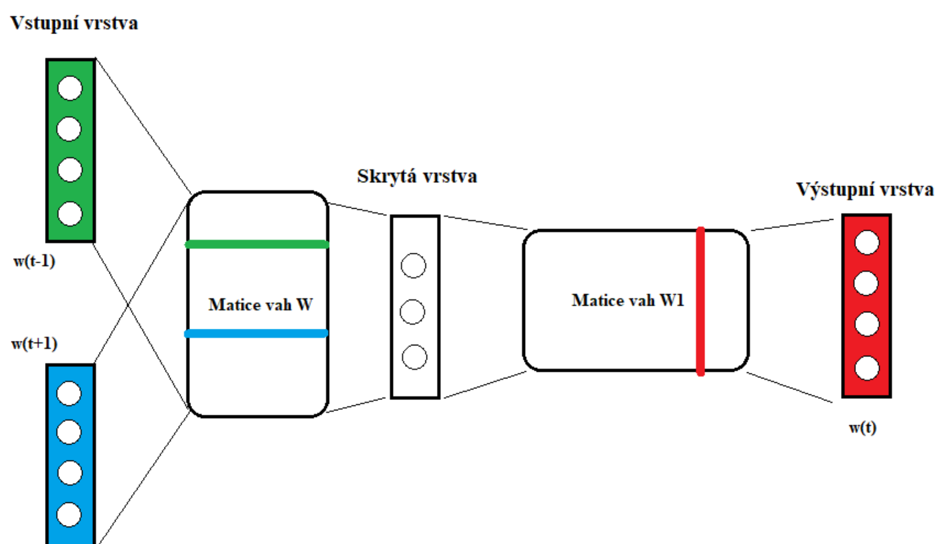
Obrázek 1.2: Vizualizace kontextového okna architektury CBOW o délce 5

Velikost kontextového okna [23], kterou CBOW využívá, je hyperparametr modelu. S větší hodnotou délky okna je umožněno zachytit více závislostí mezi sousedícími slovy. Continuous Bag of Words maximalizuje následující vzorec:

$$\frac{1}{|V|}\sum_{t=1}^{|V|} log[p(wt|wt-c,...,wt-2,wt-1,wt+1,wt+2,...,wt+c)] \tag{1.2}$$

Vstupem do této neuronové sítě jsou postupně všechna kontextová slova [21]. Ve větě to jsou slova nalevo, resp. napravo od požadovaného slova.

Princip této metody je demonstrován na obrázku - viz obr. č. 1.3. Kontextová slova [24], která jsou vstupem do neuronové sítě, jsou zde označena jako *w(t-1)* a *w(t+1)* a jsou reprezentována kódem 1 z n (vektor, který obsahuje 1 hodnotu 1 na správném indexu, zbytek jsou nuly). Pro ně se dále v matici vah W vyhledá jejich vektorová reprezentace. Ta je uložena na tom řádku matice, který je roven indexu slova ze slovníku. Do skryté vrstvy se uloží průměr vektorů všech vstupních slov. Posledním krokem je vynásobení s vektorem cílového (prostředního) slova v kontextovém okně věty, zde je označeno *w(t)*, a aplikování aktivační funkce softmax.



Obrázek 1.3: Schéma architektury CBOW

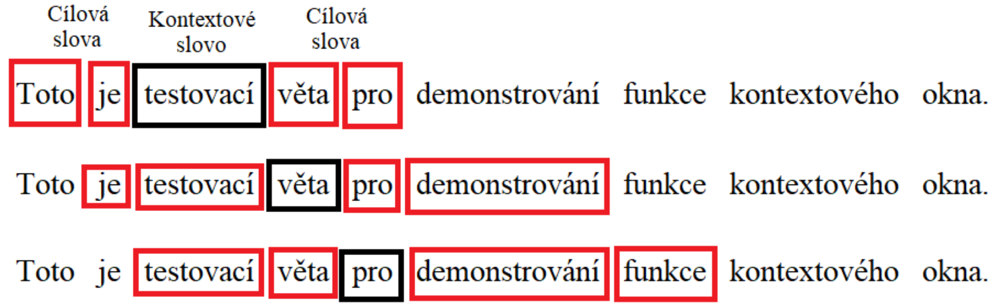
1.2.2 Skip-gram

Druhou architekturou metody Word2Vec je skip-gram [11], který je určitým rozšířením v počítačové lingvistice populárního N-gramu statisticky modelujícího vazby mezi sousedními slovy v textu. U skip-gramu, na rozdíl od N-gramu, nehraje roli pořadí slov, ale pouze přítomnost ve vymezeném okolí. Architektura si klade za cíl predikovat kontext na základě jednoho slova [20]. Tato metoda je opačná oproti technice CBOW. Zde se tedy pro slovo ležící uprostřed kontextového okna snažíme dopočítat slova nacházející se nalevo resp. napravo od něj - viz obr. č. 1.4.

Velikost kontextového okna [27], kterou skip-gram využívá, je hyperparametr modelu. S větší hodnotou délky okna je umožněno zachytit více závislostí mezi sousedícími slovy.

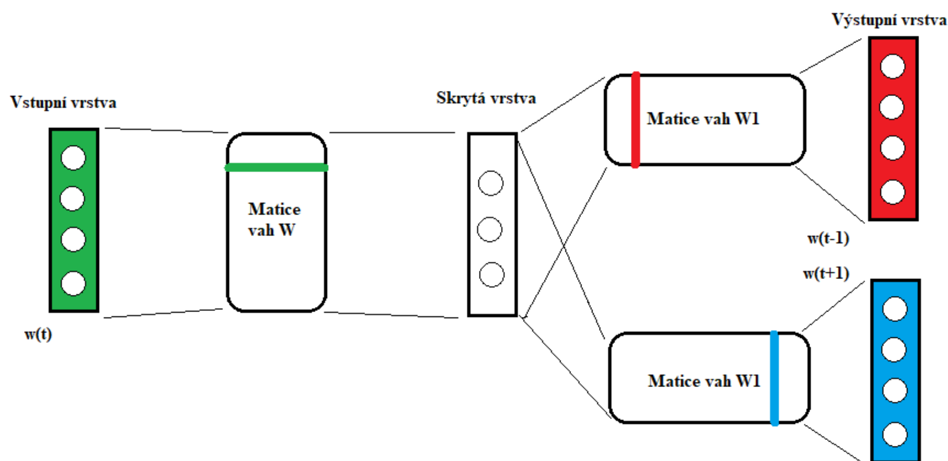
Vstupem do této neuronové sítě je vektor [21], obsahující jednu hodnotu 1 na indexu cílového slova ze slovníku. Ostatní čísla vektoru jsou rovna nule. Ve větě se jedná o slovo ležící uprostřed kontextového okna.

Princip této metody je demonstrován na obrázku - viz obr. č. 1.5. Cílové slovo (uprostřed kontextového okna) [24], které je vstupem do neuronové sítě, je zde označeno jako *w(t)* a je reprezentováno kódem 1 z n (vektor, který obsahuje hodnotu 1 na správném indexu, zbytek jsou nuly). Pro něj se dále v matici vah W vyhledá



Obrázek 1.4: Vizualizace kontextového okna architektury skip-gram o délce 5

jeho vektorová reprezentace. Ta je uložena na tom řádku matice, který je roven indexu slova ze slovníku. Vektor se uloží do skryté vrstvy. Následně se tento vektor postupně vynásobí odděleně s každým vektorem kontextových slov. Ve větě to jsou slova nalevo *w(t-1)*, resp. napravo *w(t+1)* od požadovaného slova. Pro ně se dále v matici vah W1 vyhledá jejich vektorová reprezentace. Vždy tedy pracujeme s páry slov. Posledním krokem je aplikování aktivační funkce softmax.



Obrázek 1.5: Schéma architektury skip-gram

Architektura skip-gram maximalizuje pravděpodobnost správného slova resp. třídy. Pro jeden krok, to znamená jedno aktuální slovo u v čase t, je kritérium následující:

$$L_t(U,V) = \prod_{v \in okoli(u)} P(v|u) = \prod_{v \in okoli(u)} \frac{\exp(u^T v)}{\sum_{w \in slovnik} \exp(u^T w)} \tag{1.3}$$

Protože je Word2Vec logistická regrese, minimalizujeme křížovou entropii:

$$L_{u,v}(U,V) = -u^T v + log \sum_{w \in slovnik} \exp(u^T w) \tag{1.4}$$

Tento postup opakujeme pro každé slovo u z trénovacího korpusu a pro každé v z okolí u.

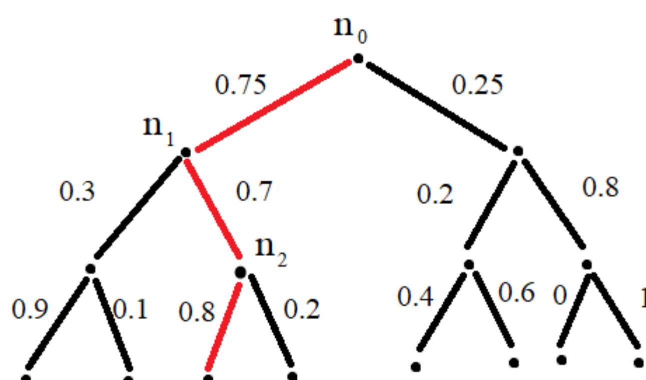
Tento výpočet ovšem představuje velkou nevýhodu kvůli časové náročnosti, protože se v každém kroku vyhodnocuje pro celý slovník. V praxi se tedy většinou využívá optimalizace v podobě záporného vzorkování (negative sampling) - viz kapitola 1.2.4.

1.2.3 Hierarchická softmax funkce

Softmax je aktivační funkce, která se využívá ve výstupní vrstvě neuronové sítě pro klasifikaci do více tříd. Výsledkem klasifikace je třída, pro kterou je funkce softmax nejvyšší:

$$\mathrm{Softmax}(y_i) = \frac{\exp(y_i)}{\sum_j \exp(y_j)} \tag{1.5}$$

Nevýhoda využití softmax funkce je ve výpočetní náročnosti algoritmu [29], protože ji je nutné vypočítat tolikrát, kolik máme slov ve slovníku. Proto se při podobných úlohách zavádí optimalizace v podobě hierarchické softmax funkce [30], která jako datovou strukturu používá binární strom. Díky této technice lze náročnost problému snížit z lineární složitosti na logaritmickou. V této struktuře listy stromu vyjadřují pravděpodobnost slov - viz obr. č. 1.6.



Obrázek 1.6: Binární strom využívaný hierarchickou softmax funkcí

1.2.4 Negative sampling

Metoda negative sampling [14], do českého jazyka lze přeložit jako „výběr negativních vzorků”, je metoda využívaná k optimalizování výpočtu při trénování neuronové sítě. Hlavní myšlenkou je modifikování pouze některých vektorů slov, namísto celé matice vah. Tato technika tedy nepředpovídá pravděpodobnost výskytu všech slov ze slovníku, ale pravděpodobnost toho, jestli některá slova sousedí se zadaným slovem, nebo nesousedí. Technika dále využívá několik náhodně vybraných slov, u kterých víme, že se zadaným slovem často nesousedí. Pro tato slova je cílem, aby výstup ze

sítě byl 0. Výběr slov nemusí být úplně náhodný, ale může být závislý například na frekvenci slov. Termínem pozitivní vzorek lze označit aktuálně zpracovávané slovo, slovním spojením záporný vzorek se označují slova, která byla vybrána tak, aby se s pozitivním vzorkem nevyskytovala v podobném kontextu. Vztah (1.4) se upravuje do následující podoby:

$$L_{u,v}(U,V) = -u^T v + log \sum_{w \in nahodny-vyber} \exp(u^T w) \tag{1.6}$$

Negative sampling [11] představuje alternativu k hierarchické softmax funkci. Obě metody zajišťují rychlejší výpočet, než kdybychom počítali pravděpodobnosti pro všechna slova ve slovníku a následně tyto hodnoty modifikovali. Jedním z hyperparametrů modelu [14] je počet vzorků využívaný metodou negative sampling. Obecným doporučením bývá zvolit celé číslo v rozmezí 5-20 slov pro menší datasety. Ovšem v případě rozsáhlejšího trénovacího souboru stačí zvolit jen 2-5 slov.

