Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Vektorová reprezentácia slov využívajúca morfológiu

Diplomová práca

2017
Bc. Mária Vajdová

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Vektorová reprezentácia slov využívajúca morfológiu

Diplomová práca

Študijný program:	Informatika
Študijný odbor:	2508 Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	RNDr. Michal Forišek, PhD.
Konzultant:	Mgr. Vladimír Boža

Bratislava, 2017
Bc. Mária Vajdová

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Mária Vajdová
Študijný program: informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Vektorová reprezentácia slov využívajúca morfológiu
Vector-space word representation using morphology

Cieľ: V poslednom čase sa ukazuje, že reprezentácia slov pomocou vektorov reálnych čísel má veľa výhodných vlastností. Väčšina algoritmov ale explicitne počíta túto reprezentáciu pre každé slovo z korpusu, čo sa prejaví v nízkej kvalite reprezentácie pre slová, ktoré sa vyskytujú málo často, nemožnosti odhadnúť reprezentáciu pre nové slová a vysokých pamäťových nárokoch. Cieľom práce je vybudovať metódu, ktorá pri odhadovaní reprezentácie slov využíva aj morfologické znaky slova a následne kvalitu tejto metódy otestovať na vhodnom korpuse.

Vedúci: RNDr. Michal Forišek, PhD.
Konzultant: Mgr. Vladimír Boža
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Dátum zadania: 11.11.2015

Dátum schválenia: 16.12.2015 prof. RNDr. Rastislav Kráľovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Ďakujem Usamovi za vedenie práce a pomoc skoro v ľubovoľnej dennej aj nočnej hodine. Mišofovi za schválenie témy a cenné rady. Ďakujem mamine za gramaticku kontrolu a Vlejdovi za nájdenie niekoľkých faktických chýb. A samozrejme, Marekovi, rodine a kamarátom za to, že to so mnou v poslednej dobe prežili a dokonca ma aj podporovali.

Abstrakt

Slovné vektory slúžia ako vstup pre mnohé algoritmy strojového učenia a spracovania prirodzeného jazyka. V diplomovej práci sa zaoberáme vytvorením novej metódy reprezentácie slov pomocou slovných vektorov s využitím morfológie. Podarilo sa nám vytvoriť model založený na metóde Word2Vec a ngramoch jednotlivých slov, ktorý má porovnateľnú úspešnosť s existujúcimi modelmi, nižšiu pamäťovú náročnosť a je schopný predpovedať slovné vektory pre nové slová. Ďalej sme sa v práci zaoberali rôznymi metódami analýzy nášho modelu. Na základe analýzy sa nám následne podarilo ešte viac znížiť pamäťovú náročnosť modelu.

Kľúčové slová: slovné vektory, morfológia, ngramy

Abstract

Word vectors are frequently used as an input for various machine learning and natural language processing algorithms. In this thesis, we propose a new word vectors model enriched with morphology, based on Word2Vec library and character ngrams. Our model achieves results comparable with existing methods and is capable of creating vectors for unseen words while having lower memory requirements. In addition, we design methods for model analysis. Based on these methods we create model with even lower memory requirements.

Keywords: word vectors, morphology, ngrams

Obsah

Úvod

Slovné vektory sú jedným zo spôsobov reprezentácie slov v algoritmoch strojového učenia a spracovania prirodzeného jazyka. Zvyčajne sú predstavované mnohorozmernými vektormi reálnych čísel. Ich úlohou je reprezentovať syntaktické a sémantické vzťahy medzi slovami. Slová, ktoré sú si syntakticky alebo sémanticky podobné, by zároveň mali mať podobné aj prislúchajúce vektorové reprezentácie. Takéto slovné vektory sa následne používajú v ďalších metódach strojového učenia, napríklad strojovom preklade alebo rozoznávaní sentimentu. V poslednej dobe sa rozvíja vytváranie slovných vektorov aj s dôrazom na morfológiu (tvaroslovie) slov. Kým pôvodné metódy brali slovo ako nedeliteľný celok, morfologické metódy sa pozerajú aj na jednotlivé časti slova (predpony, prípony a pod.).

Cieľom našej práce je vyvinúť novú metódu tvorby slovných vektorov, ktorá bude klásť dôraz na morfológiu jednotlivých slov. Zároveň bude mať porovnateľnú úspešnosť s existujúcimi metódami, bude schopná predpovedať slovné vektory pre nové slová a bude menej pamäťovo náročná. Nevýhodou už existujúcich metód je totiž nutnosť vytvárania a pamätania si vektorov pre všetky slová v dostupnom korpuse. Väčšina z týchto metód tiež nevie vytvárať slovné vektory pre nikdy nevidené slová. Náš model by mohol byť schopný rýchlejšie sa učiť (vďaka nízkej pamäťovej náročnosti), a zároveň by mohol lepšie fungovať na špecializovaných korpusoch.

Hlavná časť našej práce sa zaoberá návrhom nového modelu pre slovné vektory založeného na skladaní ngramov, ktorý zodpovedá vyššie uvedeným požiadavkám. Následne predstavujeme metódy na analýzu tohto modelu a spôsoby, akým sa dá ďalej znížiť jeho pamäťová náročnosť.

Prvá kapitola tvorí úvod do problematiky strojového učenia a neurónových sietí, ktoré sú potrebné pri vytváraní modelov slovných vektorov. Ďalej v nej predstavujeme rôzne spôsoby strojovej reprezentácie slov a známy model Word2Vec. Ten je základom všetkých modelov vytváraných v našej práci. V druhej kapitole prinášame prehľad publikovaných prác zaoberajúcich sa slovnými vektormi s využitím morfológie. Tretia kapitola opisuje návrh, implementáciu a testovanie našich modelov. V štvrtej kapitole sa zaoberáme metódami analýzy nami navrhnutého modelu, a zároveň ponúkame spôsob, ako ešte viac znížiť jeho pamäťovú náročnosť.

Kapitola 1

Strojové učenie a slovné vektory

V tejto kapitole si predstavíme vybrané metódy strojovej reprezentácie slov, teda spôsoby, akými sa dajú reprezentovať slová pri algoritmoch strojového učenia. Základom pre našu prácu je reprezentácia slov pomocou slovných vektorov - viacrozmerných vektorov reálnych čísel, ktorú v tejto kapitole podrobne opíšeme.

Model, ktorý je základom pre našu prácu je Word2Vec, v ktorom sú slovné vektory založené na pravdepodobnosti predpovedania slov v kontexte daného slova.

Začiatok kapitoly je venovaný prehľadu metód a základných konceptov strojového učenia a neurónových sietí, ktoré sa používajú pri vytváraní slovných vektorov.

1.1 Použité metódy strojového učenia

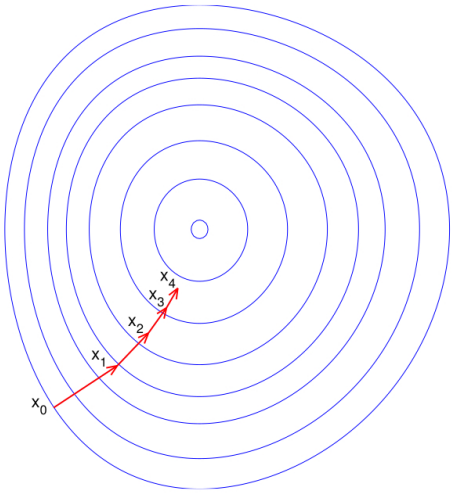
1.1.1 Učenie s učiteľom

Učenie s učiteľom je jeden zo štandardných postupov v strojovom učení. Algoritmy učenia s učiteľom sa postupne učia predpovedať najvhodnejší výstup na základe vstupu. K dispozícii musíme mať sadu vstupných dát x (zvyčajne niekoľko rozmerné vektory), pre ktoré máme dané výstupy y (zvyčajne skaláry). Následne sa snažíme čo najpresnejšie odhadnúť výstupy y pomocou nejakej funkcie na základe vstupných dát x.

Teda snažíme sa nájsť čo najlepšie parametre w pre parametrizovanú funkciu - f_w. Ak pre isté parametre w platí, že $t = f_w(x)$, cieľom je minimalizovať rozdiel medzi y a t úpravou týchto parametrov. T. j. snažíme sa minimalizovať nejakú chybovú funkciu E, ktorá môže mať napr. tvar $E = (y - t)^2$. Proces odhadovania parametrov danej funkcie f sa nazýva *trénovanie* [12].

Metóda najväčšieho zostupu

Pri *metóde najväčšieho zostupu* (gradient descent) sa snažíme nájsť lokálne minimum funkcie (spravidla viac premenných). V strojovom učení sa táto metóda najčastejšie po-



Obr. 1.1: Metóda najväčšieho zostupu [9]

užíva pri učení s učiteľom na hľadanie parametrov funkcie, ktorá najlepšie aproximuje trénovacie dáta.

Využíva sa pozorovanie, že ak sa v nejakom bode pozrieme na gradient danej funkcie a posunieme sa odtiaľ opačným smerom (dostatočne malým krokom), priblížime sa k minimu danej funkcie. Ukážka je na obrázku 1.1. Zvyčajne má táto funkcia v strojovom učení tvar:

$$Q(w) = \sum_{i=1}^{n} Q_i(w),$$

kde Q_i reprezentuje chybovú funkciu pre vstup i pre trénovacie parametre w (ako veľmi sa zatiaľ netrénovaná funkcia odlišuje od testovacieho dáta i). Hodnota pre w sa postupne iteratívne zlepšuje:

$$w = w - \eta \sum_{i=1}^{n} \nabla Q_i(w),$$

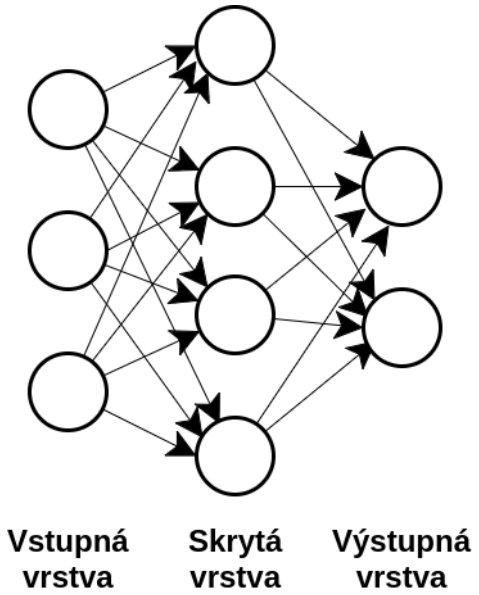
kde η je veľkosť kroku, akým sa pohybujeme, v strojovom učení sa nazýva aj *rýchlosť učenia* (learning rate). Hodnotu w zlepšujeme dovtedy, kým $|w_{old} - w_{new}| > e$, kde parameter e si môžeme určiť [5].

Stochaistická metóda najväčšieho zostupu

Táto metóda je veľmi podobná metóde najväčšieho zostupu, ale nepozeráme sa pri nej na to, aký je gradient celej funkcie, ale pozrieme sa iba na to, ako sa odlišuje jeden vstup (prípadne malá množina vstupov) od našej natrénovanej funkcie. Parameter w upravujeme nasledovne:

$$w = w - \eta \nabla Q_i(w).$$

Túto úpravu urobíme ale postupne pre každý prvok na vstupe. Vo väčšine prípadov je stochaistická metóda najväčšieho zostupu rýchlejšia ako klasická metóda najväčšieho



Obr. 1.2: Jednoduchá sieť s 3 vstupmi, 4 neurónmi skrytej vrstvy a 2 výstupmi

zostupu [5].

1.1.2 Neurónové siete

Slovné vektory sa najčastejšie vytvárajú pomocou neurónových sietí. V tejto časti priblížime koncept neurónových sietí a štruktúru jednotlivých sietí, ktoré sú použité v našej práci.

Dopredná neurónová sieť

Dopredná neurónová sieť je najjednoduchší koncept neurónovej siete. Dopredná znamená, že informácia v sieti sa posúva iba smerom vpred a nikdy sa nevracia alebo neukladá v sieti pre ďalšie použitie (takto fungujú rekurentné siete). Vznik neurónových sietí je vzdialene inšpirovaný štruktúrou neurónov v mozgu a odtiaľ aj pochádza ich názov. Neurónové siete spadajú pod koncept učenia s učiteľom, t. j. bežná neurónová sieť sa snaží čo najlepšie aproximovať výstup na základe vstupu. Sieť sa skladá z jednotlivých neurónov, ktoré sú medzi sebou prepojené. Úlohou neurónu v sieti je prijať vstup z iných neurónov, spracovať ich a poslať ich ďalej po sieti [12]. Ukážka jednoduchej neurónovej siete je na obrázku 1.2.

Najjednoduchšia neurónová sieť sa skladá z troch vrstiev:

1. vstupná vrstva - môže sa skladať z viacerých neurónov, zvyčajne reprezentuje jeden vstup zo vstupných dát

2. skrytá vrstva - môže sa skladať z voliteľného množstva neurónov, jej úlohou je

prijať vstupy zo vstupných neurónov, urobiť ich súčet (váhovaný) a následne prípadne použiť na výsledok aktivačnú funkciu, ktorá vhodne transformuje výsledok

3. výstupná vrstva - reprezentuje výsledok, výstup odhadnutý sieťou

Formálne sa dá dopredná neurónová sieť s jednou skrytou vrstvou, vstupom x dĺžky n a výstupom y dĺžky 1 definovať ako:

$$y = f(\sum_{i=1}^{n} x_i \cdot w_i),$$

kde $w_1, w_2, \ldots, w_n$ reprezentujú skrytú vrstvu, t. j. váhu jednotlivých vstupov na výstupe a f je aktivačná funkcia, napr. sigmoid - σ:

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

ktorý sa používa aj pri tvorbe slovných vektorov [21].

Spätná propagácia

Spätná propagácia v sieti je proces, počas ktorého sa sieť učí a upravuje si svoje parametre. V sieti najskôr prebehne dopredný prechod, počas ktorého sieť skúsi na základe vstupu x vytvoriť výstupy t, tie následne porovná s reálnymi výstupmi y. Následne už prebehne spätná propagácia, počas ktorej si sieť na základe rozdielu medzi y a t upravuje parametre jednotlivých vrstiev. Zvyčajne sa na to používa metóda najväčšieho zostupu [23].

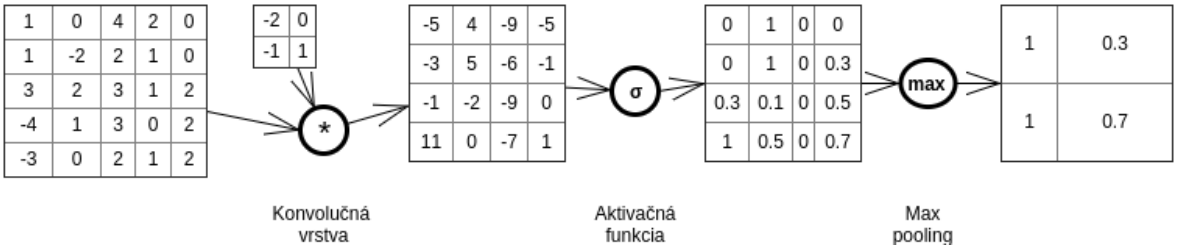
Konvolučné siete

Konvolučné siete sú špeciálny typ neurónových sietí, ktoré sa najčastejšie používajú na spracovanie dát, ktoré majú určitú priestorovú štruktúru. 1D konvolučné siete sa môžu používať na sekvenciu dát, 2D konvolučné siete sa zvyčajne používajú na spracovanie obrázkov.