Myšlenka metody negative sampling úzce souvisí s odhadem kontrastu šumu [15], v anglické literatuře se setkáváme s termínem Noise Contrastive Estimation (NCE). Model, který odhad šumu využívá, by měl být schopen odlišit data od šumu pomocí logistické regrese. NCE přibližně maximalizuje logaritmickou pravděpodobnost funkce softmax. Tato vlastnost není pro naše účely důležitá a proto této skutečnosti lze využít a NCE zjednodušit do podoby metody negative sampling. Pokud porovnáme původní funkci NCE se záporným vzorkováním, tak NCE potřebuje pro výpočet vzorky společně s distribucí šumu. Negative sampling pro výpočet využívá pouze vzorky.

1.2.5 Skip-gram with negative sampling

V praktické části práce bude využívána architektura označovaná jako skip-gram with negative sampling [11], kterou lze do češtiny přeložit jako skip-gram s výběrem negativních vzorků.

Princip implementace této techniky je následující. Prvním krokem je alokace a inicializace dvou matic vah. První z nich je matice vah mezi vstupní a skrytou vrstvou. V anglické literatuře ji často nalezneme pod termínem „embedding matrix”, protože po dokončení trénování bude obsahovat výsledné vektory. Druhá matice vah leží mezi skrytou a výstupní vrstvou a můžeme ji označit slovním spojením „kontextová matice”. Velikost těchto matic je rovna počtu slov v pracovním lexikonu vynásobeném počtem dimenzí slov. Na začátku jsou naplněny náhodnými hodnotami.

Dále probíhá trénování sítě. Postupně se prochází celý trénovací soubor s pomocí kontextového okna. Z něj přistoupíme na aktuálně zpracovávané slovo a vybereme postupně všechna jeho sousedící slova. Ty považujeme za jeho sousedy a označíme je číslem jedna. Dále následuje aplikace metody negative sampling [31] (záporné vzorkování), kdy vybereme N náhodných slov, různých od slov z kontextového okna (počet takovýchto slov je jedním z hyperparametrů modelu). Tato náhodná slova označíme číslem nula (vyjadřují slova, která se s původním slovem nevyskytují).

Následuje výpočet nového vektoru. Vektor aktuálně zpracovávaného slova (získáme z 1. matice vah) vynásobíme s vektorem jiného slova (získáme z 2. matice vah). Tento výsledek slouží jako vstup do aktivační funkce, která vrací pravděpodobnost. Na základě této hodnoty vyčíslíme chybu, kdy od značky slova (0 nebo 1) odečteme výsledek, který jsme získali z aktivační funkce. A s pomocí chyby zpětně upravíme parametry vah.

Po dokončení procesu trénování pouze uložíme matici vah mezi vstupní a skrytou vrstvou, obsahující natrénované vektory. Mezi hyperparametry modelu patří počet epoch, počet dimenzí výsledných vektorů, délka kontextového okna, počet záporných vzorků.

1.3 Varianty Word2Vec

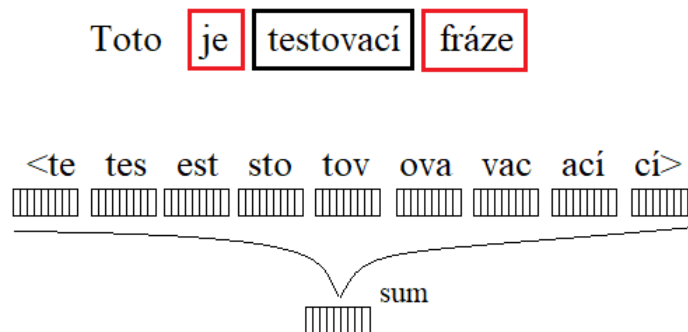
Mezi jednu z nejdůležitějších a nejznámějších implementací vektorové reprezentace slov patří metoda Word2Vec. Nejedná se ovšem o jedinou techniku, jak reprezentovat slova pomocí unikátního vektoru. Z Word2Vec vychází novější a rychlejší varianta označovaná jako FastText [32]. Další alternativou je technika GloVe a za zmínku stojí i zobecnění Word2Vec ve formátu Doc2Vec - viz níže.

1.3.1 FastText

Metoda FastText [33] byla navržena v roce 2016. Tato technika si klade za cíl vyřešit problém se zobecněním na neznámá slova. FastText vychází z Word2Vec a přináší nové algoritmy. Word2Vec vypočítává vektorovou reprezentaci na základě slov v podobném kontextu a nedokáže vytvořit vektory těch slov, které se v procesu trénování nevyskytovaly v trénovacím souboru. FastText oproti tomu zohledňuje jen části slov [34] (znakové N-gramy). Jedná se tedy o větší zobecnění, kdy je možné hledat nová slova, která mají stejný základ, ale například jiné předpony, přípony a koncovky. Díky tomuto principu je možné využít menší množství dat, protože z textu je možné získat větší množství informací.

Algoritmus metody FastText využívá techniku rozdělení slov na znakové N-gramy, často mezi třemi až šesti znaky. Prvním krokem je obalení slova do závorek, pro snazší určení začátku a konce. Dále dojde k vygenerování znakových N-gramů. Například termín „testovací" upravíme na „<testovací>" a 3-gramy budou následující: <te, tes, est, sto, tov, ova, vac, ací, cí> - viz obr. č. 1.7.

Vzhledem k tomu, že unikátních N-gramů může být mnoho a mohou zatěžovat paměť [35], lze program optimalizovat hashováním a dále naučit síť jen pro omezený počet N-gramů. Dále již probíhá proces trénování podobně jako u metody Word2Vec. To znamená, že například pro skip-gram with negative sampling se nejprve vytvoří N-gram centrálního slova v kontextovém okně, popsáno v minulém odstavci. Dále se sečtou vektory těchto N-gramů spolu s vektorem celého slova. S kontextovými slovy se tento proces neprovádí, jen se pracuje s vektory celých slov. Poté se již metoda skip-gram with negative sampling nemění a její princip a postup zůstává stejný jako u techniky Word2Vec, popsáno v minulé kapitole - viz kapitola 1.2.5.



Obrázek 1.7: Schéma metody FastText

1.3.2 GloVe

Jednou z variant Word2Vec, pro natrénovaní vektorové reprezentace slov, je GloVe [14] (Global Vectors), do českého jazyka lze doslovně přeložit jako „globální vektory”. Učení Word2Vec je založeno na závislostech cílových slov s jejich kontextem. Metoda ovšem nerespektuje to, že se některá slova vyskytují častěji než jiná [36]. Oproti tomu GloVe tyto frekvence výskytů slov zdůrazňuje. Kombinace vektorů slov tedy přímo souvisí s pravděpodobností, že se dvě slova vyskytnou společně.

Výhodou varianty GloVe je rychlost trénování [37]. Metoda je škálovatelná pro velké slovníky. GloVe poskytuje dobré výsledky i s menším slovníkem a vektory s menším počtem dimenzí. Na druhou stranu metoda GloVe disponuje i některými omezeními a nevýhodami. Spotřebovává mnoho paměti a je citlivá na počáteční inicializaci parametrů.

1.3.3 Doc2Vec

Doc2Vec [38] je nadstavba metody Word2Vec a její zobecnění. Cílem této techniky je vypočítat vektorovou reprezentaci celých dokumentů, nezávisle na jejich délce. Koncept byl, stejně jako u metody Word2Vec a FastText, navržen Tomášem Mikolovem [39]. Doc2Vec lze použít v aplikacích pro třídění dokumentů resp. článků na základě podobnosti.

2 Implementace metody Word2Vec

Implementace vektorové reprezentace slov se skládá z několika kroků. V první řadě je potřeba stáhnout a upravit rozsáhlý soubor textů, na kterém se bude neuronová síť učit své váhy. Ve druhé části je nutné spolu s upravenými texty vytvořit pracovní lexikon. V poslední části se implementuje vektorová reprezentace slov podle vybrané architektury a využije se pro následné natrénování vektorů slov.

2.1 Příprava dat

Pro vypočítání vektorové reprezentace slov je nejprve nutné stáhnout z internetu co největší objem textů. V rámci této práce byl, z hlavních zpravodajských serverů (Aktuálně.cz, Deník.cz, iDNES.cz a iROZHLAS), vytvořen ze článků trénovací soubor o velikosti 1,08 GB. Z webu Aktuálně.cz bylo celkem staženo 179 MB textů, z Deník.cz 206 MB, z iDNES.cz 710 MB a z iROZHLAS 5 MB - viz tab. č. 2.1. Podrobnější rozložení dat nabízí druhá tabulka - viz tab. č. 2.2.

Tabulka 2.1: Počet stažených dat ze zpravodajských portálů

zpravodajský web	**velikost dat (MB)**	**počet článků**
Aktuálně.cz	179	74.618
Deník.cz	206	154.805
iDNES.cz	710	415.998
iROZHLAS	5	5.929

Z každého zpravodajského webu byly nejprve, pro přehlednost, staženy odkazy na jednotlivé články. Následně byly prováděny požadavky (requesty) na webové stránky s pomocí těchto získaných odkazů a z nich staženy texty do několika souborů. Ty byly následně spojeny do jednoho souboru. Nakonec z něj bylo potřeba vyfiltrovat různé nežádoucí znaky. Všechny programy určené pro zmíněnou přípravu dat (stahování odkazů resp. textů, filtrování) byly napsány v programovacím jazyce Python, protože pro tyto úkoly poskytuje vhodné knihovny.

Prvním krokem je stažení odkazů na články. Jazyk Python nabízí pro provádění HTTP požadavků knihovnu requests. Pro extrahování dat z HTML dokumentů je v Pythonu dostupná knihovna BeautifulSoup. Obě knihovny je potřeba na začátku

Tabulka 2.2: Statistika stažených dat

sledovaná veličina	**hodnota**
celkový počet vět	13.172.758
celkový počet slov	157.261.208
počet různých slov	1.110.824
nejčastěji zastoupená slova	v, a, se, na, že
počet výskytů nejčastějších slov	v = 3.866.550 a = 3.801.129 se = 3.252.218 na = 3.055.797 že = 1.783.182

každého podobného programu vždy naimportovat - viz pseudokód č. 2.1. Na dalším řádku je vytvořeno prázdné pole, které slouží pro uložení odkazů. Na řádku č. 8 začíná cyklus, ve kterém je definován rozsah prohledávání. Ideální je získávat odkazy na články v pořadí od nejstaršího po nejnovější. Pokud je na web přidán nový článek, všechny ostatní se o jeden posunou, a mohlo by se stát, že by odkazy na některé články byly duplicitní. Při tomto prohledávání tento problém nenastane. Na dalších řádcích vygenerujeme URL, ze kterého budeme získávat odkazy a provedeme HTTP dotaz metodou GET. Nakonec vytvoříme objekt BeautifulSoup, díky kterému zde můžeme přistupovat k elementům jazyka HTML. Na řádcích č. 12-15 uvádíme, z jakých elementů chceme získat data. V tomto případě si ukládáme všechny odkazy, se třídou s názvem „w230”, ohraničené elementem „div”, jehož identifikátor je roven „list—art—count”.

Je nutné si uvědomit, že každý zpravodajský server má jiné rozložení elementů a může mít i různě pojmenované třídy a identifikátory. Proto je vždy před získáváním dat z webů nutné ručně projít strukturu webové stránky a vybrat si vhodné elementy. Kód je proto vždy potřeba trochu upravit. Nakonec nalezené odkazy uložíme do předem vytvořeného pole a tato data lze již uložit do souboru, kde je každý řádek vyhrazen pro jeden odkaz.

```
1 Vstup: url_main - odkaz na hlavní web
2 Výstup: url_out - odkazy na články
3
4 import requests
5 from bs4 import BeautifulSoup
6
7 url_out = []
8 foreach url in url_main
9     r = request(url)
10    content = r.text
11    soup = BeautifulSoup(content, "html.parser")
12    main_content = soup.find("div",
13                       attrs = {'id': 'list-art-count'})
14    odkazy = main_content.findAll('a',
15                 attrs = {'class': 'w230'})
16    foreach odkaz in odkazy
17        url_out.append(odkaz["href"])
```

Pseudokód 2.1: Kód pro stažení odkazů z iDNES.cz

Dalším krokem je stažení textů s využitím předem uložených odkazů - viz pseudokód č. 2.2. Nejprve jsou zde opět importovány knihovny pro práci s HTTP metodami a pro přistupování k HTML elementům. Poté přečteme celý soubor s odkazy do předem připraveného pole. Dále následuje cyklus, ve kterém procházíme všechny odkazy. Průběžně můžeme uživatele informovat o postupu.

S každým odkazem provedeme HTTP dotaz a získáme obsah HTML dokumentu. Některé články, například z webu iDNES.cz, jsou placené a musíme je přeskočit, protože jejich texty nejsou při tomto prohledávání přístupné. Tyto články jsou přeskočeny, neobsahují-li v HTML dokumentu script s atributem „free". Na řádku č. 15-17 si ukládáme text, který je uložen v elementu odstavce, dále ohraničený elementem „div" s atributem id, který je roven „art-text". Nakonec získaný obsah můžeme uložit do souboru s kódováním utf-8.

```
1 Vstup: odkazy - odkazy na články
2 Výstup: texty - soubor s texty
3
4 import requests
5 from bs4 import BeautifulSoup
6
7 odkazy = file.read().split("/n")
8
9 foreach url in odkazy
10     r = request(url)
11     content = r.text
12     soup = BeautifulSoup(content, "html.parser")
13     articleScript = soup.find('script')
14     if "free" in articleScript:
15         main_content = soup.find('div',
16                                  attrs = {'id': 'art-text'})
17         odstavce = main_content.findAll("p")
18         texty.append(odstavce)
```

Pseudokód 2.2: Kód pro stažení textů z iDNES.cz

Texty byly ukládány průběžně do několika menších souborů, podle zpravodajského portálu a rubriky. Posléze byly spojeny v jeden soubor - viz pseudokód č. 2.3. Python nabízí knihovnu „os" pro práci se souborovým systémem. Tato knihovna nám zde slouží pro načtení všech souborů textového formátu do předem vytvořeného pole. To je patrné z řádků č. 6-7. Dále si vytvoříme finální soubor a tam postupně nakopírujeme obsahy všech načtených souborů.

```
1 Vstup: soubory
2 Výstup: vystupni_soubor - jeden soubor
3
4 import os
5
6 path = os.path.dirname(os.path.realpath(__file__))
7 files = [f for f in os.listdir(path) if f.endswith('.txt')]
8 for soubor in soubory:
9     text = soubor.read()
10    vystupni_soubor.append(text)
```

Pseudokód 2.3: Kód pro spojení souborů

Poslední, důležitou, součástí je vyfiltrování různých nežádoucích znaků - viz pseudokód č. 2.4. Na začátku programu si připravíme soubor, do kterého budeme ukládat výsledek a soubor, ze kterého budeme načítat. Bezprostředně poté procházíme ze vstupního souboru řádek po řádku a provádíme úpravu. Na dalším řádku převádíme všechna slova na malá písmena. Dále je potřeba odstranit všechny znaky, které

specifikujeme. Jsou to například: čísla, závorky, čárky, uvozovky, lomítka, procenta a další. Dále je vhodné odstranit všechny „bílé znaky". Posledním krokem je nahrazení znaků konce věty jedním speciálním znakem. To při dalších aplikacích zajistí rychlejší zpracování. V této práci budeme považovat za konec věty znak tečky, otazníku, vykřičníku, dvojtečky a středníku. Také budeme ignorovat slova resp. znaky, které nesplňují kódování utf-8. V posledním kroku uložíme výsledek do souboru.

```
1 Vstup: soubor
2 Výstup: vyfiltrovany_soubor
3
4 texty = soubor.read()
5 foreach radek in texty
6     radek = radek.lower()
7     radek = radek.replace(nezadouci_znaky, "␣")
8     if "." in radek
9         radek.replace(konec_vety, "␣0␣")
10    vyfiltrovany_soubor.append(radek)
```

Pseudokód 2.4: Kód pro vyfiltrování souboru

2.2 Vytvoření pracovního lexikonu

Dalším krokem je vytvoření pracovního lexikonu, který bude obsahovat nejčastěji se vyskytující slova z trénovacího souboru. Třída a metody pro vytvoření slovníku byly napsány v programovacím jazyce C++ z důvodu vyšší rychlosti při provádění různých operací nad daty.

Nejprve byla vytvořena struktura „slovoVeSlovniku", do které je možné vložit slovo včetně počtu jeho výskytů v trénovacím souboru - viz pseudokód č. 2.5.

```
1 struct slovoVeSlovniku {
2         long long pocetVyskytu = 0;
3         string text;
4 };
```

Pseudokód 2.5: Struktura pro uložení slova

Dále byla vytvořena třída „Slovnik". Ta prostřednictvím svých veřejných metod umožňuje vytvořit slovník, zmenšit ho v případě nízkého počtu výskytů některého slova, zjistit počet všech slov a nakonec uložit slovník do souboru.

Metoda, která zajišťuje vytvoření slovníku, se jmenuje „vytvorSlovnik" - viz pseudokód č. 2.6. Zde se postupně prochází celý trénovací soubor, v každém cyklu se přečte jedno slovo a to se následně použije jako parametr funkce, která jej přidá do slovníku.

```
1 Vstup: trenovaci_soubor
2
3 texty = trenovaci_soubor.read()
4 foreach slovo in texty
5     pridejSlovoDoSlovniku(slovo)
```

Pseudokód 2.6: Metoda pro vytvoření slovníku

Pro přidávání slov do slovníku je vytvořena funkce „pridejSlovoDoSlovniku” - viz pseudokód č. 2.7. Parametry této metody jsou: vkládané slovo, počet jeho výskytů a index (poslední dva parametry využijeme při načítání již vytvořeného slovníku ze souboru). Do slovníku neukládáme slovo „0”, které vyjadřuje konec věty. Pro všechna ostatní slova procházíme, pomocí iterátoru, nesetříděnou mapu a hledáme výskyt vstupního slova. Pokud se v mapě ještě nevyskytuje, přidáme jej do mapy a nastavíme počet jeho výskytů na hodnotu 1. V případě, že načítáme slovník ze souboru, využijeme dále pole textových řetězců, díky němuž můžeme slova vybírat pomocí indexů. Tato strategie nám umožní dále náhodně vybírat slova ze seznamu. Užitečné je to zejména pro metodu negative sampling - viz další kapitola. Pokud se v mapě slovo již vyskytuje, inkrementujeme pouze u příslušného slova počet jeho výskytů.

Ukládání slov do mapy je výhodné, protože můžeme rychle zjišťovat přítomnost slova bez nutnosti implementovat hashování, které by mohlo generovat kolize. Na druhou stranu budeme dále potřebovat i indexování při náhodném vybírání slov, proto jsou zde slova uložena takto duplicitně. Toto řešení tedy umožní provádět rychlé operace, ale za cenu většího zaplnění paměti.

```
1 Vstup: slovo, mapaSlov
2 Výstup: mapaSlov, vektorSlov – aktualizovaná mapa slov
3
4 if slovo == "0"
5     return
6
7 jeNalezeno = mapaSlov.find(slovo)
8 if jeNalezeno is False
9     mapaSlov.insert(slovo, 1)
10    vektorSlov.append(slovo)
11 else
12    pocet_vyskytu = mapaSlov(slovo).pocet
13    mapaSlov(slovo).pocet = pocet_vyskytu++
```

Pseudokód 2.7: Metoda pro přidávání slov do slovníku

V každém trénovacím souboru se mohou vyskytovat různé překlepy a další málo používaná slova, která je vhodné vyfiltrovat. Tento nedostatek řeší metoda „zmensiSlovnik” - viz pseudokód č. 2.8. Tato funkce prochází postupně celou mapu pomocí iterátoru a v případě, že se zde nachází slovo, jehož počet výskytů je menší než je

hodnota proměnné „minPocet”, je z mapy odstraněno. Nakonec je uživateli zobrazena informace o celkovém počtu zachovaných slov. V rámci této práce byla proměnná „minPocet” nejprve nastavena na hodnotu 14, ve slovníku nakonec zůstalo přibližně 202 tisíc slov. Poté byly vytvořeny ještě další slovníky s různou délkou - viz níže.

```
1 Vstup: slovnik, min_pocet_vyskytu
2 Výstup: slovnik − aktualizovaný slovník
3
4 foreach slovo in slovnik
5     if slovo.pocet < min_pocet_vyskytu
6         slovnik.erase(slovo)
```

Pseudokód 2.8: Metoda pro zmenšení slovníku

Uložení mapy slov do souboru řeší funkce „ulozSlovnik” - viz pseudokód č. 2.9. Ta postupně prochází, díky iterátoru, celou mapu a do textového souboru je na každý řádek uložen nejprve klíč (zde je klíčem slovo jako textový řetězec) a následně hodnota (zde je hodnotou slovo a počet výskytů). Na řádku jsou položky odděleny jednou mezerou.

```
1 Vstup: slovnik
2 Výstup: soubor
3
4 foreach slovo in slovnik
5     pocet_vyskytu = slovo.pocet
6     soubor.append(slovo, pocet_vyskytu)
```

Pseudokód 2.9: Metoda pro uložení slovníku

Funkce, která načítá již jednou vytvořený slovník do paměti, má název „nactiSlovnik”. Postupně se zde prochází soubor se slovníkem a z každého řádku se přečte klíč a hodnota (klíčem je samotné slovo, hodnotou je slovo a počet jeho výskytů). Dále je nutné převést textový řetězec „pocet” na číselnou hodnotu. Nakonec se zavolá metoda „pridejSlovoDoSlovniku”, jejíž parametry jsou slovo, počet výskytů a index.

Další důležitou funkcí je zjištění počtu všech slov v trénovacím souboru. Postupně zde procházíme trénovací soubor a sčítáme všechna načítaná slova. Znak „0”, který reprezentuje konec věty nepočítáme. V rámci této práce bylo v trénovacím souboru nalezeno 157.261.208 slov. Další statistiky o počtu výskytů jednotlivých slov jsou uvedeny v následujících dvou tabulkách. První tabulka informuje o deseti nejčastěji zastoupených slovech - viz tab. č. 2.3. V textech se nejvíce objevují funkční slova, která syntakticky a gramaticky propojují významová slova, a to pomocí předložek, spojek a příslovcí, zatímco významová slova nesou skutečný význam a jsou vyjádřena podstatnými a přídavnými jmény, zájmeny, číslovkami a slovesy - viz tab. č. 2.4. Podle výsledků je zřejmé, že data byla stažena ze zpravodajských webů, neboť se zde často vyskytují slova: „korun”, „procent”, „strany”. Každé z nich se v textech vyskytlo více než 100 tis.

Tabulka 2.3: Nejčastěji se vyskytující slova v trénovacím souboru

Slovo	**Počet výskytů**
v	3.861.281
a	3.795.920
se	3.251.718
na	3.055.581
že	1.782.971
je	1.707.914
to	1.550.454
s	1.264.585
z	1.193.807
o	1.134.006

Tabulka 2.4: Nejčastěji se vyskytující významová slova

Slovo	**Počet výskytů**
roce	162.897
lidí	162.705
korun	158.383
dnes	140.810
procent	130.682
tři	127.799
dva	126.014
české	114.846
říká	106.343
strany	103.169

Dále bylo sestaveno několik slovníků o velikosti 108.000, 148.000, 202.000 a 340.000 slov - viz tab. č. 2.5. Pro každý pracovní lexikon byla spočítána hodnota, kterou označujeme jako „Out-Of-Vocabulary rate” (OOV rate), do českého jazyka lze přeložit jako „podíl slov mimo slovník”. Pro lexikon s přibližně 108.000 slovy byla vypočítána hodnota kolem 3 %. Naopak pro největší slovník podíl vyšel přibližně 0,7 %. Také můžeme zjistit, že u nejmenšího slovníku chybí téměř 4,6 mil. slov. Mnoho slov ve větách tedy bylo vypuštěno a tím mohou být vypočtené závislosti mezi slovy zkreslené. Tyto závislosti poté mohou mít značný vliv na úspěšnosti některých aplikací.