Všeobecne je konvolúcia definovaná ako operácia na dvoch funkciách reálnych čísel. Konvolúcia funkcií x a w v bode t, pričom vznikne nová funkcia s, sa označuje ako:

$$s(t) = (x * w)(t).$$

Konvolučná vrstva v podstate pre každý pôvodný bod na vstupe vyráta nový bod na základe jeho a jeho okolia. Cieľom konvolúcie môže byť odstránenie šumu z dát alebo aj detekcia istých príznakov v dátach. 1D konvolúcia môže slúžiť napríklad na predpovedanie ďalšieho prvku v časovom rade - na to, aby sme dostali ďalší prvok, sa nechceme pozrieť späť iba na posledný prvok, ale vziať do úvahy niekoľko posledných prvkov



Obr. 1.3: Ukážka konvolučnej siete

a vhodne ich naváhovať. Práve trénovanie hodnôt týchto váh zabezpečuje konvolučná sieť.

Operáciu diskrétnej konvolúcie v čase t môžeme definovať ako:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a),$$

kde x je jednodimenzionálny vstup (resp. funkcia pristupujúca k prvkom vstupu na základe indexu), v našom prípade číselný časový rad a w funkcia, pristupujúca k prvkom matice váh, ktorá sa označuje kernel. V praktických aplikáciách sa nepoužíva nekonečná matica, ale pozeráme sa späť iba na konštantný počet prvkov.

Konvolúciu je možné definovať aj na viacerých dimenziách, následne sa používa viacrozmerný kernel, napríklad 2D konvolúcia na obrázku môže byť definovaná ako:

$$S(i,j) = (I * K)(i,j) = \sum_{m} \sum_{n} I(m,n)K(i-m, j-n),$$

kde K je dvojrozmerný kernel a I je vstupný obrázok. V praxi sa zvyčajne používa niekoľko malých kernelov, pričom cieľom každého z nich je natrénovať sa na rozoznávanie konkrétneho príznaku v obrázku, napr. existujú kernely na hľadanie hrán, istých tvarov, rozostrenie obrázka a pod.

Konvolúcia sa zvyčajne v neurónových sieťach nepoužíva samostatne, ale v spojení s ďalšími dvomi vrstvami:

- Nelineárna aktivačná funkcia (napr. sigmoida) - používa sa po konvolučnej vrstve. Táto fáza je známa aj ako detekčná, pri nej sa napríklad môže rozhodovať, či je nejaký príznak dostatočne viditeľný v istej časti obrázku. Ak áno, vrstva ho prepustí ďalej, inak ho vynuluje.

- Pooling vrstva - používa sa ako posledná, zvyčajne zosumarizuje výsledky konvolúcie. Pooling vrstva zníži rozmery výsledných dát tak, že urobí maximum alebo priemer (prípadne inú operáciu) na danej množine bodov [12].

Schéma celej konvolučnej siete je na obrázku 1.3.

Konvolučné siete s hradbami

V tejto časti opíšeme neurónovú sieť, ktorá pomocou konvolúcie imituje zložitejšie - rekurentné neurónové siete. V rekurentných sieťach je výpočet skrytej vrstvy h daný rekurentne na základe predošlých vstupov. Ak máme vstup v_i, skrytá vrstva preň h_i je definovaná ako:

$$h_i = f(h_{i-1}, v_{i-1}).$$

Takáto rekurentná sieť sa bežne používa na modelovanie textu - ak máme danú sekvenciu slov $w_1, w_2, \ldots, w_n$ úlohou je predpovedať najpravdepodobnejšie nasledujúce slovo w_{n+1}. Skrytá vrstva pre slovo w_{n+1} je potom definovaná ako

$$h_{w_{n+1}} = f(h_{w_n}, w_n).$$

Na základe tohto modelu bol pre modelovanie jazyka vytvorený jednoduchší model, ktorý využíva konvolúciu a nie je rekurentný, skrytá vrstva má potom tvar:

$$h_i(X) = (X * W + b) \otimes \sigma(X * V \ + c),$$

kde X sú slovné vektory pre vstupné slovo a jeho kontext, W a V sú konvolučné matice, b a c vektory posunu, $\otimes$ násobenie po členoch. Druhá časť vzorca $\sigma(X * V + c)$ predstavuje v tejto neurónovej sieti hradby. Vďaka aktivačnej funkcii je možné regulovať, ktoré príznaky z prvej časti vzorca sú dôležité a jedine tie prepúšťať do ďalších vrstiev siete.

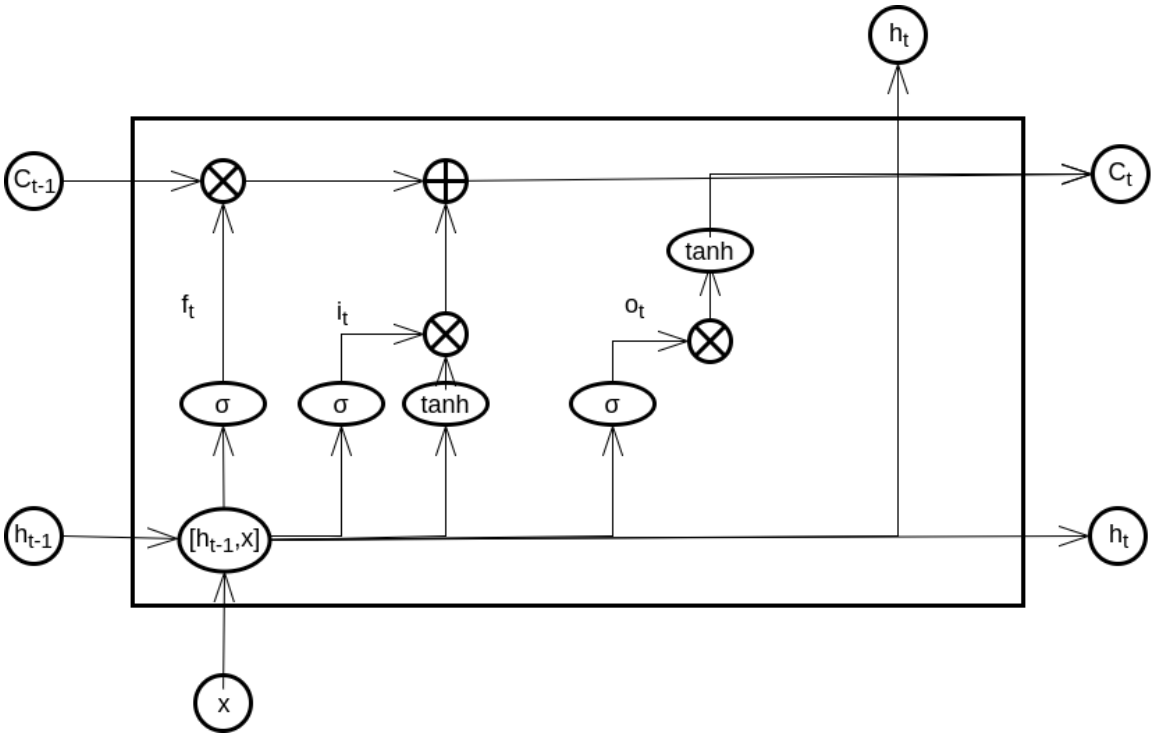
Konvolučná vrstva s hradbami dosahuje porovnateľné výsledky v porovnaní s rekurentnými vrstvami siete (pri trénovaní jazykového modelu), a zároveň nie je taká náročná na trénovanie [10].

LSTM siete

LSTM siete sú vylepšením rekurentných sietí spomenutých v predošlej časti. Klasické rekurentné siete totiž trpia problémom zvaným *vanishing gradient* (strácajúci sa gradient). Tento problém majú aj dopredné siete s viacerými vrstvami. Ak totiž v prvých vrstvách siete aj výrazne zmeníme parametre, spôsobí to iba malú zmenu vo výstupe celej siete, čím sa gradient pre danú vrstvu tzv. stráca. Vanishing gradient nastáva pri používaní istých aktivačných funkcií, ako napr. sigmoid. V rekurentnej sieti sa tento problém prejavuje ešte výraznejšie.

LSTM siete (z angličtiny Long short-term memory) je jeden z modelov rekurentných neurónových sietí. Rekurentná sieť si okrem váh dokáže vo svojom stave pamätať aj predošlé vstupy a využívať ich na predikovanie výstupu. Takáto sieť sa dá napríklad použiť na predpovedanie ďalšieho slova v texte na základe daných slov, rozpoznávanie reči a pod.

Štruktúra siete je zobrazená na obrázku 1.4. Jej jednotlivé časti si popíšeme nižšie.



Obr. 1.4: Architektúra LSTM siete

- stav siete (C_t) - predstavuje dlhodobú informáciu, ktorú si sieť drží o predchádzajúcich vstupoch
- hradby zabúdania (f_t) - predstavuje prvú úpravu stavu siete, hovorí mu, ktoré časti z predchádzajúceho stavu má zabudnúť, upravuje sa:

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

- vstupné hradby (i_t) - ich úlohou je rozhodnúť, ktoré časti vstupu sa pridajú do stavu siete:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t]) + b_i$$

- vstupné hodnoty (C'_t) - predstavujú hodnoty, ktoré by sme chceli pridať do nového stavu siete:

$$C'_t = tanh(W_C \cdot [h_{t-1}, x_t]) + b_C$$

- výstup (o_t) - je konečne spôsob, akým sa upravuje skrytá vrstva siete (h_t) na základe vstupu a stavu siete (C_t):

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t]) + b_o$$

$$h_t = o_t * tanh(C_t)$$

Následne ešte musíme upraviť stav siete na nový stav: $C_t = f_t * C_{t-1} + i_t * C'_t$ [15].

GRU siete

Jednoduchšia verzia LSTM siete je GRU (Gated recurrent unit) sieť, ktorá spája vstupné (i_t) hradby a zabúdacie hradby (f_t) do jednej hradby (r_t). Tá sa celkovo stará o úpravu stavu siete. Rovnako spája skrytú vrstvu siete a stav siete do jednej matice (h_t). Úpravy GRU siete vyzerajú takto:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$h'_t = tanh(W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * h'_t.$$

GRU sieť sa dá trénovať rýchlejšie ako LSTM, práve vďaka tomu, že spája niektoré jej časti do jedného celku [7].

1.2 Strojová reprezentácia slov

1.2.1 Motivácia

V rámci strojového učenia je veľakrát nutné reprezentovať slová prirodzeného jazyka tak, aby ich významu čo najlepšie rozumel počítač. Najbežnejším spôsobom strojovej reprezentácie slova je niekoľkorozmerný číselný vektor. Ten sa následne používa ako vstup do zložitejších modelov strojového učenia - porozumenie významu textu, poznávanie hlasu, strojový preklad a podobne [18]. V závislosti od toho, ako dobre vstupná reprezentácia zodpovedá slovu, je možné dostať lepší, prípadne rýchlejší výsledok pri použití v neskorších algoritmoch. Vyvinulo sa mnoho prístupov prekladu slova do číselného vektoru, ktoré sa líšia svojím využitím a kvalitou.

1.2.2 Distribučná hypotéza

Distribučná hypotéza hovorí, že slová, ktoré sa v texte vyskytujú v podobnom kontexte, zvyčajne mávajú aj podobný význam, pričom sa ale nikdy nemusia v texte vyskytovať spolu. Napríklad veta *Obliekla som si sveter, pretože bola zima*, bude dávať zmysel, aj keď slovo *sveter* nahradíme slovom *kabát*, *bunda* a pod., a naozaj všetky tieto slová majú podobný význam - oblečenie chrániace pred zimou. Pri testovaní distribučnej hypotézy sa skutočne ukázala pozitívna korelácia medzi kontextom, v akom sa slová vyskytujú a ich podobnosťou [22]. Práve z tohto dôvodu sa distribučná hypotéza často používa ako základ pre mnohé metódy strojovej reprezentácie slov, ktoré sa snažia zachytiť ich význam a podobnosť.

1.2.3 Jednoduché spôsoby reprezentácie slov

One-hot encoding

Najjednoduchším spôsobom ako strojovo reprezentovať slová v rámci nejakej množiny je *kód 1 z n* (one-hot encoding). Ak máme usporiadanú množinu slov veľkosti m - $(w_1, w_2, \ldots, w_m)$, i-te slovo tejto postupnosti reprezentujeme ako vektor dĺžky m tvaru:

$$(0, 0 \ldots 0, 0, 1, 0, \ldots, 0),$$

kde 1 sa nachádza na pozícii i a na zvyšných pozíciách sú nuly. Tento prístup je nevýhodný z dôvodu vysokej pamäťovej náročnosti (zapamätanie si jedného slova zaberá $O(m)$ pamäte) a tiež nijakým spôsobom nedokáže zachytiť význam daného slova.

Daným spôsobom vieme reprezentovať aj dlhšie texty zaznačením počtu výskytov jednotlivých slov v texte do vektoru. Táto metóda sa nazýva *batoh slov* (bag of words) [16].

Skrytá sémantická analýza

Pri *skrytej sémantickej analýze* (latent semantic analysis) ako vstup použijeme texty reprezentované pomocou batohu slov. Ak máme n textov, v ktorých sa celkovo vyskytuje m rôznych slov, získame maticu veľkosti $n \times m$, skladajúcu sa z reprezentácií jednotlivých textov pomocou batohu slov. Daná matica ale obsahuje príliš veľa núl, preto sa používa niektorá z metód pre zníženie počtu dimenzií, napr. *analýza hlavných komponentov* (principal component analysis), pomocou ktorej vieme znížiť dimenziu n matice napr. na k ($k < n$) a následne z matice dostávame k-dimenzionálne vektory pre všetkých m slov [28].

Náhodné indexovanie

Pri metóde *náhodného indexovania* (random indexing) najskôr slovám náhodne pridelíme elementárne vektory e_w:

$$\vec{e_w} = [a_1, a_2, \ldots, a_n], a_i \in \{-1, 0, 1\}.$$

Hodnota n sa spravidla pohybuje okolo 1000. Následne pre každé slovo zrátame:

$$\vec{v}_w = \sum_{w' \in \textit{výskyty slova } w} \; \sum_{w_2 \in \textit{okolie slova } w'} \vec{e}_{w2},$$

kde okolie je zvyčajne definované ako k slov vyskytujúcich sa pred slovom v texte a k slov, vyskytujúcich sa hneď za slovom, pričom k býva približne 10. Táto metóda napriek svojej jednoduchosti vykazuje veľmi dobré výsledky. Jej výhodou je, že vektory vznikajú inkrementálne, už po vypočítaní slovných vektorov pre niekoľko slov vieme

hľadať podobnosti medzi nimi. Pri ďalšom spracovaní je možné použiť takto vzniknuté vektory (veľkej dĺžky, obsahujúce nuly) alebo ich normalizovať na požadovanú dĺžku napr. predtým spomenutou analýzou hlavných komponentov [24]. Podobným spôsobom funguje napr. Glove model slovných vektorov [19].

1.3 Word2Vec

1.3.1 Slovné vektory

Reprezentáciou slov prirodzeného jazyka, ktorou sa budeme zaoberať v tejto práci, sú niekoľkorozmerné vektory zložené z reálnych čísel. Nazývajú sa aj kontextové vektory, pretože zdroj ich významu pochádza z kontextu, v akom sa vyskytujú vo vstupnom texte. Vzdialenosti slov vo vektorovom priestore následne dokážu reprezentovať sémantické a syntaktické podobnosti daných slov. Táto vlastnosť vzniká na základe distribučnej hypotézy. Teda ak sledujeme v texte dve slová, ktoré sa opakovane vyskytujú v blízkosti tých istých slov, sú to podobné slová. Napríklad ak máme slová **mačka** a **pes**, obe sa môžu vyskytovať v nasledujúcich kontextoch:

- **Mačka** má štyri nohy. **Pes** má štyri nohy.
- Môj domáci miláčik je **mačka**. Môj domáci miláčik je **pes**.

Zároveň sa ale nikdy nemusia vyskytnúť v texte spoločne [16]. Vektory potom vznikajú na základe pravdepodobnostného modelu, v ktorom sa snažíme pre dané slovo predpovedať okolité slová v texte.

Táto práca bude založená na vytváraní slovných vektorov pomocou knižnice a metódy *Word2Vec*. Slovné vektory vytvorené touto metódou vykazujú okrem blízkosti podobných slov aj zaujímavé aritmetické vlastnosti. Napríklad pre anglický jazyk bol pozorovaný vzťah:

$$v(king) - v(man) + v(woman) \sim v(queen)^1.$$

Ak urobíme dané numerické operácie na vektoroch, výsledkom je vektor, ktorý je v našom vektorovom priestore najbližšie k vektoru reprezentujúcemu slovo *queen*. Rovnako vieme pozorovať aj zaujímavé gramatické vlastnosti, napríklad:

$$v(better) - v(good) + v(smaller) \sim v(small)^2.$$

Touto metódou sa dajú jednoducho a časovo efektívne trénovať veľké množstvá slovných vektorov, pričom nie sú kladené veľké požiadavky na vytváranie vstupu - ako

[1] V slovenčine je to ekvivalentné vzťahu $v(kráľ) - v(muž) + v(žena) \sim v(kráľovná)$

[2] V slovenčine je to ekvivalentné vzťahu $v(lepší) - v(dobrý) + v(menší) \sim v(malý)$

Obr. 1.5: Syntaktické vzťahy vo vete

vstup pre Word2Vec sa dá použiť ľubovoľný čistý text, napr. anglická Wikipédia. Word2Vec vektory sa trénujú pomocou neurónových sietí, pričom sa používajú dva rôzne prístupy - *skip-gram* a *súvislý batoh slov* [18]. Obidve metódy aj s ich variáciami sú popísané nižšie.

1.3.2 Príklady využitia slovných vektorov

Slovné vektory sa zvyčajne nepoužívajú samostatne, aj keď je možné ich takto využiť napríklad na hľadanie synoným slov alebo vyhodnocovanie podobnosti dvoch textov.

Najbežnejšie sa predpočítané slovné vektory používajú ako vstup pre iné modely strojového učenia. Ukazuje sa, že ak sa v modeloch (ktoré ako vstup berú slová) použijú ako vstup slovné vektory namiesto náhodných vektorov, dané modely následne dosahujú väčšiu úspešnosť, príp. je možné ich natrénovať rýchlejšie.

Nižšie sú popísané vybrané spôsoby použitia slovných vektorov v iných modeloch.

Parser syntaktických vzťahov

Syntaktické vzťahy reprezentujú gramaticko-sémantický vzťah medzi jazykovými prvkami na úrovni vetnej konštrukcie. Syntaktické vzťahy spájajú jednotlivé prvky vo vete s ďalšími prvkami, s ktorými existuje nejaká súvislosť. Napríklad ak máme vetu: *Moja dcéra rada počúva modernú hudbu.*, slovo dcéra sa nachádza v dvoch syntaktických vzťahoch, a to: *moja dcéra* a *dcéra počúva*. Sú to slová, s ktorými dané slovo vo vete priamo súvisí, zdieľa s nimi gramatické kategórie a pod. Napríklad slovo *moja* by bez slova *dcéra* v danej vete vôbec nemalo zmysel. Zobrazenie syntaktických vzťahov pre túto vetu je ukázané na obrázku 1.5.

Vo svojej práci Chen a Manning [6] navrhli model, ktorý sa snaží predpovedať syntaktické vzťahy vo vete, pričom ako vstup dostáva slovné vektory. Pri porovnaní s použitím one-hot-encodingu ako vstupu sa ukázalo, že úspešnosť parsera sa môže použitím slovných vektorov zvýšiť o niekoľko percent. Takýto parser zároveň beží niekoľkonásobne rýchlejšie [6].

	výsledok presný	výsledok binárny
náhodné vektory	43.9 (0.6)	82.0 (0.5)
slovné vektory	49.7 (0.4)	87.5 (0.8)
vylepšované slovné vektory	51.0 (0.5)	88.0 (0.3)

Tabuľka 1.1: Porovnanie sietí na predpovedanie sentimentu

Analýza sentimentu

Pri analýze sentimentu úloha spočíva v odhadnutí citového zafarbenia vstupného textu. Využíva sa napríklad pri predpovedaní, či je nejaký textový komentár alebo recenzia produktu pozitívna alebo negatívna. Vo svojej práci Tai, Socher a Manning [26] navrhli konkrétnu neurónovú sieť so stromovou štruktúrou, ktorá na základe sekvencie vstupných slov predikuje sentiment. Autori robili experimenty, pri ktorých skúšali trénovať danú neurónovú sieť na rôznych typoch vstupov. Vstupné slová mohli byť do tejto siete vkladané tromi spôsobmi:

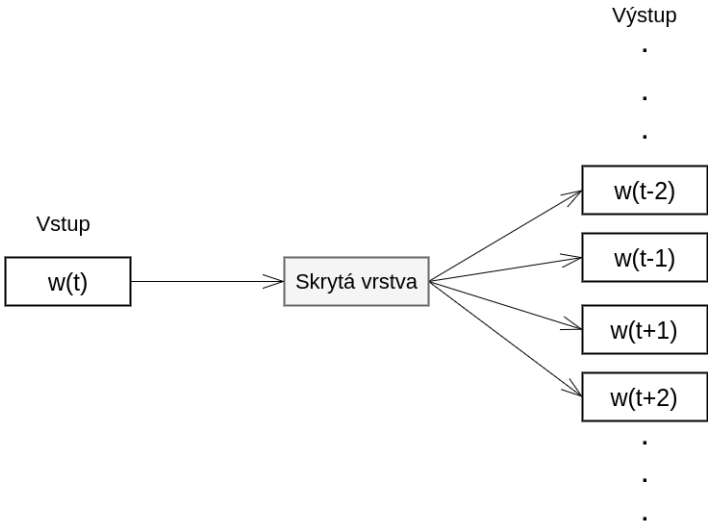
- ako náhodne inicializované vektory
- natrénované slovné vektory
- natrénované slovné vektory, ktoré následne mohol ešte model na trénovanie sentimentu ďalej vylepšovať (dotrénovávať)

Porovnanie úspešnosti natrénovania rozoznávania sentimentu je vidno v tabuľke 1.1.

Výsledok presný zodpovedá zaradeniu sentimentu do 5 kategórií, od najpozitívnejšieho po najnegatívnejšieho, *výsledok binárny* zodpovedá iba rozdeleniu na pozitívne a negatívne texty. Prvá hodnota v tabuľke zodpovedá priemernej percentuálnej úspešnosti pre 5 behov algoritmu, údaj v zátvorke štandardnej odchýlke. Z výsledkov jasne vyplýva, že slovné vektory výrazne zlepšujú presnosť následne natrénovaného modelu [26].

1.3.3 Skip-gram model

Skip-gram je jeden z dvoch základných spôsobov, ako trénovať slovné vektory. Cieľom pri skip-gram modeli je za základe vektoru vstupného slova vedieť predikovať okolité slová (slová, v ktorých kontexte sa dané slovo vyskytuje). Napríklad, ak by sme mali definovaný kontext slova veľkosti 3 a vetu *Juraj jazdí v červenom aute po ulici.*, chceli by sme na základe slova *červenom* predikovať slová: *{Juraj, jazdí, v, aute, po, ulici}*. Na to, aby sa dali predpovedať okolité slová, je definovaná pre každé dve slová pravdepodobnosť $p(w_O|w_I)$, kde w_O je výstupné slovo a w_I vstupné slovo. Ak máme



Obr. 1.6: Architektúra skip-gram modelu

danú postupnosť trénovacích slov $w_1, w_2, \ldots, w_T$, snažíme sa maximalizovať priemerný logaritmus pravdepodobnosti:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t),$$

kde T je veľkosť množiny slov a c veľkosť kontextu, ktorý pre dané slovo berieme do úvahy. Architektúra príslušnej neurónovej siete je zobrazená na obrázku 1.6. Pravdepodobnosť výskytu okolitých slov pre dané slovo je definovaná pomocou softmax funkcie:

$$p(w_O|w_I) = \frac{e^{{v_{wO}}'^{\top} v_{wI}}}{\sum_{w=1}^{W} e^{v'_w v_{wI}}},$$

kde v_I a v'_O sú vstupný a výstupný vektor reprezentujúci dané slovo a W je celkový počet slov v slovníku. Zmysluplný výsledok, ktorý sa následne používa ako vektorová reprezentácia slova w je vektor v_{w_I}. Vektor v_{w_I} je súčasťou matice veľkosti *počet ngramov* $\times$ *veľkost vektora*, v ktorej sú reprezentované všetky vektory ngramov a ich hodnoty sa počas trénovania siete upravujú (táto matica je skrytá vrstva siete). Po každom takomto výpočte sa hodnoty daných vektorov aktualizujú pomocou stochaistickej metódy najväčšieho spádu, aby sa čo najviac zmenšila chyba pri predpovedaní výstupných vektorov. Z danej funkcie je viditeľné, že dĺžka vektorov môže byť ľubovoľná (ale rovnaká pre vstupné a výstupné vektory), zvyčajne sa používajú vektory dĺžky 100, príp. väčšie.

Nevýhodou tohto prístupu je nutnosť pre každú zmenu váhy vyrátať hodnotu

$$\sum_{w=1}^{W} e^{v'_w v_{wI}},$$

t. j. prejsť cez celý slovník. Preto sa vyvinuli metódy, ktoré sú schopné tento výpočet značne urýchliť - *hierarchický softmax* (hierarchical softmax) a *negatívne vzorkovanie* (negative sampling) [18].

Softmax funkcia

Softmax funkcia slúži na reprezentovanie pravdepodobnostnej distribúcie diskrétnej premennej s n možnými hodnotami. Zvyčajne sa používa ako posledná vrstva pre neurónovú sieť, ktorá slúži ako klasifikátor do n rôznych tried. Funkcia by teda mala vyprodukovať vektor y, kde $y_i = P(y = i|x)$. Na to, aby toto bola správna distribúcia pravdepodobnosti, každá z hodnôt y_i musí byť v intervale $(0, 1)$ a zároveň $\sum_{i=1}^{n} y_i = 1$. Ak máme nenormalizované pravdepodobnosti pre všetky triedy $[z_1, z_2, \ldots, z_n]$ z predošlých vrstiev siete, normalizovanú softmax pravdepodobnosť pre z_i dostaneme ako:

$$softmax(z)_i = \frac{exp(z_i)}{\sum_{j=1}^{n} exp(z_j)} [12].$$

V prípade slovných vektorov reprezentuje softmax funkcia pravdepodobnosť predpovedania daného slova v porovnaní so všetkými ostatnými slovami v množine.

1.3.4 Model súvislého batohu slov

Táto metóda *súvislého batohu slov* (continuous bag of words) je veľmi podobná skip-gram modelu, rozdiel je v tom, že sa na základe okolia slova snažíme predikovať dané slovo. V predošlom príklade by to vyzeralo tak, že ak by sme mali trénovaciu vetu *Juraj jazdí v červenom aute po ulici.*, na základe slov *{Juraj, jazdí, v, aute, po, ulici}* by sme sa snažili predpovedať slovo *červenom*. Architektúra neurónovej siete pre túto metódu je zobrazená na obrázku 1.7. Ak máme slovo w_O a jeho okolie $w_1, w_2, \ldots, w_n$, podmienená pravdepodobnosť výskytu slova w_O je definovaná ako:

$$p(w|w_1, w_2, \ldots, w_n) = \frac{e^{v'^{T}_{w_O} h}}{\sum_{w=1}^{W} e^{v'_w h}},$$

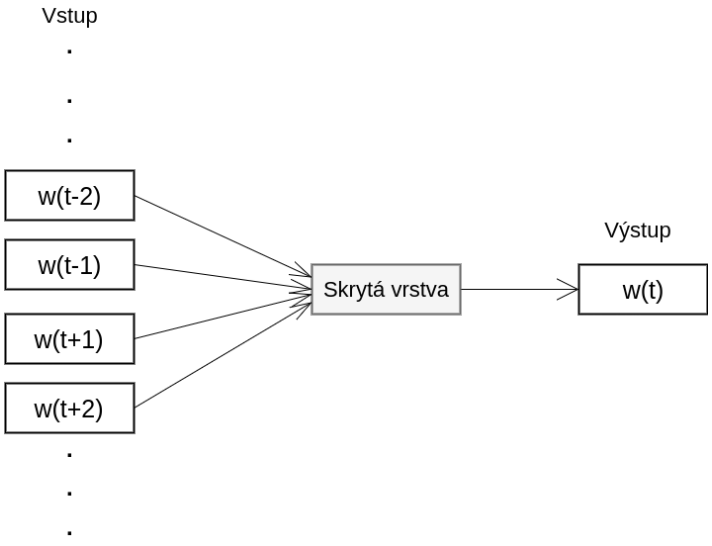
kde h je definované ako:

$$h = \frac{1}{n}(v_{w_1} + v_{w_2} + ... + v_{w_n})^T,$$

$v_{w_1}, v_{w_2}, \ldots, v_{w_n}$ sú vstupné vektory pre slová $w_1, w_2, \ldots, w_n$ a v_{w_O} výstupný vektor pre slovo w_O.

1.3.5 Algoritmické vylepšenia

Aby sa predišlo nutnosti pri každom výpočte $p(w_{t+j}|w_t)$ v skip-gram modeli prechádzať cez celú množinu trénovaných slov, vyvinulo sa niekoľko metód, ktoré daný výpočet urýchlia a pritom príliš neznižujú kvalitu získaných slovných vektorov.



Obr. 1.7: Architektúra modelu batohu slov

Negatívne vzorkovanie

Klasický Word2Vec funguje v podstate ako multinomický klasifikátor (klasifikátor do viacerých tried), len namiesto tried predikuje jednotlivé slová. Takýto klasifikátor však potrebuje reprezentovať rozdelenie pravdepodobnosti medzi triedami - napr. softmax funkciou, ktorej počítanie je veľmi výpočtovo náročné. Preto bola vyvinutá metóda *noise contrastive estimation*, ktorá napodobňuje multinomický klasifikátor pomocou binárneho klasifikátora. Tento binárny klasifikátor rozhoduje iba, či dané slovo patrí do nami vybranej triedy alebo nepatrí [14].

Pri tejto metóde sa pri každom výpočte $p(w_{t+j}|w_t)$ nepozeráme na celú množinu slov, ale náhodne vyberieme k z nich (k môže byť 2-20 v závislosti od veľkosti datasetu), pre ktoré sa snažíme výsledok znížiť, a zároveň výsledok pre naše slovo zvýšiť. Namiesto $p(w_{t+j}|w_t)$ definovanej vyššie, chceme maximalizovať:

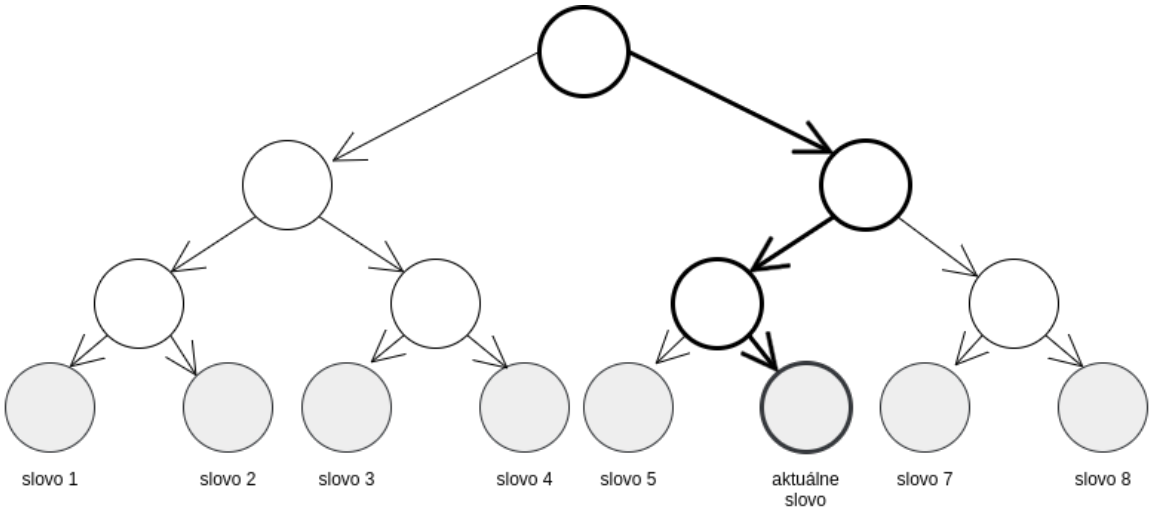
$$\log \sigma(v'^{\top}_{w_O} v_{w_I}) + \sum_{i=1}^{k} E_{w_i \sim P_n(w)}[\log \sigma(-v'^{\top}_{w_i} v_{w_I})],$$

t. j. vyberieme náhodných k vstupov z distribúcie P (zvyčajne rovnomerná distribúcia), ktoré nepatria do kontextu slova a ich vzdialenosť sa snažíme zvýšiť, zatiaľ čo vzdialenosť slova v okolí sa snažíme znížiť. [18].