Tabulka 2.5: Závislost OOV rate na velikosti pracovního lexikonu

Počet unikátních slov [tis.]	**Min. počet výskytů**	**Počet OOV slov [mil.]**	**OOV rate [%]**
108	45	4,6	2,94
148	25	3,28	2,09
202	14	2	1,47
340	5	1,16	0,74

2.3 Trénování neuronové sítě

Další část bakalářské práce se zabývá natrénováním neuronové sítě. Ta obsahuje vstupní vrstvu, kterou reprezentuje vektor o velikosti pracovního lexikonu. V rámci této práce byly trénovány vektory s využitím slovníků s délkou přibližně 100 tis., 200 tis. a 340 tis. slov. Dále se síť skládá ze skryté vrstvy, tu vyjadřuje vektor o velikosti nejčastěji 100-300 neuronů. V bakalářské práci byly trénovány vektory s 50 a 100 neurony. Poslední vrstva je výstupní, která má stejnou velikost jako vstupní vrstva.

Hlavním zdrojem inspirace byl komentovaný kód [40], napsaný v programovacím jazyce C, od Tomáše Mikolova a dále podobná implementace v jazycích C++ a Python [41], opět vycházející z původního programu od Mikolova.

V první části bylo nutné si alokovat matice vah. Nejprve mezi vstupní a skrytou vrstvou, ta je označena jako W. Poté mezi skrytou a výstupní vrstvou, ta je označena jako W1. Matici W nastavíme na náhodné reálné hodnoty z intervalu -1 a 1, matici W1 vynulujeme - viz pseudokód č. 2.10. Každá matice vah tvoří pole o velikosti: počet slov × počet dimenzí. Hodnoty jsou uloženy v jedno rozměrném poli, protože se jedná o rychlejší a efektivnější způsob, než při využití dvourozměrných polí.

```
1 Vstup: slovnik, pocet_neuronu
2 Výstup: W, W1 - matice vah
3
4 pocet_slov = slovnik.size()
5 W = float[pocet_slov*pocet_neuronu]
6 W1 = float[pocet_slov*pocet_neuronu]
7
8 i=0
9 while i < pocet_slov*pocet_neuronu
10      nahodne_cislo = rand()
11      W[i] = nahodne_cislo
12      W1[i] = 0
13      i++
```

Pseudokód 2.10: Příprava neuronové sítě

Dále bylo potřeba vytvořit metodu pro čtení jednotlivých vět ze souboru a ty poskytnout pro další zpracování. Věty jsou v trénovacím souboru odděleny znakem „0". Postupně se prochází tento soubor a jednotlivá slova jsou ukládána do pole reprezentující větu.

Hlavním jádrem programu je metoda pro natrénování neuronové sítě - viz pseudokód č. 2.11. Vstupem do metody je trénovací soubor, délka kontextového okna, počet příznaků vektorů a počet vzorků pro metodu negative sampling. Výstupem je matice vah mezi vstupní a skrytou vrstvou W.

Program probíhá v cyklu, který je ukončen po průchodu celým souborem. Celý proces se opakuje podle toho, kolik si zvolíme epoch (počet průchodů celým trénovacím souborem). U podobně rozsáhlých trénovacích souborů stačí zvolit 5-10 epoch, zde bylo zvoleno 5 epoch. Nejprve se přečte nová věta ze souboru a uloží se do předem připraveného pole. Poté se postupně prochází všechna slova z načtené věty a pro každé z nich se projdou všechna jeho sousedící slova, která jsou od něj ve vzdálenosti zadané uživatelem (délka kontextového okna). Do skryté vrstvy, neu1, bude uložena reprezentace tohoto centrálního slova. Dále je v práci využitá metoda negative sampling. V cyklu postupně vybíráme ze slovníku náhodná slova, nevyskytující se v kontextovém okně. Zadané slovo považujeme za pozitivní vzorek, všechna ostatní za záporný vzorek. Následně sčítáme příznaky slova uložené ve skryté vrstvě s příznaky slova uložené v matici vah mezi skrytou a výstupní vrstvou W1. Na základě tohoto součtu vyhodnotíme gradient. Ten vynásobíme s vektorovou reprezentací slova uložené v matici W1 a získáme vektor reprezentující chybu. Následuje zpětná propagace chyby na matici vah W1 a následně aktualizace i matice vah W.

```
1 Vstup: trenovaci_soubor, okno, dimenze, pocet_vzorku
2 Výstup: W
3
4 while není konec souboru
5     veta[] = prectiVetu(trenovaci_soubor)
6     foreach cilove_slovo ∈ veta
7         foreach slovo ∈ kontextove_okno(cilove_slovo, okno)
8             for d=0 to dimenze do neu1[d]+= W[slovo][d]
9         foreach vzorek ∈ pocet_vzorku
10            if vzorek == cilove_slovo then hodnota = true
11            else hodnota = false
12            for d=0 to dimenze do f+=neu1[d]+W1[vzorek][d]
13            g = gradient(f, hodnota)
14            for d=0 to dimenze do neu1e[d]+=g*W1[vzorek][d]
15            for d=0 to dimenze do W1[vzorek][d]+=g*neu1[d]
16        foreach slovo ∈ kontextove_okno(cilove_slovo, okno)
17            for d=0 to dimenze do W[slovo][d]+=neu1e[d]
```

Pseudokód 2.11: Pseudokód pro trénování sítě

Nakonec jsou natrénované vektory uloženy do souboru. Ukládáme pouze matici vah mezi vstupní a skrytou vrstvou. Druhou matici, mezi skrytou a výstupní vrstvou, již dále potřebovat nebudeme.

Následně bylo natrénováno celkem 9 modelů s různými hyperparametry (velikost slovníku, počet dimenzí vektorů a délka kontextového okna), počet epoch byl nastaven na hodnotu 5 - viz tab. č. 2.6. Vektory byly trénovány na počítači s procesorem AMD Ryzen 5 2600X a doba výpočtu se u jednotlivých modelů značně lišila, od 20 do 55 minut. Nejvíce ovlivňuje dobu počet dimenzí vektoru a délka kontextového okna. Počet epoch také prodlužuje dobu trénování, proto byly v bakalářské práci natrénovány sítě vždy s 5 epochami. Často se i v literatuře setkáváme s názory, že při využití rozsáhlejších trénovacích dat (>1GB) stačí zvolit nižší počet epoch.

Tabulka 2.6: Natrénované modely neuronové sítě

Označení	**Velikost slovníku [tis.]**	**Počet dimenzí**	**Délka okna**	**Doba trénování [min]**
model1	108	50	10	21
model2	108	50	20	36
model3	108	100	10	32
model4	108	100	20	52
model5	202	50	10	23
model6	202	50	20	38
model7	202	100	10	32
model8	202	100	20	54
model9	340	100	10	34

3 Praktické aplikace

Natrénované vektory slov lze využít k celé řadě užitečných aplikací. Cílem bakalářské práce je implementovat aplikace týkající se hledáním podobných resp. opačných slov a tvorbou slovních analogií. Dále si práce klade za cíl provést analýzu sentimentu a zařazování zadaného textu do předem definovaných kategorií.

3.1 Podobná slova

První aplikací vektorové reprezentace slov je nalezení podobných slov k zadanému termínu. To znamená, že hledáme takové slovo, jehož vzdálenost je nejmenší k zadanému slovu - viz pseudokód č. 3.1.

```
1 Vstup:  vektory_slov, vstupni_slovo
2 Výstup:  pole_n_slov
3
4 vstupni_slovo = vstupni_slovo.lower()
5 foreach slovo in vektory_slov:
6     vzdalenost = cos_podobnost(vstupni_slovo, slovo)
7     pole_n_slov = insertion_sort(vzdalenost, slovo)
```

Pseudokód 3.1: Algoritmus hledání podobných slov

Prvním bodem programu je načtení natrénovaných vektorů do předem připravených datových struktur.

Dále již následuje hlavní cyklus, ve kterém uživatel může zadávat své vstupy. Program ukončí zadáním slova „exit". Při načítání musíme vstupy převádět na malá písmena, jelikož je takto upraven i náš pracovní lexikon. Dále je nutné zjistit, zda zadané slovo existuje ve slovníku. Pokud ne, upozorníme uživatele na tento problém.

Dále zjistíme, jaký vektor přísluší zadanému vstupu a načteme jej do připraveného pole. Následuje výpočet kosinové podobnosti mezi zadaným slovem a všemi slovy ze slovníku. Tyto hodnoty postupně ukládáme a nakonec vypíšeme na obrazovku například 10 nejbližších slov.

Pro otestování úspěšnosti byl vytvořen testovací set obsahující 100 vstupů - viz příloha č. A.1. Na každém řádku souboru je nejprve slovo, které je vstupem do programu, a dále je zde uvedeno slovo, které by se mělo vyskytnout jako jedno z nejbližších slov. Program vyhodnotí test jako úspěšný, pokud se požadovaný termín nachází v poli nejbližších slov. Toleranci, do jakého pořadí se požadované slovo

musí nacházet, je nutné na začátku programu specifikovat. V rámci této práce byly testovány tolerance 1, 20 a 50 slov, což je dále značeno jako Top1, Top20 a Top50.

Tabulka 3.1: Podobná slova, porovnání různých modelů

Model	**Úspěšnost [%]**		
	Top1	**Top20**	**Top50**
model1	13,3	37,7	58,8
model2	11,1	34,4	54,4
model3	16,7	45,6	64,4
model4	13,3	44,3	60,2
model5	13,3	45,5	64,4
model6	13,3	37,8	58,8
model7	18,8	47,8	65,6
model8	15,6	44,4	58,8
model9	20,0	56,0	73,0

Dále byly otestovány všechny natrénované modely, uvedené v kapitole 2.3 - viz tab. č. 3.1. Nejvyšší úspěšnosti bylo dosaženo na modelu s nejrozsáhlejším slovníkem, s vektory se 100 dimenzemi a 10 slovy v kontextovém okně. Nejmenší úspěšnost byla vyčíslena na modelu s nejmenším slovníkem, s vektory s 50 dimenzemi a 20 slovy v kontextovém okně. Z výsledků provedených testů lze prohlásit, že velikost pracovního lexikonu má vliv na úspěšnost aplikací. Z uvedených pokusů vyplývá, že modely s 200 tis. slovy mohou mít i o 5 % vyšší úspěšnost než u modelů se slovníkem se 100 tis. slovy. Dále z experimentů vyplývá, že počet dimenzí má také vliv na přesnost. Vektor se 100 dimenzemi je schopen zachytit více závislostí a proto jsou modely úspěšnější než ty, jejichž vektory mají jen 50 dimenzí. Posledním parametrem je délka kontextového okna. Z provedených experimentů vyplynulo, že modely s menší délkou okna jsou úspěšnější, než ty s větší délkou. To si lze vysvětlit například tím, že při menším počtu slov v kontextovém okně jsou příbuzná slova často vzájemně zaměnitelná. Větší délku okna je dobré využít v případě, že je v aplikaci důležité vyjádřit příbuznost slov například jako synonyma. Na tuto skutečnost je nutné myslet i při vytváření testovacího setu.