Táto metóda výrazne znižuje časovú náročnosť vytvárania slovných vektorov, pre každé slovo na vstupe sa menia hodnoty iba pre konštantný počet vektorov, v klasickom algoritme sme museli meniť hodnoty vektorov všetkých slov, ktoré máme v slovníku.

Hierarchický softmax

Pri tejto metóde sa vytvorí binárny strom, v ktorom listy reprezentujú slová v našom slovníku. Zvyčajne sa používa Huffman tree, pretože má tú vlastnosť, že častejšie sa vy-



Obr. 1.8: Strom hierarchického softmaxu

skytujúce slová sa nachádzajú bližšie pri koreni. Každá aktualizácia parametrov potom prebieha ako jeden prechod od koreňa k listu daného slova, pričom sa aktualizujú všetky vrcholy po ceste. Na obrázku 1.8 je cesta zvýraznená. Listy reprezentujú jednotlivé výstupné slová a prechod cez vnútorné vrcholy stromu nahradzuje pri tomto prístupe výstupnú maticu. Pravdepodobnosť výstupného slova w_o, ak máme dané vstupné slovo w_i, sa potom počíta ako:

$$p(w|w_o) = \prod_{j=1}^{L(w)-1} \sigma([\![n(w,j+1) = ch(n(w,j))]\!]) \cdot {v'_{n(w,j)}}^T h,$$

kde $h = v_{w_i}$ v prípade skip-gram algoritmu (t. j. h reprezentuje slovný vektor vstupného slova, ktorý trénujeme). Funkcia $n(w,k)$ reprezentuje k-te dieťa na ceste k listu reprezentujúcemu slovo w a $ch()$ reprezentuje ľavé dieťa vrcholu. Funkcia $[\![x]\!]$ je definovaná ako:

$$[\![x]\!] = \begin{cases} 1 \text{ ak x platí} \\ -1 \text{ inak} \end{cases}$$

Napríklad ak výstupné slovo leží v ľavej polovici stromu, hierarchický softmax mierne zvýši pravdepodobnosť, že všetky slova v ľavej časti stromu sú výstupné slová pre dané vstupné slovo, a zároveň zníži túto pravdepodobnosť pre všetky slová v pravej časti stromu [18, 21].

Bežné implementácie Word2Vec používajú skôr jednoduchšiu metódu negatívneho vzorkovania, ktorú budeme aj my v práci používať.

Podvzorkovanie

Myšlienkou *podvzorkovania* (subsampling) je nebrať veľmi do úvahy slová, ktoré sa v texte vyskytujú príliš často, napr. v angličtine určitý člen *the*. Ak by sme napríklad

mali anglickú vetu *Armstrong was the first person to walk the moon* a ak by sme napríklad použili ako vstup slovo *first* a snažili sa natrénovať jeho kontext, viac informácií nám dá kontext *{Armstrong, was, person, walk}* ako kontext *{was, the, person, walk}*. Slovo *the* sa totiž vyskytuje v kontexte skoro každého slova, preto ním nevieme dané slovo ovplyvniť. Rovnako to platí aj naopak - ak natrénujeme slovo *the* na niekoľko tisíc prípadoch, ďalšie trénovanie ho už veľmi neovplyvní. Preto sa používa nasledovný vzorec pre vyhadzovanie príliš častých slov, slovo w sa zo vstupu odstráni s pravdepodobnosťou:

$$P(w) = 1 - \sqrt{\frac{t}{f(w)}},$$

kde t je nastaviteľná konštanta, zvyčajne 10^{-5} a $f(w)$ je celkový počet výskytov slova w na vstupe.

Použitie podvzorkovania značne zvyšuje kvalitu vytvorených slovných vektorov pre slová, ktoré nemajú veľkú frekvenciu výskytu vo vstupnom texte [18].

1.3.6 Testovanie slovných vektorov

Neexistuje žiadny jednoznačný exaktný spôsob ako ohodnotiť kvalitu slovných vektorov, ale vyvinulo sa niekoľko jednoduchých testovacích úloh, ktoré sa bežne používajú ako metriky. Ich výhodou je nenáročné použitie a kvantitatívny výsledok, ktorý sa ľahko porovnáva.

Podobnosť dvoch vektorov

Podobnosť dvoch slovných vektorov je definovaná na základe ich kosínusovej podobnosti. Ak máme dva slovné vektory a a b dĺžky n, ich podobnosť je definovaná ako:

$$\frac{a \cdot b}{||a||_2||b||_2} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2}\sqrt{\sum_{i=1}^{n} b_i^2}},$$

pričom a_i a b_i sú jednotlivé komponenty vektorov. Kosínusová podobnosť nadobúda hodnoty od -1 po 1, kde -1 predstavuje najmenšiu možnú podobnosť (negatívnu koreláciu) a 1 najväčšiu možnú podobnosť (pozitívnu koreláciu).

Metrika - podobnosť slov

Cieľom tejto úlohy je overiť, do akej miery zodpovedá podobnosť slov medzi slovnými vektormi tomu, ako podobnosť slov vníma človek. V anglickom jazyku (ale aj v mnohých iných) existuje dataset dvojíc slov s následným ohodnotením ich podobnosti na stupnici 1-10. Tento dataset bol vytvorený spriemerovaním množstva hodnotení vytvorených ľuďmi. Hodnotenie úspešnosti algoritmu následne spočíva vo vyrátaní podobností natrénovaných slovných vektorov a vo výpočte korelačného koeficientu (používa sa Pearson aj Spearman) medzi výsledkami danými ľuďmi a slovnými vektormi.

Pre túto úlohu sú zaujímavé dva datasety v anglickom jazyku - `EN-WS353`, ktorý obsahuje bežné slová v angličtine a novší dataset `EN-RW`, ktorý obsahuje hlavne zriedkavé anglické slová (bežné slovné vektory na tomto datasete nedosahujú dobré výsledky, používa sa najmä na vektory využívajúce morfológiu).

Metrika - slovné analógie

Tak ako najznámejší analogický vzťah medzi slovnými vektormi $v(king) - v(man) + v(woman) \sim v(queen)$, sú slovné vektory schopné naučiť sa aj množstvo iných analogických vzťahov. Tieto vzťahy sú dané exaktne, napr:

$$Praha - Česko + Slovensko = Bratislava,$$

teda vzťahy medzi krajinami a ich hlavnými mestami.

Formálne môžeme tento test definovať tak, že máme 4 slová, ktoré sú v exaktnom vzťahu $w_1 - w_2 + w_3 = w_4$. Chceme nájsť najbližší vektor v' k vektoru $v(w_1) - v(w_2) + v(w_3)$. Ak platí, že $v' = v(w_4)$, daný výsledok sa vyhodnocuje ako správny, inak ako nesprávny. Niekedy sa používa aj jemnejšie hodnotenie, napr. že vektor $v(w_4)$ musí patriť medzi 5 najpodobnejších slov. Existuje niekoľko datasetov analógií, ktoré obsahujú syntaktické aj sémantické vzťahy medzi slovami.

Kapitola 2

Podobné práce

V tejto kapitole ukážeme vybrané práce, ktoré sa zaoberali podobnou problematikou ako naša práca. Takýchto štúdií je oveľa viac, ale vo veľa prípadoch sú postupy v nich podobné, preto opíšeme 3 výrazne rozdielne spôsoby, ako pristupovať k morfologickej reprezentácii slov.

2.1 Vektory morfém

Morfológia (alebo v slovenčine aj tvaroslovie) je jazykovedná náuka, ktorá sa zaoberá gramatickými tvarmi slov a slovami, ktoré majú funkciu tvarov. *Morféma* je najmenšia časť slova, ktorá ma nejaký význam alebo funkciu [11]. Predošlé prístupy k vytváraniu slovných vektorov vôbec neberú ohľad na morfológiu slov, na to, aká je vnútorná skladba slova a na slovo sa pozerajú iba ako na nedeliteľnú jednotku. V slovenskom jazyku to napríklad znamená, že môžeme natrénovať kvalitný slovný vektor pre slovo *šťastie*, ale už môže byť problém natrénovať správny vektor pre slová *najšťastnejší, našťastie, šťastia* a iné tvary tohto slova. Dané tvary slova zdieľajú časť významu slova *šťastie*, ale nemusia sa vyskytovať veľmi často v korpuse, a preto ich slovné vektory nemusia byť také presné.

Klasické metódy pre vytváranie slovných vektorov dokážu zachytiť niektoré morfologické vzťahy medzi slovami v angličtine, napr. $v(car) - v(cars) \sim v(apple) - v(apples)$, avšak tieto vzťahy už nefungujú veľmi dobre pre menej frekventované slová v jazyku.

Metóda, ktorá sa snaží vylepšiť predošlé metódy využitím morfológie spočíva v rozbití slov na jednotlivé morfémy - najmenšie jazykové jednotky, ktoré nesú význam. Použilo sa už existujúce morfologické rozdelenie pre slová daného jazyka (predpony, prípony, základ slova). Táto metóda pochádza z článku Luonga a spol. [27].

2.1.1 Trénovanie vektorov jednotlivých morfém

Cieľom je natrénovať slovné vektory pre jednotlivé morfémy v jazyku. Vytvoríme si zoznam všetkých morfém v jazyku M a maticu náhodne iniciovaných vektorov všetkých morfém v danom jazyku - $W_e \in R^{d \times |M|}$, kde d je želaná dimenzia morfologického vektoru a $|M|$ je počet morfém.

Vektory pre zložitejšie slová potom iteratívne vytvárame vždy spojením dvoch morfém pomocou spájajúcej matice W_m, ktorá vytvorí nový vektor dĺžky d z vektorov dvoch morfém. K nemu následne prirátavame tzv. prerušujúci vektor b_m dĺžky d:

$$p = f(W_m[v_{zaklad}; v_{pripona/predpona}] + b_m).$$

Následne výsledok ešte prejde cez aktivačnú funkciu, napr. `tanh`. Vektor b_m, rovnako aj matica W_m sú spoločné pre všetky spájania slovných vektorov, celkovo teda trénujeme:

- maticu W_e veľkosti $d \cdot |M|$, reprezentujúcu vektory jednotlivých morfém
- maticu W_m veľkosti $d \cdot 2d$, ktorá spojí dva vektory morfém (rozmeru $2d$) do jedného vektora veľkosti d
- vektor b_m veľkosti d, ktorý vektorovo reprezentuje spojenie dvoch morfém

Kvalita modelu sa potom zisťuje tak, že porovnávame výsledné vektory pre celé slová, ktoré vytvoril náš model so slovnými vektormi natrénovanými iným spôsobom (napr. Word2Vec) [27].

2.1.2 Vytváranie slovných vektorov z vektorov morfém

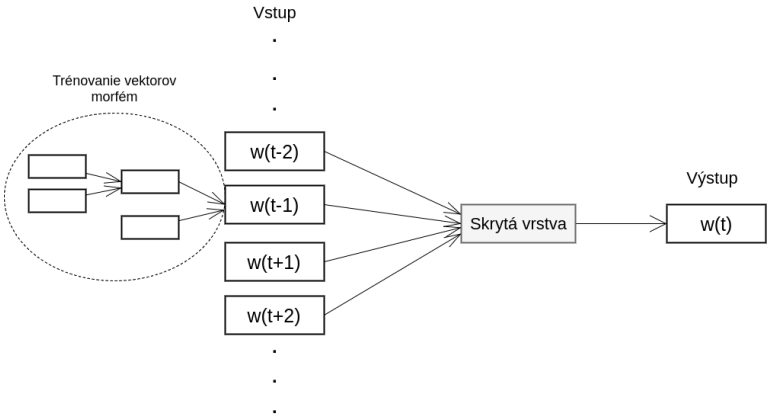
Nevýhodou predošlej metódy je, že pokiaľ sa nejaké zložené slovo nevyskytuje v jazyku príliš často, napr. *distinctness*, jeho slovný vektor nemusí byť správne natrénovaný. Ak sa následne použije na trénovanie morfologických vektorov, môže spôsobiť ich nesprávne natrénovanie.

Preto sa použije kombinovaný prístup - daný model trénuje v najspodnejšej vrstve morfologické vektory, ktoré neskôr použije na vytváranie zložitejších slov a následne štandardným spôsobom trénuje vektory pre ne spolu s vektormi pre jednoduché slová (ktoré sa neskladajú z viacerých morfém).

Spätne sa potom vylepšujú jednotlivé vektory až po vektory pre morfémy, ako je vidno na obrázku 2.1 [27].

2.1.3 Tvorba morfologických značiek

Morfologické značky sa vytvorili pomocou externého programu, ktorý použil skryté markovovské modely na odhalenie predpôn a prípon slov. Následne sa tieto automatické predpony a prípony ešte manuálne pretriedili, aby zostali len skutočné morfémy



Obr. 2.1: Trénovanie slovných vektorov z morfém

v anglickom jazyku. Pri ich vytváraní sa predpokladalo, že všetky slová angličtiny zodpovedajú regulárnemu výrazu:

$$(predpona)^*(základ\ slova)\{1,2\}(pripona)^*,$$

teda slová obsahujú najviac dva slovné základy. Toto platí pre angličtinu, ale nemusí platiť pre jazyky, kde sa iným spôsobom vytvárajú slová, napr. nemecký jazyk alebo slovenský jazyk [27].

Iné metódy hľadania morfologických značiek

Existuje viacero spôsobov hľadania morfologických značiek v texte, zvyčajne sa využíva kombinácia manuálneho označovania človekom a ručnej anotácie.

Napríklad v prípade anotovania Slovenského národného korpusu sa najskôr ručne anotovala menšia časť korpusu a na jej základe sa automaticky vytvorili morfologické značky pre zvyšok [1]. Takisto existujú metódy ako robiť morfologickú anotáciu úplne automaticky [25].

2.2 Ngramové vektory

V práci Bojanowského, Piotra a spol.[4] sa rozhodli použiť iný prístup k morfológii slovných vektorov. Namiesto použitia predtrénovanej informácie o morfémach v danom jazyku, a teda skladaniu vektora pre slovo z morfém, sa rozhodli pre jednoduchšie implementovateľný prístup - skladať vektor pre slovo z vektorov všetkých ngramov (za sebou idúcich sekvencií dĺžky n) v slove. Ako sa ukazuje, tento prístup napriek svojej jednoduchosti vykazuje veľmi dobré výsledky aj pre morfologicky bohaté jazyky.

Táto neurónová sieť funguje veľmi podobne ako klasická Word2Vec sieť, s tým rozdielom, že pre každé slovo sa prepočítajú všetky ngramy dĺžky $3 - 6$, pričom sa pred a za slovo ešte pridajú špeciálne znaky. Pre slovo *vzdelanie* (*$vzdelanie#*) by sme

následne mali n-gramy: *$vz, vzd, zde, del, ela, lan, ani, nie, ie#, $vzd, vzde, zdel, . . . , delani, elanie, lanie#*. Ak máme pre slovo w množinu ngramov G_w, slovný vektor pre slovo w vznikne ako:

$$v(w) = v'(w) + \sum_{g \in G_w} v(g),$$

pričom táto sieť ale zároveň natrénuje aj samostatný vektor pre každé slovo $v'(w)$, ktorý tiež vstupuje do konečnej reprezentácie slova w, t.j. vektora $v(w)$. Napríklad anglické slovo *the* bude mať iný vektor ako samostatné slovo a iný vektor ako ngram v slove *theorem*.

V danej práci nevytvárali ngramy pre n najčastejších slov v jazyku, pre tie boli vytvorené iba slovné vektory, čo zrýchlilo výpočet modelu. Výber n je možné prispôsobiť, čím menšie n, tým presnejší model, ale zároveň pomalšie trénovanie.

Táto metóda napriek svojej jednoduchosti dosahuje lepšie výsledky ako jednoduchý Word2Vec alebo predošlá morfologická metóda. Daný model je veľmi úspešný pri zriedkavých slovách, morfologicky bohatých jazykoch a malých datasetoch. Porovnanie je možné vidieť v tabuľke 2.1. Ako metrika je použitá podobnosť slov z časti 1.3.6. Na základe výsledkov je vidno, že ngramové vektory fungujú z daných modelov najlepšie. Vektory boli v daných prácach natrénované na celej anglickej Wikipédii [4].

	EN-WS353	EN-RW dataset
slovné vektory - skipgram	72	45
slovné vektory s morfológiou	64	34
slovné vektory s ngramami	73	46

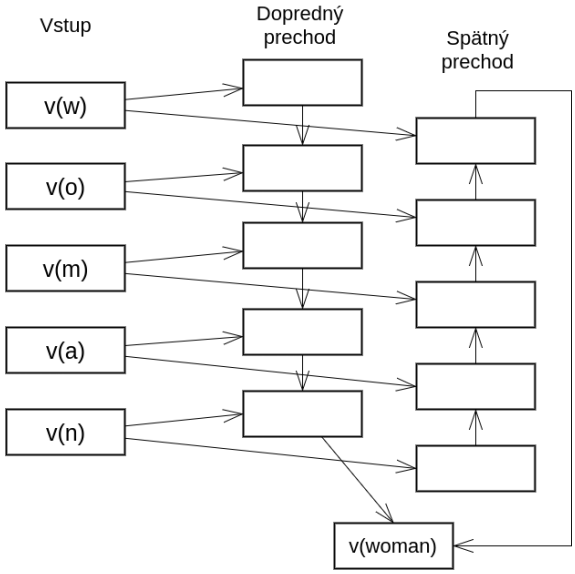
Tabuľka 2.1: Porovnanie jednotlivých modelov slovných vektorov

Skip-gram model uvedený v tabuľke bol tiež natrénovaný autormi práce o ngramových vektoroch.

2.3 Znakové siete

V poslednej dobe sa výrazne rozširuje vytváranie vektorov priamo pre jednotlivé znaky - písmena v texte. Slovné vektory sa následne tvoria spojením vektorov pre znaky. Takáto sieť tiež dokáže zachytávať morfologické vlastnosti textu, napríklad sa dokáže naučiť, čo znamená v anglickom jazyku pridanie *s* na koniec slova a pod.

Takéto modely sa vytvárajú pomocou neurónových sietí, zvyčajne sa na to používajú rekurentné (LSTM) siete, keďže dopredná sieť nie je schopná zapamätať si informáciu o predchádzajúcich znakoch v slove. V práci Linga a spol. [17] použili obojsmernú LSTM sieť [13]. Ako vstup sieť dostane vektory pre jednotlivé znaky a na



Obr. 2.2: Obojsmerná znaková LSTM sieť pre slovo *woman*

základe toho sa snaží predpovedať vektor pre slovo. Bežne takáto sieť znakmi prechádza dvakrát - raz spredu a raz zozadu, a následne oba prechody skombinuje. Keďže je to LSTM sieť, v stave si môže pamätať aj dôležité informácie z predchádzajúcich slov. Architektúra siete je na obrázku 2.2.

Táto sieť, ktorá spája znaky do slov, sa následne používa ako súčasť väčšej siete, pretože tak ako ostatné siete potrebuje slovný kontext na to, aby sa bola schopná učiť slovné (a následne znakové) vektory. Celá sieť sa zvyčajne trénuje tak, že sa snaží na základe sekvencie slov predpovedať ďalšie slovo vo vete (podobne ako batoh slov). Pre všetky slová v slovníku si vyráta slovné vektory a porovná pravdepodobnosť predpovedania správneho slova s pravdepodobnosťou predpovedania ostatných slov v slovníku. Na to sa používa softmax funkcia, tak ako pri klasických Word2Vec modeloch. Následne sa spätnou propagáciou upravujú hodnoty slovných vektorov a váh až po vektory jednotlivých znakov [17].

Kapitola 3

Návrh modelu

V tejto časti sa budeme venovať cieľu našej práce a jej motivácii. Predstavíme nami navrhnuté modely na reprezentáciu slov. Ďalej popíšeme detaily ich implementácie a spôsoby overenia úspešnosti. Nakoniec uvedieme výsledky vykonaných experimentov.

3.1 Cieľ práce a motivácia

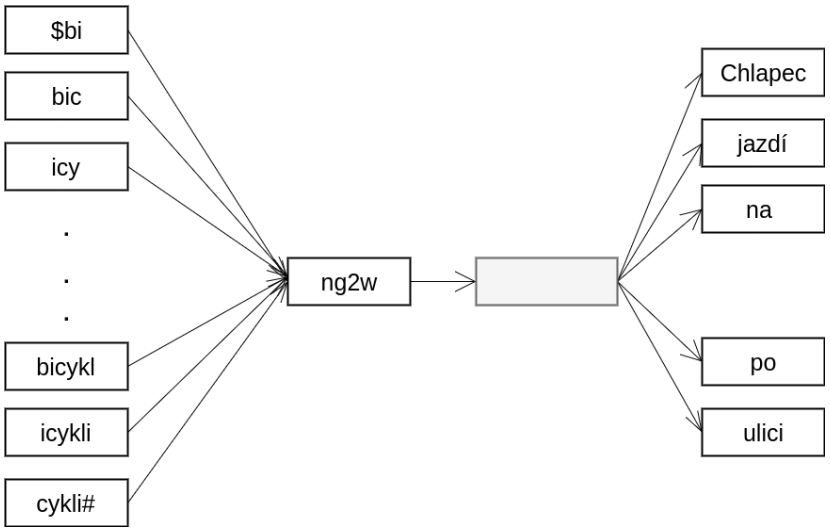
Všetky predošlé uvedené práce si potrebovali pamätať vektory pre jednotlivé slová alebo mali problém so slovami, ktoré nepatrili do ich slovníka. Ak sa pozrieme na 60 miliónov viet v anglickej Wikipédii, nachádza sa v nej asi 20 miliónov rôznych slov a slovných tvarov [17]. Preto sme sa rozhodli navrhnúť model, ktorý nebude závislý od pamätania si vektorov pre jednotlivé slová v jazyku. Požiadavky na náš model sú:

- nízka pamäťová náročnosť
- dostatočne dobrá reprezentácia, porovnateľná s predošlými modelmi
- schopnosť predpovedať neznáme slová v jazyku
- nájdenie čo najlepšieho kompromisu medzi danými vlastnosťami

Predpokladáme, že ak by sa výstupy z takejto siete s malým počtom vektorov použili ako vstup do ďalších modelov, mohli by zrýchliť a zjednodušiť ich trénovanie. Rovnako by mohli fungovať lepšie pri použití v špecifických textoch zameraných na istú oblasť (napríklad odborné články a pod.).

3.2 Návrh

Pri návrhu nášho modelu sme sa rozhodli inšpirovať modelom s vektormi ngramov z predchádzajúcej kapitoly. Daný model sme si vybrali kvôli univerzálnosti ngramov - ich vytváranie zo slov je priamočiare a nezávislé od jazyka, pre ktorý model vytvárame



Obr. 3.1: Ukážka návrhu nášho modelu

(ak berieme do úvahy jazyky, v ktorých sa slová skladajú z jednotlivých znakov písmen). Tento model tiež vykazoval veľmi vysokú úspešnosť. Na rozdiel od modelu opísaného v predošlej kapitole vôbec nebudeme trénovať vektory pre jednotlivé slová, zameriame sa iba na trénovanie vektorov často sa vyskytujúcich ngramov a spôsoby ich skladania do slovných vektorov.

3.2.1 Všeobecný návrh

Náš model budeme trénovať pomocou skip-gram metódy s využitím negatívneho vzorkovania. Rozhodli sme sa brať do úvahy iba vektory dĺžky $3 - 6$ (ako v spomínanej práci), ktoré sa vyskytujú v trénovacom texte aspoň päťkrát. Slovo na vstupe w_I prevedieme na jeho ngramy:

$$w_I \rightarrow [ng_1, ng_2, \ldots, ng_n].$$

Každý z ngramov prevedieme na jeho vektor:

$$[ng_1, ng_2, \ldots, ng_n] \rightarrow [v_{ng_1}, v_{ng_2}, \ldots, v_{ng_n}].$$

Následný krok vytvárania slovného vektora z ngramov je rozdielny pre každý z našich modelov a môžeme si ho všeobecne označiť ako funkciu $ng2w$ ktorej vstupom sú vektory ngramov a výstupom slovný vektor:

$$v_{w_I} = ng2w(v_{ng_1}, v_{ng_2}, \ldots, v_{ng_n}).$$

Cieľom neurónovej siete je potom maximalizovať:

$$\log \sigma(v'^{\top}_{w_O} v_{w_I}) + \sum_{i=1}^{k} E_{w_i \sim P_n(w)}[\log \sigma(-v'^{\top}_{w_i} v_{w_I})].$$

Návrh celého modelu je vidieť na obrázku 3.1. Cieľom siete je teda natrénovať maticu vektorov ngramov, ktoré budú po spojení do vektora slova vytvárať čo najlepšiu reprezentáciu.

Následne predstavíme všetky možnosti pre funkciu $ng2w$, ktoré sme v našej práci vyskúšali.

Kvôli zjednodušeniu modelov sme si určili, že do úvahy budeme brať pri trénovaní iba slová do dĺžky 18 znakov. To nám umožní odhadnúť zhora počet ngramov dĺžky $3-6$ v slovách, takýchto ngramov môže byť najviac 74. Následne nám to umožňuje obmedziť veľkosti trénovaných matíc v skrytých vrstvách.

3.2.2 Sum model

Najjednoduchší z našich modelov je priamo inšpirovaný modelom opísaným v časti 2.2, na rozdiel od neho však nebudeme trénovať slovný vektor pre jednotlivé slová, ale čisto iba pre ngramy. Spojenie vektorov ngramov do jedného slovného vektora je jednoducho suma všetkých vektorov ngramov:

$$v_{w_I} = \sum_{i=i}^{n} v_{ng_i}.$$

Tento model funguje napriek svojej jednoduchosti prekvapivo veľmi dobre. Jeho nevýhodou však je to, že ďalej sa už veľmi nedá vylepšovať - tento model napr. neberie ohľad na poradie ngramov, ani ho nijak nevieme vynútiť.

3.2.3 Min-Max model

Pre modely slovných vektorov existuje predpoklad, že model sa natrénuje tak, aby každá z dimenzií výsledných vektorov reprezentovala istú syntaktickú alebo sémantickú vlastnosť pre daný ngram. Ak teda máme viacero ngramových vektorov, mohlo by byť zaujímavé sa pozrieť na extrémne hodnoty cez jednotlivé dimenzie. Tie by mali reprezentovať najväčšie kvantitatívne vyjadrenie pre dané pravdepodobné vlastnosti. Cieľom tohto modelu je teda vybrať pre každú dimenziu extrémnu hodnotu. Výsledný slovný vektor je dvojnásobnej dĺžky oproti ngramovým vektorom a pozostáva zo zreťazenia minima a maxima ngramových vektorov, pričom jednotlivé zložky ngramového vektora v_{ng_i} dĺžky m definujeme ako $v_{ng_{i1}}, v_{ng_{i2}}, \ldots, v_{ng_{im}}$, maximum a minimum cez dimenziu definujeme ako:

$$min_i = min(v_{ng_{1i}}, v_{ng_{2i}}, \ldots, v_{ng_{ni}})$$

$$max_i = max(v_{ng_{1i}}, v_{ng_{2i}}, \ldots, v_{ng_{ni}})$$

a následne celý slovný vektor:

$$v_{w_I} = [max_1, max_2, \ldots, max_n, min_1, min_2, \ldots, min_n].$$

Tento model je vzdialene inšpirovaný pooling vrstvami spomenutými v časti 1.1.2, ktoré sú zvyčajne súčasťou konvolučných modelov.

3.2.4 Konvolučné modely

Keďže Sum model bol pri tejto úlohe úspešný, rozhodli sme sa vyskúšať konvolučný model, ktorý ešte k sumácii pridáva aj váhovanie jednotlivých ngramov. Štruktúra konvolučného modelu je opísaná v časti 1.1.2.

Pri predošlých modeloch nezáležalo na poradí vkladania ngramov do siete, ale konvolučná sieť už využíva aj poradie vkladaných prvkov, preto vzniká otázka, akým spôsobom vkladať jednotlivé ngramy do siete. V tejto časti budeme ngramy označovať ako napr. $ng3_i$, kde 3 predstavuje ngram dĺžky 3 a i jeho poradie v rámci ngramov daného slova dĺžky tri (ngramy dĺžky 4 budú číslované samostatne, atď.). Možnosti, ako zoradiť ngramy, ktoré sme vyskúšali, sú:

1. zoradenie všetkých dostupných ngramov v rámci 1 konvolúcie, s tým, že nedostupné ngramy sa preskakujú:

   (a) zoradenie podľa dĺžky:

   $$ng3_1, ng3_2, \ldots, ng4_1, ng4_2, \ldots, ng5_1, ng5_2, \ldots, ng6_1, ng6_2, \ldots$$

   (b) zoradenie podľa poradia v slove:

   $$ng3_1, ng4_1, ng5_1, ng6_1, ng3_2, ng4_2, ng5_2 \ldots$$

   (c) vyhradenie dostatočného miesta pre ngramy danej dĺžky, na konci vyplnenie ngramami konca ($*AFT*$) a špeciálne označenie pre ngram mimo slovníka ($*UNK*$). Takéto zoradenie zabezpečí, že ngram istej dĺžky, ktorý začína na istom mieste v slove, bude vždy v našom zoradení na tom istom mieste:

   $$ng3_1, ng3_2, \ldots, ng3_i, *UNK*, ng3_{i+1}, \ldots, *AFT*, *AFT*, \ldots, ng4_1, \ldots,$$

   $$*AFT*, \ldots, ng5_1, ng5_2, \ldots, *AFT*, \ldots, ng6_1, ng6_2, \ldots, *AFT*, \ldots$$

2. 4 konvolúcie pre 4 rôzne dĺžky ngramov:

   - konvolúcia 1 - $ng3_1, ng3_2, \ldots$
   - konvolúcia 2 - $ng4_1, ng4_2, \ldots$
   - konvolúcia 3 - $ng5_1, ng5_2, \ldots$
   - konvolúcia 4 - $ng6_1, ng6_2, \ldots$

Ďalej sme experimentovali s rôznymi veľkosťami konvolučných kernelov. Ako vstup pre konvolúciu sme použili 2D maticu z ngramov veľkosti $n \times m$, kde n označuje počet ngramov pre konvolúciu a m počet dimenzií ngramového vektora. Ak máme ngramy $ng_1, ng_2, \ldots, ng_n$ (zoradené podľa niektorej z možností), vstupná matica má tvar:

$$I = \begin{pmatrix} v_{ng_{1,1}} & v_{ng_{1,2}} & \cdots & v_{ng_{1,m}} \\ v_{ng_{2,1}} & v_{ng_{2,2}} & \cdots & v_{ng_{2,m}} \\ \vdots & \vdots & \ddots & \vdots \\ v_{ng_{n,1}} & v_{ng_{n,2}} & \cdots & v_{ng_{n,m}} \end{pmatrix},$$

zatiaľ čo pre konvolučný kernel sme skúšali veľkosti $10 \times m$, $30 \times m$, $n \times m$ (t.j. kernel pokrýva celý vstup). V každom z modelov sme použili m rôznych kernelov - každý z nich na výstupe reprezentoval jednu z dimenzií slovného vektora.