Vypočítaná slova většinou nikdy nebývají přesně na tomto místě, ale v určitém okolí. Při podobných aplikacích je vždy užitečné se po skončení testu podívat manuálně, jaké výsledky program vrací pro jaké vstupy. Následující výpis demonstruje výsledek pro vstupní slovo „škola".

```
Zadej slovo: škola

školy, pedagogická, pedagogického, pedagogické, vzdělávací,
studia, vzdělávacím, školou, zš, fakulta
```

Je nutné myslet na to, že výsledné slovo může mít jinou předponu, příponu či koncovku. To znamená, že by možná bylo vhodnější do testovacích dat ukládat pouze kořeny slova, nebo ještě lépe tzv. lemmata. Pomoci by mohla i nadstavba Word2Vec metody FastText, která by tento problém mohla eliminovat - viz kapitola č. 1.3.1.

3.2 Protichůdná slova

Druhá aplikace vektorové reprezentace slov se zabývá nalezením protichůdných slov (antonym). Program vychází z aplikace hledání podobných slov - viz pseudokód č. 3.2.

```
1 Vstup:  vektory_slov, vstupni_slovo
2 Výstup:  pole_n_slov
3
4 vstupni_slovo = vstupni_slovo.lower()
5 novy_vektor = ((,,pekny'' - ,,osklivy'') +
6      (,,mlady''-,,stary'') + ...))/N + vstupni_slovo
7 foreach slovo in vektory_slov:
8      vzdalenost = cos_podobnost(novy_vektor, slovo)
9      pole_n_slov = insertion_sort(vzdalenost, slovo)
```

Pseudokód 3.2: Algoritmus hledání protichůdných slov

Na úvod programu je potřeba načíst lexikon spolu s vektory do vytvořených datových struktur. Opět je nutné zjišťovat, zda slovo zadané od uživatele existuje ve slovníku. Pokud slovo existuje, uložíme jeho vektorovou reprezentaci.

Nyní je možné úlohu vyřešit několika způsoby. První, nejjednodušší, možností je úlohu implementovat stejně jako při hledání příbuzných slov, ovšem nehledat nejbližší, ale naopak nejvzdálenější slova. To znamená, že takto nalezená slova se s nejmenší pravděpodobností budou vyskytovat spolu se zadaným slovem. Implementace tímto způsobem by ovšem sloužila k nalezení „teoreticky opačných slov“, nikoliv slov opačného významu.

Vhodným řešením je tedy využít sčítání resp. odčítání vektorů a aplikovat podobnou analogii: Dobrý - X = 1/N *((Pěkný - Ošklivý) + (Mladý - Starý) + ...). Očekávaným výsledkem tedy budou slova opačného významu ke slovu „dobrý“, to znamená slova příbuzná ke slovu například „špatný“. Následuje výpočet kosinové míry mezi tímto vektorem a všemi vektory ze slovníku. Tyto hodnoty postupně ukládáme a nakonec vypíšeme na obrazovku například 10 nejbližších slov.

Pro aktuální úlohu byl vytvořen testovací set 100 vstupů, podobně jako u minulé aplikace - viz příloha č. A.2. To znamená, že na každém řádku je nejprve uvedeno

vstupní slovo spolu s požadovaným výstupem. Program vyhodnotí test jako úspěšný, pokud se požadovaný termín nachází v poli nejbližších slov. Toleranci, do jakého pořadí se požadované slovo musí nacházet, je nutné na začátku programu specifikovat. V rámci této práce byly testovány tolerance 1, 20 a 50 slov, což je dále značeno jako Top1, Top20 a Top50.

Tabulka 3.2: Opačná slova, porovnání různých modelů

Model	**Úspěšnost [%]**		
	Top1	**Top20**	**Top50**
model1	10,3	21,6	35,8
model2	10,3	19,5	35,4
model3	13,4	25,4	46,3
model4	10,3	23,5	40,0
model5	9,2	21,6	35,8
model6	9,2	19,5	33,7
model7	11,3	25,6	46,3
model8	10,3	24,1	42,1
model9	14,0	26,0	53,0

Následně byl proveden experiment, při kterém bylo otestováno všech devět modelů - viz tab. č. 3.2. Z uvedených výsledků je zřejmé, že úspěšnost této aplikace je přibližně o dvacet procent nižší, než u předchozí úlohy. Skutečnost lze vysvětlit tak, že při výpočtu nového vektoru se žádaný vektor slova nemusí nacházet přímo na tomto vypočteném místě, ale ve větším okolí. Je pochopitelné, že při zvyšování tolerance, to znamená pořadí do kolikátého se může vyskytovat požadovaný výsledek, se bude zvyšovat i úspěšnost. Přestože má aplikace menší přesnost, stále je možné porovnat modely mezi sebou. Podobně jako u hledání podobných slov, nejnižší přesnosti dosahují modely s nejmenším pracovním lexikonem, naopak nejlepší modely využívaly rozsáhlejší slovník. Zvýšení počtu dimenzí vektoru z 50 na 100 zvyšuje úspěšnost přibližně o pět až deset procent. Stejně jako u předchozí úlohy i zde zvýšení počtu slov v kontextovém okně nevedlo ke zvýšení úspěšnosti. Možné vysvětlení je stejné jako u úlohy pro hledání podobných slov, kdy je při delším okně brán zřetel i na další slova a dochází zde k úpravě závislostí mezi slovy. Místo automatického vyhodnocování bývá tedy zajímavější si manuálně ověřit několik vstupů a podívat se na výsledky. Následující výpis demonstruje výsledek pro vstupní slovo „sever”.

```
Zadej slovo: sever

severu, jih, severovýchod, jihu, severovýchodě, severní,
oblastí, severozápadě, lokální, sousedním
```

V tomto případě se slovo „jih” skutečně nachází v očekávané sekvenci. Je ale patrné, že se zde vyskytují často i slova příbuzná, než opačná. Možná by zde bylo i vhodnější přesněji specifikovat rovnici, nebo ji i rozšířit o další protiklady.

3.3 Analogie

Třetí úloha vektorové reprezentace slov se zabývá tvorbou slovních analogií. Program vychází z aplikace hledání podobných a opačných slov - viz pseudokód č. 3.3.

```
1 Vstup:   vektory_slov, slovo1, slovo2, slovo3
2 Výstup:  pole_n_slov
3
4 slovo1 = slovo1.lower()
5 slovo2 = slovo2.lower()
6 slovo3 = slovo3.lower()
7 novy_vektor = slovo1 − slovo2 + slovo3
8 foreach slovo in vektory_slov:
9     vzdalenost = cos_podobnost(novy_vektor, slovo)
10    pole_n_slov = insertion_sort(vzdalenost, slovo)
```

Pseudokód 3.3: Algoritmus tvorby slovních analogií

Prvním krokem programu je načtení pracovního lexikonu a vektorů těchto slov. Dále následuje hlavní cyklus, ve kterém uživatel postupně zadává několik slov (standardně 3), například ve formátu: „Praha Česko Německo”. Hlavním cílem této aplikace je tedy vyřešit analogii: „(Praha - Česko) = (X - Německo)”. To znamená, že vypočteme vektor X tímto způsobem: „X = Praha - Česko + Německo”. Pro vypočtený vektor určíme podobnost se všemi slovy ve slovníku a vypíšeme na obrazovku požadovaný výstup. Očekávaným výsledkem by mělo být slovo „Berlín”.

Tímto způsobem lze vyřešit i předchozí aplikaci, která se zabývá hledáním antonym. Případně ji rozšířit pro vyhledání synonym.

Pro vyhodnocení úspěšnosti aplikace byl, podobně jako u předchozích úloh, vytvořen testovací set se 100 vstupy - viz příloha č. A.3. Na každém řádku souboru je uložena čtveřice slov, např. „Praha Česko Berlín Německo”. Vstupními slovy jsou první tři a čtvrté je cílem vypočítat. Tolerance, do jakého pořadí se požadované slovo musí nacházet, byly v rámci této práce nastaveny na 1, 20 a 50 slov, což je dále značeno jako Top1, Top20 a Top50. Nejlepší úspěšnost této aplikace byla 55% - viz tab. č. 3.3.

Výsledky aplikace tvorby slovních analogií jsou téměř podobné jako u úlohy hledání opačných slov. Opět platí, že čím větší je slovník a čím více má vektor dimenzí, tím je aplikace úspěšnější. Naopak větší velikost kontextového okna vede ke

Tabulka 3.3: Analogie, porovnání různých modelů

Model	**Úspěšnost [%]**		
	Top1	**Top20**	**Top50**
model1	11	22	36
model2	11	21	31
model3	13	27	48
model4	12	26	42
model5	10	24	35
model6	12	24	37
model7	13	28	52
model8	12	27	43
model9	14	30	56

snížení úspěšnosti, například z toho důvodu, že se při trénování síť mohla zaměřovat na jiné závislosti mezi slovy.

Zřejmě se zde potvrzuje tvrzení, že požadovaný vektor slova se může nacházet ve větším okolí od vypočítaného vektoru. Platí zde stejné doporučení, přesvědčit se vždy manuálně o některých výstupech z programu pro přesnější pochopení situace. Následující výpis demonstruje použití programu při zadaných vstupech „Praha Česko Bratislava”:

```
Zadána tato slova: ['praha', 'česko', 'bratislava']

varšava, slovensko, slovenským, čtyřky, slovenském, bulharska,
paksi, lotyšsko, metalurgs, bulharským
```

Zde je požadované slovo „Slovensko” součástí výsledku, ale nachází se zde i slova v jiném tvaru (s jinou příponou). Díky tvorbě slovních analogií lze například zjišťovat informace o aktuálních či bývalých hlavách jednotlivých států - viz následující výpis:

```
Zadána tato slova: ['česko', 'zeman', 'usa']

amerických, trump, prezidentskou, romney, prezidentské,
clintonův, trumpa, republikánského, švejnar, mccainův
```

3.4 Analýza sentimentu

Další a často důležitou aplikací zpracování přirozeného jazyka (NLP) je analýza sentimentu ve zkoumaném textu resp. dokumentu. Jedná se o možnost, jak z textu určit emocionální názor autora textu a tyto informace dále využít například v marketingu.

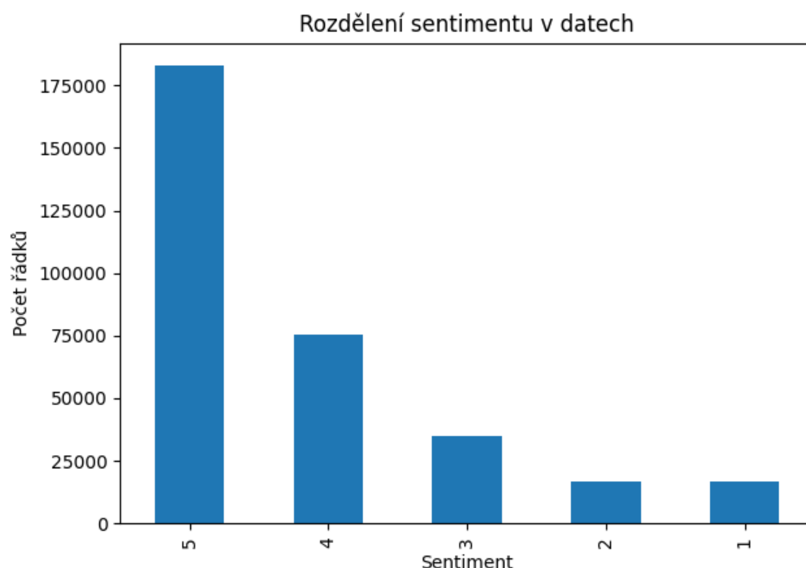
Prvním krokem je získání a stažení vhodných dat. V rámci této práce byly staženy uživatelské hodnocení filmů z webu csfd.cz. Konkrétně byly vybrány všechny komentáře k filmům uvedených v následujících kategoriích: nejlepší filmy, nejhorší filmy a nejrozporuplnější filmy. Celkem se jedná o 900 filmů, z každé skupiny jich je 300. Pro každý takovýto film byla nalezena sekce s komentáři. Nakonec byly všechny komentáře uloženy do csv souboru ve formátu: počet udělených hvězdiček (1-5), komentář. Podrobnější statistiku dat demonstruje tabulka - viz tab. č. 3.4.