V závislosti od použitého kernelu mohlo byť ešte potrebné použiť pooling vrstvu (pri kerneloch veľkosti menšej ako vstup). Ako pooling funkciu sme použili priemer, keďže maximum pri predošlom modeli v časti 3.2.3 sa neukázalo ako funkčné. V prípade použitia viacerých konvolúcií sme tiež ich výstupy spriemerovali.

Konvolúcia mohla fungovať v dvoch rôznych módoch:

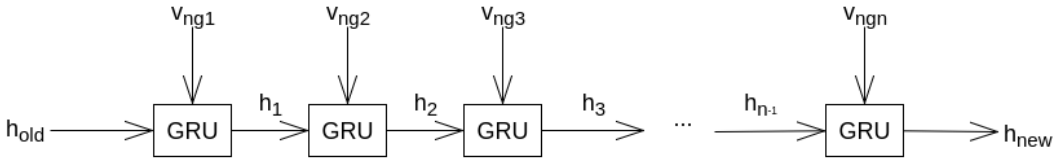
- valid mód - striktne pokrýva iba vstupné dáta, veľkosť výstupu je menšia ako veľkosť vstupu
- same mód - oblasť okolo vstupu sa vyplní nulami a konvolúcia pokrýva všetky možnosti, v ktorých aspoň jeden bod patrí do pôvodného vstupu - veľkosť výstupu je rovnaká ako veľkosť vstupu

Valid mód sa pri experimentoch ukázal ako úspešnejší, zároveň mal aj nižšiu výpočtovú zložitosť (konvolúcia prechádzala cez menej pozícií).

3.2.5 Gated konvolučný model

Pri tomto modeli sme sa inšpirovali gated konvolučným modelom, ktorý sme opísali v časti 1.1.2. Tento model konvolučnou sieťou simuluje hradby. Keďže čisto konvolučné modely sa ukázali ako úspešné, predpokladali sme, že pridanie hradieb, teda vlastne pridanie viac interakcie medzi ngramami by mohlo modelu pomôcť. Vstup sme brali do úvahy rovnaký ako v predošlej časti, budeme ho označovať I veľkosti $n \times m$. Ďalej sme vytvorili dva konvolučné kernely:

- K_1 - štandardný kernel, ktorý váhuje hodnoty jednotlivých ngramov do výsledku
- K_2 - hradbový kernel, jeho úlohou je všímať si dôležité interakcie medzi ngramami a na základe toho vytvárať hradby pre prvý kernel, čiže rozhoduje, ako veľmi budú prepustené ďalej jednotlivé hodnoty z prvého kernelu



Obr. 3.2: GRU model

Na konvolúciu sme použili valid mód, formálne je ju možné definovať ako:

$$h(X) = (X * K_1 + b) \otimes (X * K_2 + c)$$

Výstupom jednej takejto konvolúcie $h(X)$ je vektor veľkosti 1, celkovo sme teda konvolučných kernelov vytvorili m, čo priamo slúžilo ako slovný vektor pre neskoršie vrstvy.

3.2.6 Rekurentné modely

Rekurentný model sme navrhli s použitím GRU - trénovanie modelov je výpočtovo veľmi náročné a GRU sieť je jednoduchšia ako LSTM sieť, takže jej trénovanie prebieha rýchlejšie. Táto sieť je vzdialene inšpirovaná sieťou v časti 2.3. Vyskúšali sme dva spôsoby vkladania ngramov do siete, ktoré sa z predošlej konvolučnej siete javili ako najperspektívnejšie:

1. 4 samostatné GRU pre ngramy rôznych dĺžok zoradených podľa poradia v slove
2. 1 GRU, v ktorej sú ngramy zoradené

   $$ng3_1, ng4_1, ng5_1, ng6_1, ng3_2, ng4_2, ng5_2 \ldots$$

   (rovnako ako pri konvolučnej sieti 1b)

Do siete postupne pre každý vstup vkladáme jeho vektory ngramov $v_{ng_1}, v_{ng_2}, \ldots, v_{ng_n}$, ale na jej výstup, takže na h sa pozeráme vždy iba na konci slova, a vtedy prebieha aj úprava parametrov. Graficky je to znázornené na obrázku 3.2.

3.3 Implementácia

V tejto časti popíšeme knižnicu, pomocou ktorej sme implementovali jednotlivé siete, takisto technické detaily a parametre použitých neurónových sietí.

3.3.1 Keras

Keras je knižnica pre programovací jazyk Python, ktorá poskytuje možnosť rýchlo a jednoducho vytvárať neurónové siete. Jeho výhodou je veľká modularita prvkov, je

možné skoro ľubovoľne skladať jednotlivé vrstvy siete do seba, rovnako definovať sieti niekoľko vstupov a výstupov. Knižnica Keras funguje ako nadstavba nad knižnicami Theano a Tensorflow, ktoré ale nie sú tak úzko špecializované ako Keras a sú komplikovanejšie na používanie [8].

3.3.2 Implementačné detaily modelu

Štruktúra modelu

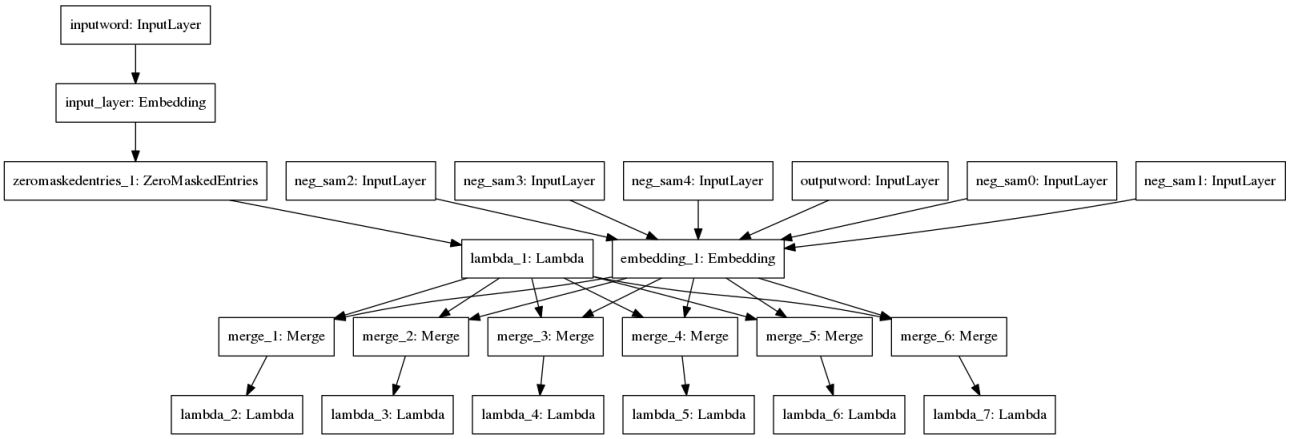
Na obrázku 3.3 je vizualizácia štruktúry Sum modelu v Kerase. Popíšeme, ako jednotlivé časti korešpondujú s návrhom modelu v predošlej časti (ostatné modely sú ekvivalentné okrem časti, ktorá skladá ngramy do slova - $ng2w$):

- inputword - vstup pre sieť, vstupné slovo reprezentované ako poradové čísla jeho ngramov v rámci slovníka
- input_layer - reprezentuje maticu vektorov ngramov vstupného slova
- zeromasked_entries_1 - vrstva, ktorá zabezpečuje doplnenie vektorov ngramov pri menšom počte ngramov ako dané maximum
- neg_sam(1-5) - poradové čísla náhodných slov, vstupy pre negatívne vzorkovanie
- output_word - poradové číslo výstupného slova
- lambda_1 - špeciálna funkcia $ng2w$ - v tomto prípade suma všetkých vektorov ngramov
- embedding_1 - matica vektorov výstupných slov
- merge_(1-6) - skalárne súčiny medzi vstupným vektorom a všetkými výstupnými vektormi (pozitívnym príkladom aj negatívnymi vzorkami)
- lambda_(2-7) - sigmoid funkcie

Model má niekoľko výstupov, pričom výstup pre pozitívne výstupné slovo sa snažíme maximalizovať a výstupy pre negatívne vzorky sa snažíme minimalizovať. Finálny logaritmus, ktorý sa nachádza vo vzorci negatívneho vzorkovania, je tu zabezpečený použitou chybovou funkciou (binary_crossentropy).

V tabuľke 3.1 sú k dispozícii názvy jednotlivých modelov, tak ako sú uvedené v zdrojovom kóde, v súbore `models.py`. Modely sú popísané na základe prehľadu modelov z predošlej časti.

Počet parametrov modelov (okrem parametrov, ktorými sa líšia), je 9144900. Tieto parametre pokrývajú iba výstupnú maticu, t. j. vektory dĺžky 100 pre 91449 slov. Ak



Obr. 3.3: Ukážka Sum modelu v Kerase

by sme ale následne chceli nami natrénované vstupné vektory používať, prípadne ich zapojiť do ďalšej siete (tak ako v modeli v časti 1.3.2), kde sa budú dotrénovávať, tieto parametre sa už používať nebudú. Pre ďalšie použitie modelov teda na ich počte až tak nezáleží.

Počet parametrov jednotlivých modelov, ktorými sa líšia, uvádzame v tabuľke 3.1. Práve tieto parametre sa starajú o vytváranie slovných vektorov, preto je ich počet dôležitý pre následné použitie.

Binárna cross-entropia

Binárna cross-entropia je druh chybovej funkcie, ktorá sa používa pre klasifikovanie dát do dvoch tried. Presnejšie povedané, určuje, či daný prvok patrí alebo nepatrí do danej vybranej triedy. Je zovšeobecnením multinomickej cross-entropie, ktorá sa používa na klasifikovanie do viacerých tried. Vzorec pre binárnu cross-entropiu je len špeciálnym prípadom vzorca pre miltinomickú s $m = 2$:

$$\mathcal{L}(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)\right] = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij}\log(\hat{y}_{ij}),$$

kde n je počet videných príkladov, m počet tried, $\hat{y}_i$ pravdepodobnosti dané modelom a y_i skutočné pravdepodobnosti.

Trénovacie dáta

Modely sme trénovali na textoch z anglickej Wikipédie, ktoré sú voľne dostupné na stiahnutie [2]. Na očistenie textov od XML znakov a pod. sme použili nástroj `WikiExtractor.py` [3] . Z tejto množiny sme následne vyhodili tzv. stop words - slová, ktoré sa vyskytujú v anglickom jazyku príliš často a zhoršovali by trénovanie (the, a, do, we. . .). Ďalej sme odstránili aj všetky slová, ktoré sa v trénovacom texte vyskytovali menej ako 10-krát. Pre každé slovo sme našli jeho ngramy a rovnako aj pri ngramoch

Typ modelu	Názov triedy	Počet parametrov	Ďalšie detaily
Word2Vec (bez ngramov)	WordVectorModel	9 144 900	-
Sum model	NgramSumModel	5 181 500	-
Min-Max model	NgramMaxModel	2 038 850	veľkosť vektorov ngramov - 50
Konvolučný (1a) model	NgramConvShorterFirstModel	5 481 700	veľkosť kernelu 30 × 100, same mód
Konvolučný (1b) model	NgramConvBeginFirstModel	5 481 700	veľkosť kernelu 30 × 100, same mód
Konvolučný (1c) model	NgramConvPaddedModel	5 481 700	veľkosť kernelu 30 × 100, same mód
Konvolučný (2) model	NgramConvMultModel	4 478 100	veľkosť kernelu 10 × 100, same mód
Konvolučný (1a) model	NgramConvShorterFirstValid Model	5 481 700	veľkosť kernelu 30 × 100, valid mód
Konvolučný (1b) model	NgramConvBeginFirstValid Model	5 921 700	veľkosť kernelu *num ngramov* × 100, valid mód
Konvolučný gated (1b) model	NgramConvBeginFirstValid GatedModel	6 661 700	veľkosť kernelu *num ngramov* × 100, valid mód
GRU model 1	NgramGRUMultModel	4 318 900	-
GRU model 2	NgramGRUBeginFirstModel	5 241 800	-

Tabuľka 3.1: Názvy jednotlivých modelov v zdrojovom kóde

vybrali iba tie, ktoré mali minimálny výskyt aspoň 5. Vždy sme na trénovanie používali rovnaké dáta.

Detaily trénovania

Pre všetky trénované modely sme použili parametre opísané v tejto časti. Parametre, ktorými sa jednotlivé modely líšili, sme opísali pri ich definícii. Počas celého trénovania bola rýchlosť trénovania modelov 0.1, okrem `WordVectorModel`, pre ktorý bola rýchlosť trénovania 1.0. Veľkosť slovných a vektorových ngramov sme nastavili na 100 (okrem Min-Max modelu, kde ngramové vektory mali dĺžku 50). Na negatívne vzorkovanie sme

použili 5 negatívnych vzoriek. Pre každé slovo sme brali do úvahy kontext veľkosti 10, t. j. 5 slov, ktoré sa nachádzali v texte pred ním a 5 slov, ktoré sa v texte nachádzali za ním. Všetky modely sme trénovali iba jednou epochou - pri trénovaní slovných vektorov je to štandardné a postačujúce.

Keďže sme brali do úvahy slová iba do veľkosti 18 (20 aj so značkami pred a za slovom), maximálny počet ngramov dĺžok $3 - 6$ v týchto slovách je 74. Väčšina modelov mala 74 stanovený ako maximálny počet ngramov, okrem modelov, ktoré mali samostatné vrstvy pre jednotlivé ngramy, kde bol pre jednu takúto vrstvu stanovený maximálny počet na 20.

3.4 Experimenty

Pre všetky dáta sme trénovanie vykonávali na rovnakých dátach z anglickej Wikipédie. Modely, ktoré nevykazovali veľkú úspešnosť, sme trénovali iba na dátach veľkosti 20 miliónov tokenov, tie ktoré dosahovali lepšie výsledky, sme natrénovali na 100 miliónoch tokenov.

Ako test úspešnosti sme použili úlohu o podobnosti dvoch slov na obidvoch trénovacích setoch dát - `EN-WS353` (dataset bežných slov) a `EN-RW` (dataset zriedkavých slov), ktorá je opísaná v časti 1.3.6. Táto úloha porovnáva koreláciu medzi podobnosťou slov označenou ľuďmi a podobnosťou danou našimi vektormi. Bol použitý Pearsonov korelačný koeficient. Výsledky pre všetky modely na 20 miliónoch tokenov sú v tabuľke 3.2. Výsledky pre úspešnejšie modely na 50 a 100 miliónoch tokenov uvádzame v tabuľke 3.3.

Na výsledkoch je vidno, že veľmi dobre fungoval Sum model, ktorý bol inšpirovaný pôvodným modelom. Konvolučné modely, ktoré používali menšie kernely, t. j. kernel museli tzv. posúvať po ngramoch, veľmi nefungovali. Rovnako sa ukázalo, že GRU modely sa s danou úlohou prekvapivo nevedeli vysporiadať.

Ako najlepšie nami navrhnuté modely sa ukázali modely, ktoré používali kernel veľkosti $n \times m$, kde n je počet ngramov a m ich dimenzia. Zároveň sme pre tieto modely nastavili valid mód, t. j. tieto modely robili to, že pre každú dimenziu výsledného slovného vektora naváhovali všetky dimenzie všetkých ngramov, ktoré do siete vstupovali. Ako náš najlepší model môžeme určite prehlásiť `NgramConvBeginFirstValidModel`. Tento model mal pri 20 a 50 miliónoch tokenoch dokonca lepšie výsledky pre zriedkavé slová ako klasický `WordVectorModel`. To naznačuje, že tento model má vyššiu rýchlosť trénovania pre zriedkavé slová ako `WordVectorModel`.