Tabulka 3.4: Statistika dat pro analýzu sentimentu

sledovaná veličina	**hodnota**
celkový počet komentářů	326.451
celkový počet vět	1.110.318
celkový počet slov	9.332.541
počet různých slov	366.034
nejčastěji zastoupená slova	a, je, se, to, na, v, že, jsem, ale, film
zastoupení nejčastějších slov	a = 294.228 je = 161.013 se = 143.238 to = 139.434 na = 121.320 v = 100.740 že = 90.504 jsem = 75.012 ale = 70.014 film = 69.567

Je užitečné si vizualizovat rozdělení sentimentu v datech, to znamená počet unikátních komentářů pro každou hodnotu ve stupnici 1 až 5, kde 1 je nejhorší a 5 nejlepší hodnocení, a u každého počet komentářů. Následující graf popisuje takovéto rozdělení dat - viz obr. č. 3.1.

Převažují tedy hodnocení, jejichž filmy získaly nejvyšší známku. Naopak komentářů s jedním, nebo dvěma body je nejméně. Konkrétně 5 bodů odpovídá téměř 180.000 hodnocení, 4 body udělilo přibližně 75.000 uživatelů, 3 body 35.000, 2 body přibližně 17.000 a 1 bod také přibližně 17.000 uživatelů.



Obrázek 3.1: Rozdělení dat pro analýzu sentimentu

Další důležitou součástí je úprava stažených dat. To znamená, odstranění všech nežádoucích znaků, jako jsou interpunkční znaménka, číslovky, nadbytečné mezery, znaky nerespektující kódování utf-8 a podobně.

Posledním krokem úpravy dat je úprava kategorií. Cílem neuronové sítě bude klasifikovat do třech tříd (nejlepší, průměrný a nejhorší film/komentář). Proto byla původní stupnice 1-5 nahrazena tak, že hodnocení 4 a 5 odpovídá nejlepší třídě, hodnocení 2 a 3 odpovídá průměrné třídě, hodnocení 1 odpovídá nejhorší třídě.

Neuronová síť byla napsána v programovacím jazyce Python s využitím knihoven keras, pytorch, gensim, sklearn a knihoven pro vizualizaci výsledků.

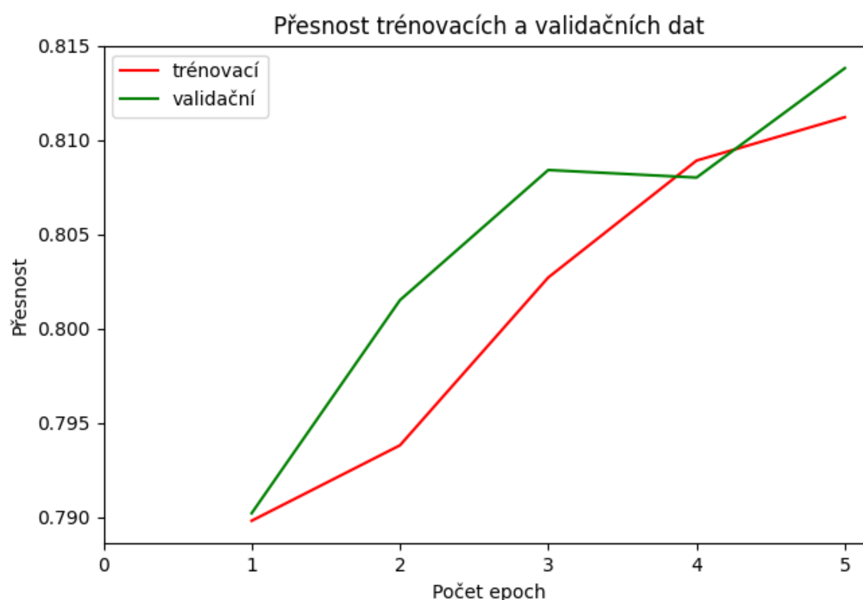
V programu je potřeba rozdělit data do trénovacího validačního a testovacího setu. Bylo zde nastaveno rozdělení na 60% pro trénovací, 20% pro validační a 20% pro testovací data. Dalším krokem je načtení předtrénovaných vektorů slov a vektorizace trénovací a testovací sady dat.

Posledním úkolem je definování a nastavení modelu neuronové sítě. Pro aplikace, jako je například analýza sentimentu, se často využívají rekurentní neuronové sítě. Zde bude využita architektura LSTM, která nabízí řadu výhod oproti klasické RNN, protože ta trpí problémy například s tokem gradientů. Tento typ sítě je schopen se naučit dlouhodobé závislosti v datech a tím pádem je efektivní. Problematika těchto sítí a jejich trénování jde již nad rámec zaměření této práce, a proto není podrobněji vysvětlována. V Pythonu můžeme pomocí knihovny pytorch a keras jednoduše sestrojit síť přidáváním různých vrstev. Nejprve je vložena matice vah, která obsahuje námi již předtrénované vektory. Dále přidáme LSTM vrstvu a nakonec výstupní vrstvu s aktivační funkcí softmax, která bude klasifikovat do třech tříd, specifikovaných výše. Podrobnější informace o zapojených vrstvách jsou zachyceny v následujícím výpise:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 100)         20200900
_________________________________________________________________
lstm (LSTM)                  (None, 128)               117248
_________________________________________________________________
dense (Dense)                (None, 3)                 387
=================================================================
Total params: 20,318,535
Trainable params: 117,635
Non-trainable params: 20,200,900
_________________________________________________________________
```

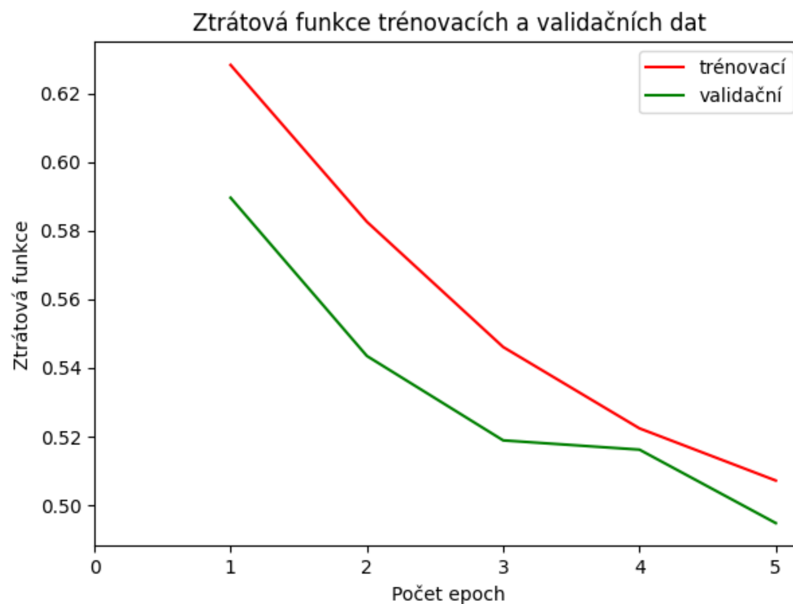
Nakonec se vybere optimalizátor, kterým může být například Adam, nebo SGD. Zde byl zvolen Adam. Posledním krokem je natrénování modelu metodou fit(), ve které je specifikován počet epoch, a další optimalizační parametry. Model byl trénován 5 epoch.

Výsledkem je natrénovaný model, který dosáhl nejlepší úspěšnosti 81,38% na testovacích datech pro model číslo devět. Graf popisuje přesnost na trénovacích a validačních datech, přesnost byla také přes 81% - viz obr. č. 3.2. Ztrátová funkce zde dosahuje hodnoty 0,49 - viz obr. č. 3.3.



Obrázek 3.2: Analýza sentimentu - úspěšnost nejlepšího modelu

Celkem bylo v rámci této úlohy natrénováno pět modelů. Rozdíly v úspěšnosti jsou zachyceny v tabulce - viz tab. č. 3.5. Ztrátové funkce pro tyto modely zaznamenává druhá tabulka - viz tab. č. 3.6. Neuronové sítě byly trénovány na počítači



Obrázek 3.3: Analýza sentimentu - ztrátová funkce nejlepšího modelu

na grafické kartě NVIDIA GeForce GTX 1070 TI. Vždy bylo nastaveno trénování na 5 epoch, jedna epocha trvala vždy přibližně 16 minut, celkem přibližně 95 minut. Z tabulek můžeme zjistit, že se úspěšnosti pro jednotlivé modely příliš neliší, nejhorší je rovna přibližně 79,5 % a nejlepší přibližně 81,4 %. I přesto lze zaregistrovat, že model9 je nejúspěšnější. Jeho vektory využívaly nejrozsáhlejší pracovní lexikon (340.000 slov) a vektory se 100 dimenzemi. Naopak nejméně přesný je model1, který využíval nejmenší slovník a jen vektory s 50 dimenzemi. Modely 4, 7 a 8 vycházejí přibližně stejně a je tedy složitější pro ně vypozorovat nějaké závislosti.

Tabulka 3.5: Porovnání úspěšnosti - analýza sentimentu

Model	**Trénovací data [%]**	**Validační data [%]**	**Testovací data [%]**	**Doba trénování [min]**
model1	79,91	79,62	79,81	89
model4	80,93	80,90	80,96	99
model7	80,87	81,12	80,95	100
model8	80,78	80,50	80,33	95
model9	81,10	81,40	81,38	93

Tabulka 3.6: Porovnání ztrátové funkce - analýza sentimentu

Model	**Trénovací data [%]**	**Validační data [%]**	**Testovací data [%]**
model1	56,33	55,33	55,29
model4	51,83	51,01	50,97
model7	51,17	50,13	50,21
model8	52,10	52,76	52,78
model9	51,60	49,00	49,11

3.5 Kategorizace

Další aplikací je zařazování článků nebo jakýchkoliv textů do různých kategorií. Program řešený v této práci je navržen pro klasifikaci textů do 3 kategorií. Tyto třídy vycházejí z rubrik původních dat, které byly staženy ze zpravodajských webů. Kategorie jsou proto následující: kultura, sport a domácí.

Tabulka 3.7: Statistika dat pro zařazení článku do kategorie

sledovaná veličina	**hodnota**
celkový počet vět	600.000
celkový počet slov	6.703.378
počet různých slov	422.091
nejčastěji zastoupená slova	v, se, a, na, to
zastoupení nejčastějších slov	v = 141.828 se = 137.518 a = 136.978 na = 119.278 to = 74.596

Ze všech těchto rubrik byly staženy texty a z nich odstraněny všechny nežádoucí znaky. Podrobný postup stažení dat je popsán v kapitole 2.1. Dále byly texty rozděleny na jednotlivé věty, které byly společně se třídou uloženy do souboru ve formátu csv. Způsob uložení v souboru je následující: třída (články ze sportu mají kategorii 0, domácí zprávy třídu 1 a kultura byla označena třídou 2), věta. U všech těchto kategorií bylo zajištěno rovnoměrné rozdělení dat tak, že pro každou bylo vybráno 200.000 vět. Podrobnější statistiku dat demonstruje tabulka - viz tab. č. 3.7.

Dále bylo pro tento dataset nastaveno rozdělení na 60% pro trénovací, 20% pro validační a 20% pro testovací data. Poté byl načten soubor s předtrénovanými vektory a vektorizován trénovací soubor.

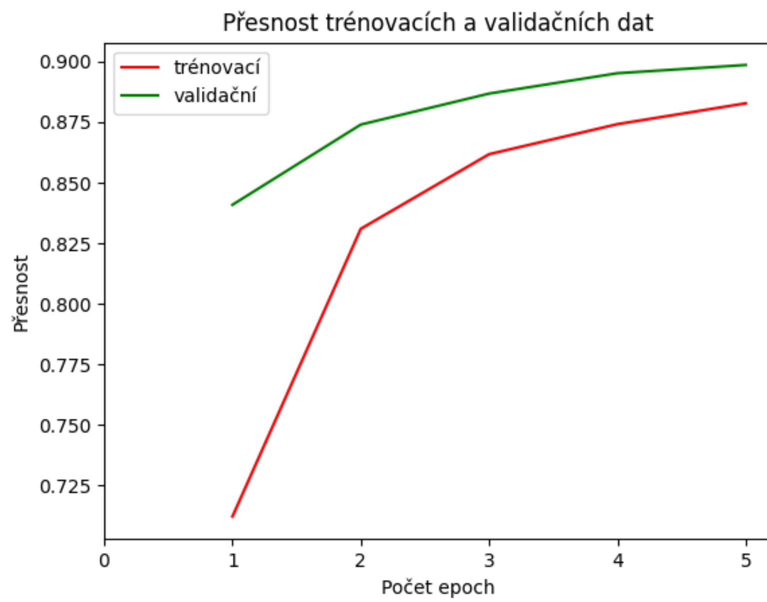
Aplikace pro zařazování článků do kategorií má mnoho společných rysů s aplikací pro analýzu sentimentu, protože zde například také klasifikujeme do třech tříd. Koncept neuronové sítě tedy můžeme použít stejný jako ve zmíněné aplikaci. To znamená, že využijeme architekturu LSTM, která je doplněna výstupní vrstvou s aktivační funkcí softmax. Podrobnější informace o zapojených vrstvách jsou zachyceny v následujícím výpise:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 100)         20200900
_________________________________________________________________
lstm (LSTM)                  (None, 128)               117248
_________________________________________________________________
dense (Dense)                (None, 3)                 387
=================================================================
Total params: 20,318,535
Trainable params: 117,635
Non-trainable params: 20,200,900
_________________________________________________________________
```