Zároveň pri 100 miliónoch tokenov tento model `NgramConvBeginFirstValidModel` prekonal jednoduchší `NgramSumModel`, ktorý sa už po 50 miliónoch tokenov nebol schopný veľmi ďalej zlepšovať. Zároveň `NgramConvBeginFirstValidModel` pri 50 a 20

Názov triedy modelu	EN-WS353 20 mil.	EN-RW 20 mil.
WordVectorModel	39.8	9.8
NgramSumModel	36.4	27.9
NgramMaxModel	17.3	21.1
NgramConvShorterFirstModel	30.0	21.4
NgramConvBeginFirstModel	22.1	9.8
NgramConvPaddedModel	3.3	4.6
NgramConvMultModel	27.2	16.3
NgramConvShorterFirstValidModel	19.6	9.6
NgramConvBeginFirstValidModel	**39.0**	**28.5**
NgramConvBeginFirstValidGated-Model	32.9	26.5
NgramGRUMultModel	11.1	11.5
NgramGRUBeginFirstModel	1.7	0.01

Tabuľka 3.2: Výsledky jednotlivých modelov

Názov triedy modelu	EN-WS353 50 mil.	EN-RW 50 mil.	EN-WS353 100 mil.	EN-RW 100 mil.
WordVectorModel	56.1	16.5	62.9	39.0
NgramSumModel	38.5	30.5	42.4	31.2
NgramConvBeginFirstValidModel	**40.7**	**29.9**	**52.2**	**31.5**
NgramConvBeginFirstValidGated-Model	37.0	28.6	51.0	31.5

Tabuľka 3.3: Výsledky úspešnejších modelov

miliónoch tokenov vo väčšine parametrov prekonal `NgramSumModel`. Výhodou tohto modelu teda je, že sa rýchlo naučí veľmi slušne vytvárať slovné vektory pre neznáme slová, a zároveň sa ešte ďalej dokáže výrazne zlepšovať.

Kapitola 4

Hľadanie dôležitých parametrov

V tejto kapitole sa pokúsime analyzovať, akým spôsobom sa správa náš najlepší model `NgramConvBeginFirstValidModel`. Otázky, na ktoré by sme chceli odpovedať, sú:

- Ktoré sú najdôležitejšie ngramy? Dá sa ešte obmedziť počet použitých ngramov?
- Ngramy na ktorých pozíciach sú pre model podstatné?
- Ako dobre bude fungovať model založený iba na najdôležitejších ngramoch?

Daným otázkam sa chceme venovať preto, aby sme lepšie porozumeli fungovaniu nášho najlepšieho modelu. Ten nie je jednoducho pochopiteľný iba z pohľadu na jeho parametre, ktorými váhuje ngramy (ich počet sa blíži k miliónu). Takisto by sme sa chceli pokúsiť (v súlade s cieľom práce) navrhnúť model založený na ešte menšom počte ngramov, t. j. aj na menšom počte parametrov.

Na začiatku kapitoly predstavíme použitú metódu na vysvetľovanie predikcií, ktorú sme si prispôsobili na našu úlohu. Následne predstavíme dve metódy ako hľadať dôležité parametre a ich výsledky. Na konci natrénujeme model založený na obmedzenom množstve ngramov a porovnáme jeho výsledky s predošlými modelmi.

4.1 LIME metóda na vysvetlenie predikcií klasifikátorov

Metóda pochádza z práce Ribeiro a spol. [20]. Slúži na pochopenie predikcií ľubovoľného klasifikátora alebo regresie tak, že ho aproximuje jednoduchším modelom, ktorého parametrom by mal byť schopný porozumieť aj človek. Základné kritériá pre takýto nástroj na porozumenie sú:

- interpretovateľnosť - prináša porozumenie vzťahu medzi vstupom a výstupom modelu, ktoré je pochopiteľné aj pre človeka

- lokálna presnosť - každá vstupná inštancia by mala byť správne vysvetlená aspoň vo vzťahu ku svojmu okoliu, t. j. k inštanciám, ktoré sa na vstupe vyskytujú dostatočne blízko pri nej
- model-agnostickosť - byť schopný vysvetliť každý model bez ohľadu na to, ako vyzerá zvnútra, považovať ho za čiernu skrinku
- globálna perspektíva - ponúknuť človeku pohľad na celý model na základe pár inštancií

4.1.1 Formálna definícia LIME metódy

Definícia 4.1.1. Vysvetlenie modelu je model $g \in G$, kde G je trieda potenciálne vysvetliteľných modelov. Definičný obor funkcie g je $\{0,1\}^{d'}$.

Definícia 4.1.2. $\Omega(g)$ označujeme mieru komplexnosti vysvetlenia g.

Definícia 4.1.3. Model, ktorý sa pokúšame aproximovať modelom g, označujeme ako $f : R^d \to R$.

Definícia 4.1.4. Ak máme vstupné inštancie $x, z \in R^d$, tak $\pi_x(z)$ označuje mieru vzdialenosti medzi inštanciami x a z.

Definícia 4.1.5. Mieru toho, ako veľmi nedôveryhodná je aproximácia funkcie f pomocou funkcie g v okolí bodu x, označujeme $L(f, g, \pi_x)$.

Definícia 4.1.6. Vysvetlenie modelu dostaneme ako:

$$\xi(x) = argmin_{g \in G} L(f, g, \pi_x) + \Omega(g).$$

Vstup, pre ktorý sa snažíme nájsť vysvetlenie, budeme označovať ako x a upravený vstup pre funkciu g ako x'. Na to, aby sme sa naučili lokálne správanie funkcie f, budeme aproximovať funkciu $L(f, g, \pi_x)$ tak, že vyberieme náhodné vstupy z okolia x' váhované funkciou π_x. Pre náhodný vstup $z' \in R^d$ z okolia x' nájdeme jeho reprezentáciu x, použijeme jeho výstup $f(z)$ ako požadovaný výstup preň pre model g. Následne sa snažíme optimalizovať $\xi(x)$ tak, aby na základe vstupov z_1, z_2, z_n z okolia x čo najlepšie vysvetľovala výstupy $f(z_1), f(z_2), \ldots, f(z_n)$.

V prípade, že G obmedzíme na triedu lineárnych modelov, ako napríklad $g(z) = w_g \cdot z$, môžeme definovať L ako:

$$L(f, g, \pi_x) = \sum_{z, z' \in Z} \pi_x(z)(f(z) - g(z'))^2,$$

pričom $\pi_x(x)$ definujeme ako:

$$\pi_(z) = \frac{exp(-D(x, z)^2}{\sigma^2},$$

kde D je miera vzdialenosti medzi vstupmi x a z definovaná daným algoritmom a σ parameter funkcie D [20].

4.2 Naše metódy

V tejto časti popíšeme ako sme použili LIME metódu na vysvetlenie nášho modelu, a zároveň ďalšiu, inú metódu, ktorú sme použili na hľadanie dôležitých ngramov.

4.2.1 Implementácia LIME

Keďže chceme zistiť, ktoré ngramy sú vo výpočte slovného vektora dôležité, jediná časť našich modelov, ktorá nás zaujíma, je prerábanie vektorov ngramov na slovný vektor. Funkciu f definujeme ako funkciu v rámci nášho modelu, ktorá dostáva na vstupe vektory ngramov a jej výstup je slovný vektor. Ak teda máme slovo w a jeho ngramy $ng_1, ng_2, \ldots, ng_n$, vstupom pre funkciu f sú vektory ngramov $v_{ng_1}, v_{ng_2}, \ldots, v_{ng_n}$ a výstupom slovný vektor v_w.

Z dôvodu, aby funkcia g bola čo najjednoduchšia a pochopiteľná človekom ju definujeme ako váhovanú sumu vektorov ngramov na vstupe, t. j.:

$$g(v_{ng_1}, v_{ng_2}, \ldots, v_{ng_n}) = \sum_{i=1}^{n} w_i \cdot v_{ng_i},$$

kde $w_1, w_2, \ldots, w_n$ sú skalárne hodnoty - váhy jednotlivých ngramov. Dané váhy sú parametre nášho modelu pre funkciu g, ktoré sa model snaží optimalizovať v súlade s funkciou $L(f, g, \pi_x)$. V našom prípade sú teda vstupy pre funkciu f a g totožné.

Vzdialenostnú funkciu pre dva vstupy x, z:

$$x = [v_{ng_{x1}}, v_{ng_{x2}}, \ldots, v_{ng_{xn}}]$$

$$z = [v_{ng_{z1}}, v_{ng_{z2}}, \ldots, v_{ng_{zn}}]$$

sme definovali ako:

$$D(x, z) = \sum_{i=1}^{n} c(ng_{xi}, ng_{zi}),$$

kde c je kosínusová vzdialenosť medzi vektormi. Na minimalizovanie $\Omega(g)$ sme použili pri trénovaní parametrov funkcie g $L1$ regularizáciu, ktorá tlačí niektoré parametre do nuly. Pričom samotný model je aj tak ohraničený maximálnym počtom ngramov, z ktorých sa skladajú vektory - t. j. v našom prípade 74.

Ako fungovala naša implementácia LIME algoritmu, ilustrujeme v pseudokóde, kde x je vstup do algoritmu, t. j. vektory ngramov, f pôvodná funkcia, ktorú aproximujeme, *pi* miera vzdialenosti medzi inštanciami.

```
def lime_explanation(x,f, pi):
        inputs ← [500 uniform random samples]
        outputs ← [f(i) for i~in inputs]
        weights ← [pi(x,i) for i~in inputs]
```

```
    g ← fit(inputs, outputs, weights) with L1 regularization
    return g.weights
```

4.2.2 Substitúcia ngramov

Substitúcia ngramov je jednoduchá metóda, ktorú sme vyvinuli pre vyhodnocovanie dôležitosti jednotlivých ngramov. Spočíva v substitúcii jedného z ngramov v danom slove náhodným ngramom. Ak máme slovo w skladajúce sa z ngramov $ng_1, ng_2, \ldots, ng_n$, dôležitosť ngramu ng_i zistíme tak, že ho nahradíme náhodným ngramom z celkovej množiny ngramov ng_r a vyrátame

$$dist = c(f(v_{ng_1}, v_{ng_2}, \ldots, v_{ng_i}, \ldots, v_{ng_n}), f(v_{ng_1}, v_{ng_2}, \ldots, v_{ng_r}, \ldots, v_{ng_n})).$$

Rovnako ako v predošlom prípade, funkcia f bude označovať funkciu, ktorá v našom modeli predpovedá slovný vektor na základe vektorov ngramov a c kosínusovú vzdialenosť. Čím väčšia kosínusová vzdialenosť medzi pôvodným vektorom a vektorom so substituovaným ngramom, tým dôležitejší bol pravdepodobne daný ngram pre daný slovný vektor.

Kvôli dôveryhodnosti výsledku ngram skúsime substituovať niekoľkokrát za rôzne iné náhodné ngramy. Celý algoritmus, ktorý pre dané slovo zráta dôležitosť všetkých jeho ngramov, vyzerá nasledovne:

```
def substitute ngrams(x, f):
        ngram_weights ← []
        for ngram in ngrams(x):
                sum_distance ← 0
                for i~in 100:
                        new_x ← random_substitute(ngram)
                        sum_distance ← sum_distance
                                + cosine_distance(f(x), f(new_x))
                ngram_weights.append(sum_distance)
        return ngram_weights
```

4.2.3 Použitie oboch prístupov

V ďalšej práci budeme používať obidva predtým spomenuté prístupy, čím zároveň otestujeme ich úspešnosť v hľadaní dôležitých parametrov. Cieľom je pomocou nich zistiť, ako dôležité sú ngramy na jednotlivých pozíciách, a zároveň konkrétne ngramy.

Na to, aby sme to zistili, sme náhodne vybrali 10000 anglických slov, pre ktoré sme obidvomi algoritmami zrátali váhy jednotlivých ngramov. Na výstupe sme si zapamätali

Dáta	**Korelačný koeficient**
Ngram	0.34
Pozícia	0.96

Tabuľka 4.1: Korelácia LIME a substitučného prístupu

nielen ktoré konkrétne ngramy dosahovali aké váhy, ale aj váhy ngramov na jednotlivých pozíciách. T. j. pre obidva algoritmy máme výsledky dvoch typov:

- váha konkrétneho ngramu v danom slove, napr. že váha ngramu *ing* bola 0.5
- váha ngramu na pozícii - napr. ngram na 7. pozícii mal váhu 0.3

Následne sme agregovali výsledky pre všetky ngramy, aj pre všetky pozície a tieto hodnoty normalizovali. Pre každý ngram a pozíciu nám už teda zostala iba jedna hodnota označujúca jeho dôležitosť.

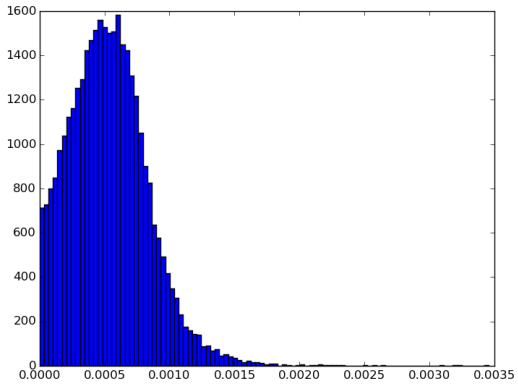
4.3 Analýza a výsledky

V tejto časti sa pozrieme na výsledky obidvoch algoritmov pre hľadanie najdôležitejších ngramov a porovnáme ich výsledky. Budeme sa zaoberať rôznymi aspektami modelu, aké typy ngramov sú dôležité, ako veľmi sa odlišujú výsledky oboch algoritmov a pod.

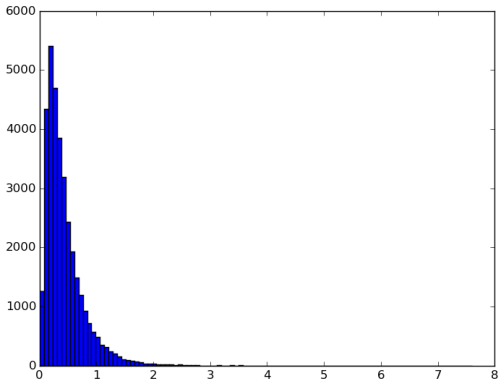
4.3.1 Porovnanie oboch modelov

Pozreli sme sa na koreláciu oboch modelov z obidvoch aspektov - použili sme na to Spearmanov korelačný koeficient. Ukazuje sa, že modely dosahujú podobné hodnoty, pokiaľ ide o pozíciu, ale rozdiely pre jednotlivé ngramy sú výraznejšie, aj keď stále majú pozitívnu koreláciu. Výsledky uvádzame v tabuľke 4.1.

4.3.2 Histogramy hustoty váh ngramov



Obr. 4.1: Rozloženie váh LIME algoritmu

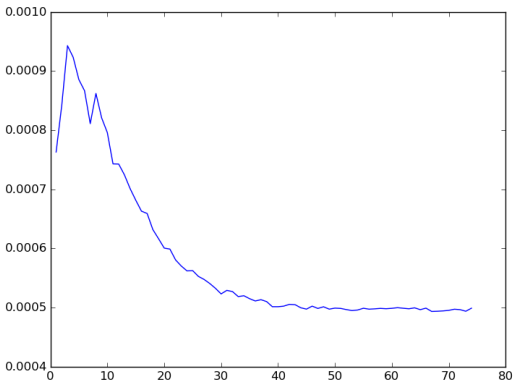


Obr. 4.2: Rozloženie váh subst. algoritmu

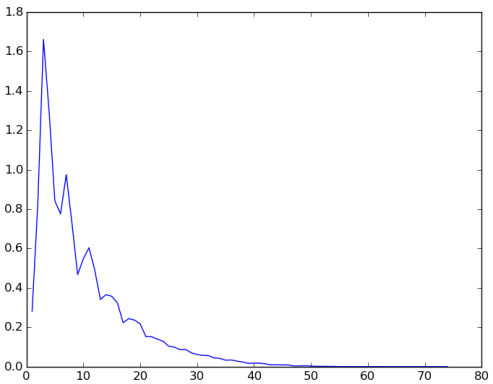
Pre obidva algoritmy sme si dali vykresliť histogramy hustoty váh jednotlivých ngramov. Tieto vizualizácie môžu byť zaujímavé preto, že by mohli vedieť jednoducho oddeliť veľmi dôležité ngramy od nedôležitých. To sa však ani pre jeden algoritmus nepotvrdilo, ako je možné vidieť na obrázkoch 4.1 a 4.2.