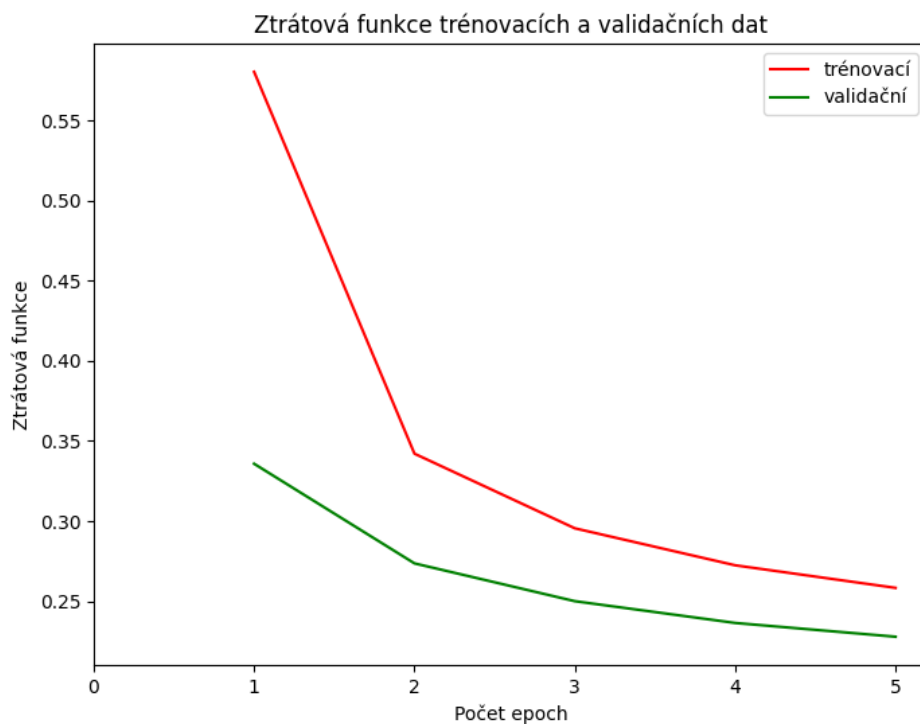
Jako optimalizátor byla zvolena funkce Adam, počet epoch byl nastaven na hodnotu 5.

Výsledkem je natrénovaný model, který dosáhl nejlepší úspěšnosti 89,86% na testovacích datech pro model číslo devět. Graf popisuje přesnost na trénovacích/ validačních datech - viz obr. č. 3.4. Ztrátová funkce zde dosahuje hodnoty 0,22 - viz obr. č. 3.5.

Celkem bylo pro aktuální úlohu natrénováno pět modelů. Rozdíly v úspěšnosti jsou zachyceny v tabulce - viz tab. č. 3.8. Ztrátové funkce pro tyto modely zaznamenává druhá tabulka - viz tab. č. 3.9. Neuronové sítě byly trénovány na počítači na grafické kartě NVIDIA GeForce GTX 1070 TI. Vždy bylo nastaveno trénování na 5 epoch, jedna epocha trvala vždy přibližně 32 minut, celkem přibližně 165 minut. Z tabulek můžeme zjistit, že se úspěšnosti pro jednotlivé modely příliš neliší, podobně jako u předchozí aplikace, nejhorší je rovna přibližně 86 % a nejlepší přibližně 90 %. I přesto lze zaregistrovat, že model9 je nejúspěšnější. Jeho vektory využívaly nejrozsáhlejší pracovní lexikon (340.000 slov) a vektory se 100 dimenzemi. Naopak nejméně přesný je model1, který využíval nejmenší slovník a jen vektory s 50 dimenzemi. Modely 4, 7 a 8 vycházejí přibližně stejně a je tedy složitější pro ně vypozorovat nějaké závislosti.



Obrázek 3.4: Kategorie článků - úspěšnost nejlepšího modelu



Obrázek 3.5: Kategorie článků - ztrátová funkce nejlepšího modelu

Tabulka 3.8: Porovnání úspěšnosti - kategorie článků

Model	**Trénovací data [%]**	**Validační data [%]**	**Testovací data [%]**	**Doba trénování [min]**
model1	83,48	86,11	86,14	170
model4	87,65	89,55	88,80	166
model7	88,10	89,61	89,12	171
model8	87,85	89,62	88,94	161
model9	88,50	89,86	89,45	165

Tabulka 3.9: Porovnání ztrátové funkce - kategorie článků

Model	**Trénovací data [%]**	**Validační data [%]**	**Testovací data [%]**
model1	33,74	29,60	30,02
model4	26,77	23,35	23,54
model7	26,03	23,25	23,50
model8	26,51	23,26	23,29
model9	25,01	22,30	22,41

Závěr

Bakalářská práce je rozdělena do tří částí: teoretická a řešeršní část, implementační část včetně přípravy dat a natrénování vektorů slov, a část aplikační včetně experimentálního vyhodnocování vlivu různých parametrů na úspěšnost těchto aplikací.

Nejprve je zmapována vektorová reprezentace slov založená na metodě Word2Vec, jsou zde popsány jednotlivé používané architektury a jsou zde zmíněny i varianty této techniky.

Ve druhé sekci se bakalářská práce zabývá přípravou trénovacího souboru pro následné natrénování vektorů slov. Trénovací dataset byl stažen z hlavních zpravodajských webů a byl složen z více než 650.000 článků. Z provedené statistiky, podle očekávání, vyplynulo, že nejvíce se v datech vyskytují spojky a předložky. Dále byly vytvořeny čtyři pracovní lexikony, které obsahují postupně přibližně 100 tis., 150 tis., 200 tis. a 340 tis. nejčastěji se vyskytujících slov. Bylo například zjištěno, že každé z deseti nejčastějších slov se v trénovacím souboru vyskytlo více než milionkrát. Pro jednotlivé slovníky byl vypočítán podíl slov mimo slovník. U nejrozsáhlejšího slovníku odpovídá parametr 0,74 %. Naopak pro nejmenší slovník byl parametr vyčíslen na 2,94 %. Vyšší hodnota zde napovídá, že aplikace, které tento lexikon budou využívat, mohou dosahovat nižší úspěšnosti než aplikace s největším slovníkem. V bakalářské práci je následně rozepsán postup implementace metody Word2Vec. Jako architektura metody byl zvolen skip-gram with negative sampling, který predikuje kontext na základě jednoho vstupního slova. Vzhledem k rozsáhlosti trénovacího souboru byly pro metodu negative sampling vybrány tři vzorky a počet epoch byl nastaven na hodnotu 5. Celkem bylo natrénováno 9 modelů s různými parametry (velikost slovníku, počet dimenzí vektoru a délka kontextového okna).

Třetí, poslední, část demonstruje využití vektorové reprezentace slov pro několik ukázkových úloh. V bakalářské práci jsou implementovány úlohy týkající se vyhledávání podobných a protichůdných slov. Další naprogramovanou aplikací je tvorba slovních analogií. V práci byl pro každou úlohu vytvořen unikátní testovací set, na kterém byly aplikace otestovány. Úspěšnosti se pro jednotlivé úlohy výrazně liší, na základě výsledků bylo zjištěno několik zajímavých skutečností. Prvním důvodem je velikost pracovního lexikonu. Při experimentech s rozsáhlým pracovním lexikonem (200 tis. a 340 tis. slov) aplikace dosahovaly vyšší úspěšnosti než s menší velikostí slovníku (100 tis.). Tento fakt lze potvrdit i díky provedenému výpočtu podílu slov mimo slovník, kde pro slovníky s 200 tis. a 340 tis. slovy vyšla hodnota menší než 2 procenta. Naopak pro nejmenší slovník se 100 tis. slovy vyšel parametr přibližně 3 procenta. Druhým důvodem, proč se úspěšnosti jednotlivých modelů v aplikacích liší, je počet dimenzí vektorů. Bylo zjištěno, že větší počet dimenzí vektorů (100)

vede k lepší úspěšnosti, než při využití vektorů s 50 dimenzemi. Třetím důvodem je nastavení počtu slov v kontextovém okně. Z výsledků vyplývá, že z provedených testů byly úspěšnější ty modely, které měly nastavenou menší hodnotu (10), než modely, které počítaly s velikostí 20 slov. U aplikací pro hledání opačných slov a pro tvorbu slovních analogií je nutné pracovat s určitou mírou tolerance. Vypočtené vzdálenosti neodpovídají přesně pozici požadovaného slova, je potřeba výsledné slovo hledat v určitém okolí. Čím větší toto okolí je, tím se i přesnost úlohy zvyšuje. Je to dáno i způsobem trénování, kdy se námi požadované slovo mohlo vyskytnout se slovem, které mělo odlišný tvar, nebo se jednalo například o synonymum. V úlohách se projevily problémy českého jazyka, kdy vypočítaná slova často měla jinou předponu či příponu. Pokud bychom například nevyžadovali přesný tvar slova, ale testovali bychom na kořenech slov, úspěšnosti by byly vyšší. Pro možné vylepšení by mohla být využita modernější metoda FastText, která dokáže predikovat i slova, která se v trénovacím souboru neobjevila.

Následně je v práci vytvořen program pro analýzu sentimentu do třech tříd (kladné, neutrální a záporné vyznění textu) a aplikace pro zařazování článků do několika tematických kategorií (sport, domácí a kultura). Pro každou úlohu bylo nejprve potřeba stáhnout nový dataset, ve kterém jsou jednotlivé věty označkovány správnou třídou. Aplikace pro analýzu sentimentu využila dataset vytvořený z uživatelských recenzí filmů, aplikace pro zařazování článků do kategorií využívá data z různých rubrik zpravodajských webů. Ačkoliv lze obě aplikace řešit stejným způsobem (architekturou sítě), úspěšnosti se lehce liší. Celkem bylo pro každou úlohu natrénováno 5 modelů s využitím různých vektorů ze druhé kapitoly. Obě úlohy dosahují úspěšnosti přes 79 %. Aplikace pro zařazování článků do kategorií je úspěšnější (89 %). Je to zřejmě dáno skutečností, že data byla stažena ze zpravodajských webů a tím pádem mají podobný tvar jako soubor na kterém byly natrénovány vektory slov, není zde použita obecná čeština, slang, nespisovné či vulgární výrazy. Uživatelské recenze filmů, pro analýzu sentimentu, zpravidla nebývají spisovné, obsahují různé slangové termíny. Druhým důvodem může být skutečnost, že ačkoliv autor udělil libovolnému filmu například nižší hodnocení, mohl zde naopak vyzdvihnout některé dobré rysy filmu. I na těchto dvou úlohách se projevily, již zmíněné, poznatky o velikosti pracovního lexikonu a o počtu dimenzí vektorů. To znamená, že aplikace s nejrozsáhlejším slovníkem a větším počtem dimenzí vektorů byly úspěšnější.

Pokud bych se tématu mohl věnovat i v budoucnu, pokusil bych se zaměřit např. na eliminaci problému spočívajícího v tom, že čeština je ohebný jazyk. Na rozdíl od angličtiny, na kterou se zaměřuje většina autorů publikovaných článků, má velmi bohatou morfologii, díky níž má většina slov mnoho odvozených slovních tvarů lišících se koncovkami, případně i příponami a předponami. Pracovní lexikon pro češtinu má proto mnohonásobně více slov než v případě angličtiny a také vazby mezi slovy ve větě jsou mnohem složitější. Kromě významové souvislosti ji do velké míry řídí gramatická pravidla, která ovlivňují např. konkrétní pád slova či rod slovesa. Bylo by proto zajímavé pokusit se vytvořit a vyhodnotit vektorovou reprezentaci založenou na slovních lemmatech. Před tím by ale bylo nutné sestavit nebo získat vhodný lemmatizér, který by k danému slovnímu tvaru přiřadil správné lemma, což není úplně jednoduchá úloha.

Svět strojového učení se rychle vyvíjí, autoři metod zpřesňují své výsledky a optimalizují je pro rychlejší zpracování. To umožňuje vytvářet nové zajímavé aplikace. Bakalářská práce ukázala jen některé úlohy. Dalšími aplikacemi by mohlo být například: překlad mezi dvěma jazyky, automatické generování titulků pro videa, nebo pokročilejší jazykové modelování.

Literatura

[1] Vnoření slov [online]. 2018. [cit. 1.2.2021]. Dostupné z: https://cs.wikipedia.org/wiki/Vnoření_slov

[2] Jason Brownlee. What Are Word Embeddings [online]. 2017. [cit. 1.2.2021]. Dostupné z: https://machinelearningmastery.com/what-are-word-embeddings/

[3] Tomáš Mikolov. word2vec [online]. 2013. [cit. 2.2.2021]. Dostupné z: https://code.google.com/archive/p/word2vec/

[4] Facebookresearch. FastText: Library for fast text representation and classification. [program]. 18.7.2020. [cit. 2.2.2021]. Dostupné z: https://github.com/facebookresearch/fastText

[5] Word embeddings [online]. 2021. [cit. 3.2.2021]. Dostupné z: https://www.tensorflow.org/tutorials/text/word_embeddings

[6] Sémantická analýza textů [online]. 2011. [cit. 2.2.2021]. Dostupné z: https://blog.seznam.cz/2011/09/semanticka-analyza-textu-2/

[7] Shashank Gupta. Word Embeddings in NLP and its Applications [online]. 2019. [cit. 3.2.2021]. Dostupné z: https://www.kdnuggets.com/2019/02/word-embeddings-nlp-applications.html

[8] Mihajlo Grbovic, Haibin Cheng. Real-time Personalization using Embeddings [online]. 2018. [cit. 5.2.2021]. Dostupné z: https://www.kdd.org/kdd2018/accepted-papers/view/real-time-personalization-using-embeddings-for-search-ranking-at-airbnb

[9] Charlene Chambliss. Using word2vec to Analyze News Headlines and Predict Article Success [online]. 2019. [cit. 4.2.2021]. Dostupné z: https://towardsdatascience.com/using-word2vec-to-analyze-news-headlines-and-predict-article-success-cdeda5f14751

[10] Archna Oberoi. What are Language Models in NLP? [online]. 2020. [cit. 17.4.2021]. Dostupné z: https://insights.daffodilsw.com/blog/what-are-language-models-in-nlp

[11] Jay Alammar. The Illustrated Word2vec [online]. 2019. [cit. 6.2.2021]. Dostupné z: http://jalammar.github.io/illustrated-word2vec/

[12] Kosinová podobnost [online]. 2012. [cit. 6.2.2021]. Dostupné z: https://k47.cz/programovani/kosinova-podobnost.html

[13] Cosine Similarity [online]. 2020. [cit. 29.4.2021]. Dostupné z: https://www.geeksforgeeks.org/cosine-similarity/

[14] Ria Kulshrestha. NLP 102: Negative Sampling and GloVe [online]. 2019. [cit. 6.2.2021]. Dostupné z: https://towardsdatascience.com/nlp-101-negative-sampling-and-glove-936c88f3bc68

[15] Tomáš Mikolov. Distributed Representations of Words and Phrases and their Compositionality [online]. 2013. [cit. 17.4.2021]. Dostupné z: https://arxiv.org/pdf/1310.4546.pdf

[16] Zafar Ali. A simple Word2vec tutorial [online]. 2019. [cit. 7.2.2021]. Dostupné z: https://medium.com/@zafaralibagh6/simple-tutorial-on-word-embedding-and-word2vec-43d477624b6d

[17] Renu Khandelwal. Word Embeddings for NLP [online]. 2019. [cit. 8.2.2021]. Dostupné z: https://towardsdatascience.com/word-embeddings-for-nlp-5b72991e01d4

[18] Tomáš Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space [online]. 2013. [cit. 9.2.2021]. Dostupné z: https://arxiv.org/pdf/1301.3781.pdf

[19] Tomáš Mikolov, Piotr Bojanowski, Edouard Grave, Armand Joulin. Enriching Word Vectors with Subword Information [online]. 2017. [cit. 10.2.2021]. Dostupné z: https://arxiv.org/pdf/1607.04606.pdf

[20] Jiří Materna. word2vec – jednoduchá aritmetika se slovy [online]. 2015. [cit. 10.2.2021]. Dostupné z: http://www.mlguru.com/cs/word2vec-jednoducha-aritmetika-se-slovy/

[21] Harsha Bommana. Deep NLP: Word Vectors with Word2Vec [online]. 2019. [cit. 10.2.2021]. Dostupné z: https://deeplearningdemystified.com/article/nlp-1

[22] Chris Nicholson. A Beginner's Guide to Word2Vec and Neural Word Embeddings [online]. 2020. [cit. 11.2.2021]. Dostupné z: https://wiki.pathmind.com/word2vec

[23] Guide to Word2vec Natural Language Processing [online]. 2018. [cit. 11.2.2021]. Dostupné z: https://medium.com/voice-tech-podcast/an-idiots-guide-to-word2vec-natural-language-processing-5c3767cf8295

[24] Ria Kulshrestha. NLP 101: Word2Vec — Skip-gram and CBOW [online]. 2019. [cit. 11.2.2021]. Dostupné z: https://towardsdatascience.com/nlp--101-word2vec-skip-gram-and-cbow-93512ee24314

[25] Words as Vectors [online]. 2015. [cit. 12.2.2021]. Dostupné z: https://iksinc.online/tag/continuous-bag-of-words-cbow/

[26] Jeevan Madugunda. Representation of words of big text data in vector space [online]. 2020. [cit. 13.2.2021]. Dostupné z: http://home.iitj.ac.in/ suman/-courses/csl7030/students_blog/jeevan_madugunda/

[27] Sanket Doshi. Skip-Gram: NLP context words prediction algorithm [online]. 2019. [cit. 13.2.2021]. Dostupné z: https://towardsdatascience.com/skip--gram-nlp-context-words-prediction-algorithm-5bbf34f84e0c

[28] Word2Vec [online kurz]. 2020. [cit. 13.2.2021]. Dostupné z: https://www.coursera.org/lecture/nlp-sequence-models/word2vec-8CZiw

[29] Halyna Oliinyk. Hierarchical softmax and negative sampling [online]. 2017. [cit. 14.2.2021]. Dostupné z: https://towardsdatascience.com/hierarchical--softmax-and-negative-sampling-short-notes-worth-telling-2672010dbe08

[30] Joulin, A. Bagof tricks for efficient text classification [online]. 2016. [cit. 10.5.2021]. Dostupné z: https://arxiv.org/pdf/1607.01759.pdf

[31] Yoav Goldberg. word2vec Explained: deriving Mikolov et al.'snegative-sampling word-embedding method [online]. 2014. [cit. 10.5.2021]. Dostupné z: https://arxiv.org/pdf/1402.3722.pdf

[32] Kavita Ganesan. Word2Vec: A Comparison Between CBOW, SkipGram & SkipGramSI [online]. 2020. [cit. 14.2.2021]. Dostupné z: https://kavita-ganesan.com/comparison-between-cbow-skipgram-subword/#.YCfFpGhKiM8

[33] Nishan Subedi. FastText: Under the Hood [online]. 2018. [cit. 15.2.2021]. Dostupné z: https://towardsdatascience.com/fasttext-under--the-hood-11efc57b2b3

[34] Armand Joulin. Fasttext.zip: Compressing text classificationmodels [online]. 2016. [cit. 10.5.2021]. Dostupné z: https://arxiv.org/pdf/1612.03651.pdf

[35] Amit Chaudhary. A Visual Guide to FastText Word Embeddings [online]. 2020. [cit. 15.2.2021]. Dostupné z: https://amitness.com/2020/06/fasttext--embeddings/

[36] Jeffrey Pennington, Richard Socher, Christopher D. Manning. GloVe: Global Vectors for Word Representation [online]. 2014. [cit. 15.2.2021]. Dostupné z: https://nlp.stanford.edu/pubs/glove.pdf

[37] Word Vectors in Natural Language Processing: Global Vectors (GloVe) [online]. 2018. [cit. 16.2.2021]. Dostupné z: https://medium.com/sciforce/word-vectors-in-natural-language-processing-global-vectors-glove-51339db89639

[38] Gidi Shperber. A gentle introduction to Doc2Vec [online]. 2017. [cit. 16.2.2021]. Dostupné z: https://medium.com/wisio/a-gentle-introduction-to-doc2vec-db3e8c0cce5e

[39] Sémantická analýza textů (1) [online]. 2011. [cit. 16.2.2021]. Dostupné z: https://blog.seznam.cz/2011/08/semanticka-analyza-textu-1/

[40] Chrisjmccormick. Word2Vec commented [program]. 2019. [cit. 19.2.2021]. Dostupné z: https://github.com/chrisjmccormick/word2vec_commented

[41] SongRb. Word2Vec [program]. 2018. [cit. 19.2.2021]. Dostupné z: https://github.com/SongRb/Word2Vec

A Přílohy

```
malý nízký
obrovský velký
hodný hodnej
špatný nedobrý
dobrý skvělý
hezký krásný
tmavý černý
světlý svítící
plakat křičet
obrazovka monitor
chamtivý lakomý
relaxovat odpočívám
formule ferrari
obvyklý častý
balvan kámen
vzácně ojediněle
premiér babiš
premiér sobotka
amerika donald
usa čína
```

Příloha A.1: Ukázka testovacího setu pro podobná slova

```
hodný zlý
vysoký nízký
bohatý chudý
hodně málo
drahý levný
těžký lehký
tlustý hubený
mokro sucho
hezký škaredý
mladý starý
černý bílý
východ západ
sever jih
nahoře dole
funkční nefunkční
funkční rozbitý
dobrý špatný
chutný nechutný
rovný nerovný
chytrý hloupý
```

Příloha A.2: Ukázka testovacího setu pro opačná slova

```
praha česko londýn anglie
praha česko berlín německo
praha česko bratislava slovensko
praha česko moskva rusko
praha česko washington usa
praha česko brusel belgie
praha česko paříž francie
praha česko řím itálie
praha česko kyjev ukrajina
praha česko soul korea
česko praha anglie londýn
česko praha německo berlín
česko praha slovensko bratislava
česko praha rusko moskva
česko praha usa washington
česko praha belgie brusel
česko praha francie paříž
česko praha itálie řím
česko praha ukrajina kyjev
česko praha korea soul
```

Příloha A.3: Ukázka testovacího setu pro tvorbu analogií