4.3.3 Rozloženie váh pozícií

Rovnako nás zaujímalo, ako veľmi dôležité sú ngramy na jednotlivých pozíciách. Pre obidva prístupy uvádzame graf, kde na osi x je pozícia ngramu a na osi y jeho váha.



Obr. 4.3: Váhy pozícií LIME algoritmu



Obr. 4.4: Váhy pozícií subst. algoritmu

Obidva algoritmy sa zhodujú v tom, že úplne najdôležitejší je 3. ngram - tento ngram zvyčajne zodpovedá 1. ngramu dĺžky 5. Teda malo by z toho vyplývať, že úplne najdôležitejšie sú v každom slove prvé 4 znaky spolu s informáciou, že ide o začiatok slova.

Pri LIME algoritme vidíme ešte druhé lokálne maximum na pozícii 8 (čo pravdepodobne zodpovedá druhému ngramu dĺžky 6). Pri substitučnom algoritme je ešte výrazné lokálne maximum na pozícii 7, ktorá reprezentuje 2. ngram dĺžky 5. Táto vzorka sa opakuje aj 3. krát, ďalšie výrazne maximum je pri 3. ngrame dĺžky 5. Z toho vyplýva, že substitučný algoritmus zjavne favorizuje ngramy dĺžky 5 zo začiatku slova.

4.3.4 Vlastnosti dôležitých ngramov

V tejto časti budeme analyzovať iba kvantitatívne výsledky pre jednotlivé ngramy (nie pozície).

Dĺžka ngramov

To, či sú ngramy nejakej konkrétnej dĺžky dôležitejšie, je zaujímavé sledovať preto, aby sme mohli prípadne viac zjednodušiť model (napr. odstrániť ngramy dĺžky 3), a samozrejme aj kvôli lepšiemu pochopeniu modelu.

Výsledok pre LIME algoritmus je zobrazený v tabuľke 4.2. Tento prístup mierne favorizuje výsledky pre ngramy dĺžky 3.

Dĺžka ngramu	**Počet**	**Suma váh**	**Priemer**
3	4881	3.10	0.000636
4	12941	7.09	0.000548
5	11139	5.55	0.000498
6	5509	2.59	0.000470

Tabuľka 4.2: Váhy jednotlivých dĺžok pre LIME algoritmus

Výsledky pre substitučný algoritmus sú v tabuľke 4.3. Substitučný algoritmus považuje za najdôležitejšie ngramy dĺžky 3, čo je v miernom rozpore s výsledkom z časti 4.3.3. Nemusí to tak však byť, tento výsledok môže naznačovať, že na daných pozíciách v slove sa už nemuseli nachádzať ngramy dĺžky 5 (napríklad neboli v slovníku), ale nachádzali sa tam už nasledujúce ngramy dĺžky 3.

Dĺžka ngramu	**Počet**	**Suma váh**	**Priemer**
3	4881	3528	0.72
4	12941	5297	0.41
5	11139	4561	0.41
6	5509	1574	0.29

Tabuľka 4.3: Váhy jednotlivých dĺžok pre substitučný algoritmus

Obidva algoritmy sa ale zhodujú v tom, že ngramy dĺžky 3 majú najväčšiu váhu vzhľadom na počet výskytov.

Pozície ngramov

V tejto časti ngramy rozdelíme do troch skupín:

- začiatočné - obsahujú začiatočnú značku ^
- koncové - obsahujú koncovú značku $
- stredové - neobsahujú žiadne značky

a rovnako ako v predošlej časti budeme analyzovať váhy jednotlivých skupín.

V tabuľke 4.4 je vidno, že najdôležitejšie v prepočte na jeden ngram sú začiatočné ngramy, ale celkovo najväčšiu váhu nesú ngramy v strede slova.

Pozícia	**Počet**	**Suma váh**	**Priemer**
začiatok	5799	3.50	0.000603
stred	24205	12.65	0.000523
koniec	4540	2.19	0.000482

Tabuľka 4.4: Váhy jednotlivých pozícií pre LIME algoritmus

Výsledky pre substitučný algoritmus uvádzame v tabuľke 4.5. Obidva algoritmy sa zhodujú v tom, že ngramy na začiatku majú jednotlivo najvyššiu váhu, ale celkovo najdôležitejšie sú ngramy v strede.

Pozícia	**Počet**	**Suma váh**	**Priemer**
začiatok	5799	4094	0.71
stred	24205	9035	0.37
koniec	4540	1831	0.40

Tabuľka 4.5: Váhy jednotlivých pozícií pre substitučný algoritmus

4.3.5 Substitúcie ngramov za nuly

Následne sme sa chceli pozrieť na to, ako ovplyvní náš najlepší (už natrénovaný) model, keď mu dáme k dispozícii iba obmedzenú množinu ngramov M, namiesto všetkých, ktoré má natrénované. Ak máme slovo w s ngramami $ng_1, ng_2, \ldots, ng_n$ a ich vektormi $v_{ng_1}, v_{ng_2}, \ldots, v_{ng_n}$ a napríklad vektor $v_{ng_i} \notin M$, tak tento vektor na vstupe vynulujeme. Konvolučná sieť prakticky neberie do úvahy nulové vektory, takže ďalšie výpočty budú prebiehať iba na vektoroch z množiny M.

Ako množinu M sme skúsili použiť 5000 a 10000 ngramov s najvyššími váhami. Takýto model však dosahoval rádovo horšie výsledky ako predošlé modely, preto sme usúdili, že takýmto spôsobom sa nedá počet dôležitých ngramov odhadovať.

4.3.6 100 najdôležitejších ngramov

Nakoniec sa ešte konkrétne pozrieme pre obidva algoritmy na 100 ngramov, ktoré majú podľa nich najvyššiu váhu.

LIME algoritmus:
^ist gmon ^wig ^nap ^jok ^kle ^crai ^club ^omn ^nih ^barto carbon dida leoni ^quint ocul har$ ^rip ^reo eplo galli gut doze parts$ insa ^coin ^pott ^pyrr pierre ^proph ^spor garn risb egy ^decor ^parr ^anna ingul hull ^bara ^squar nott ^cott ^whi ^preci ^blai sir$ ^pater ardle ansen$ ^lef uun ^intui ^leon egitim oui anett nshu ^chime elso ^pea ectar americ odynam enari ccent ^rebu dach ^kins ^bud obt depic roqu runa gane ecka ^can dou$ ^magne jew ^mous value ahon ^pear nstal relle ^grie ^dat dop eversi ibbl anarc ^flood ^polic arden anite$ ^descr null shun ^outc

Substitučný algoritmus:
^pain eup bc$ iv$ tp$ ^wing aa$ ^poly lux cc$ due$ god wet ai$ fur vi$ oe$ pm$ ^anti fl$ ei$ pope ^ward ^clan ez$ ^warm omt goa ^care pay jet da$ ^cell laid ^chip cay axe ^call ^cold cw$ ski we$ ^mr fee ^ferr aty ii$ ^ship wax ae$ job ^mall ^seed ^fran ye$ ^air ob$ ^love sap ^mind uice bay ^rain map ^wind ^rail ba$ ^trib pat$ ^snow ap$ ie$ ^fire ^herb mar$ ^east ^dish iry dru ^salt ^wear jew rug azz sex nn$ ^sold law zoo buy gas ^fish sax ^head ^mast ^camp ag$ ^sour ^firm pn$

Je veľmi zaujímavé, že pre substitučný algoritmus je väčšina slov zmysluplných, sú to ľahko indetifikovateľné časti nejakých slov. LIME algoritmus za najdôležitejšie považuje oveľa neštandardnejšie slová, aj keď aj tam sa dajú nájsť isté zjavné časti slov. Na základe tohto by sa dalo usudzovať, že lepšie funguje substitučný algoritmus. Či je to naozaj tak, ešte skúsime otestovať v časti 4.4.

4.3.7 Porovnanie s najčastejšími ngramami

Pre daných 10000 slov sme vyextrahovali 10000 najčastejších ngramov, ktoré sa v nich vyskytovali. Porovnali sme túto množinu s 10000 najdôležitejšími ngramami podľa obidvoch algoritmov. Vysvitlo, že tieto množiny sú značne odlišné. LIME algoritmus vyberie medzi 10000 najdôležitejších iba 4475 najčastejších ngramov. Substitučný algoritmus ich vyberie 4888.

4.4 Model založený na najdôležitejších ngramoch

V tejto časti predstavíme návrh modelu, ktorý bude založený na menšom počte ngramov ako modely z predošlej kapitoly. Keďže prístup s nulovaním ngramov z časti 4.3.5 nefungoval, nevieme jednoducho odhadnúť, koľko ngramov by mohol potrebovať.

Obmedzený počet ngramov sme preto skúsili nastaviť na 10000. Na rozdiel od modelu v predošlej kapitole, ktorý sa naučil vektorovú reprezentáciu všetkých ngramov, ktoré sa vyskytujú častejšie ako 5-krát, tento model dostane ngramy pevne pridelené. Všetky ostatné ngramy v slovách sa budú ignorovať podľa bežného postupu konkrétneho modelu.

Z obidvoch algoritmov (LIME aj substitučného) sme vyextrahovali 10000 ngramov s najväčšou normalizovanou váhou a na nich sme natrénovali náš najlepší model - `NgramConvBeginFirstValidModel`.

Motivácia pre tento model je ďalšie zníženie počtu parametrov modelu, a tým zrýchlenie a zjednodušenie jeho trénovania. Zároveň nás zaujíma aj odpoveď na otázku, či je vôbec takýto model schopný dostatočne úspešne slová reprezentovať.

4.4.1 Výsledky

Oba modely, LIME aj substitučný sme natrénovali na 20 miliónoch tokenov. V tabuľke uvádzame výsledky pre klasickú úlohu s podobnosťou slov definovanou v časti 1.3.6. Zároveň uvádzame v tabuľke aj výsledky pre pôvodný `NgramConvBeginFirstValidModel` aj počty parametrov jednotlivých modelov. Tieto hodnoty rovnako ako v predošlej kapitole nezahŕňajú parametre pre výstupné vektory - 9 144 900 parametrov. Ako sme vysvetlili v časti 3.3.2, tento počet nie je až taký dôležitý.

Názov triedy modelu	Počet parametrov modelu	EN-WS353 20 mil.	EN-RW 20 mil.
NgramConvBeginFirstValidModel	5 921 700	39.0	28.5
LIME model	1 740 100	23.5	17.1
Substitučný	1 740 100	36.3	22.2

Tabuľka 4.6: Výsledky LIME a substitučného modelu

Ako je možné vidieť v tabuľke 4.6, substitučný model dosahoval porovnateľné výsledky s pôvodným `NgramConvBeginFirstValidModel`, pričom oproti nemu používa asi tretinu parametrov.

Záver

V našej práci sme sa zaoberali návrhom a implementáciou nového modelu tvorby slovných vektorov využívajúcom morfológiu. Podarilo sa nám navrhnúť, implementovať a otestovať model využívajúci ngramy, ktorý dosahuje porovnateľné výsledky s už existujúcimi modelmi, a zároveň spĺňa nami požadované vlastnosti - nižší počet parametrov a schopnosť predpovedať slovné vektory pre predtým nevidené slová.

Náš model sa zo začiatku trénuje lepšie ako nami natrénovaný klasický Word2Vec model pre zriedkavé slová, a preto rýchlejšie dosahuje použiteľné výsledky. Tento Word2Vec model potrebuje až 9144900 parametrov pri vytváraní slovných vektorov, zatiaľ čo náš najlepší ngramový model ich potrebuje iba 5921700. Slušné výsledky dosahuje ngramový model aj s obmedzeným počtom ngramov - 10000. Tieto ngramy boli získané pomocou nášho substitučného algoritmu. Zároveň náš oklieštený model používa iba 1740100 parametrov, čo je výrazne menej ako nami implementovaný Word2Vec model. Pri následnom použití by preto mohol dopomôcť k rýchlejšiemu trénovaniu zložitejších modelov.

Výhodou nášho modelu je, že vďaka ngramom implicitne využíva morfologickú informáciu z jazyka, ale pri trénovaní nepotrebuje žiadnu dodatočnú informáciu o morfológii daného jazyka okrem čistého textu. Toto rovnako platí aj pre substitučný model s oklieštením počtom parametrov. Obidva modely sú preto automaticky prenosné do množstva jazykov.

Vďaka tomu, že sa náš model automaticky učí morfologické informácie o danom jazyku namiesto toho, aby takéto informácie dostával na vstupe, dal by sa pravdepodobne v budúcnosti využiť na extrakciu morfologických informácií z textu, napríklad na hľadanie predpôn a prípon v texte, prípadne na vytvorenie automatického stemmera pre daný jazyk.

Prekvapením pre nás bolo, že napríklad Min-Max alebo GRU model navrhnutý v 3. kapitole dosahoval horšie výsledky, ako sa očakávalo. Analýza týchto výsledkov by mohla prípadne priniesť nové poznatky o tvorbe slovných vektorov.

Literatúra

[1] Morfologická anotácia textov slovenského národného korpusu. "http://korpus.juls.savba.sk/morpho.html".

[2] Texty anglickej wikipédie na stiahnutie. https://meta.wikimedia.org/wiki/Data_dumps.

[3] Wikipédia extractor. https://github.com/attardi/wikiextractor.

[4] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[5] Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20, pages 161–168. NIPS Foundation (http://books.nips.cc), 2008.

[6] Danqi Chen and Christopher D Manning. A fast and accurate dependency parser using neural networks. In *EMNLP*, pages 740–750, 2014.

[7] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[8] François Chollet. Keras. https://github.com/fchollet/keras, 2015.

[9] Wikimedia Commons. Illustration of gradient descent on a series of level sets. "https://en.wikipedia.org/wiki/Gradient_descent#/media/File:Gradient_descent.svg", 2012.

[10] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. *arXiv preprint arXiv:1612.08083*, 2016.

[11] Ladislav Dvonč, Gejza Horák, František Miko, Jozef Mistrík, Ján Oravec, Jozef Ružička, and Milan Urbančok. *Morfológia slovenského jazyka*. Vydavateľstvo Slovenskej akadémie vied, 1966. http://www.juls.savba.sk/ediela/msj/.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[13] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610, 2005.

[14] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *AISTATS*, volume 1, page 6, 2010.

[15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[16] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 1188–1196, 2014.

[17] Wang Ling, Tiago Luís, Luís Marujo, Ramón Fernandez Astudillo, Silvio Amir, Chris Dyer, Alan W Black, and Isabel Trancoso. Finding function in form: Compositional character models for open vocabulary word representation. *arXiv preprint arXiv:1508.02096*, 2015.

[18] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.

[19] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[20] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM, 2016.

[21] Xin Rong. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*, 2014.

[22] Herbert Rubenstein and John B. Goodenough. Contextual correlates of synonymy. *Commun. ACM*, 8(10):627–633, October 1965.

[23] DE Rumelhart. Learning internal representations by error propagation. *Nature*, 323:533–536, 1986.

[24] Magnus Sahlgren. An introduction to random indexing. In *In Methods and Applications of Semantic Indexing Workshop at the 7th International Conference on Terminology and Knowledge Engineering, TKE 2005*, 2005.

[25] Radu Soricut and Franz Josef Och. Unsupervised morphology induction using word embeddings.

[26] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

[27] Minh thang Luong, Richard Socher, and Christopher D. Manning. Better word representations with recursive neural networks for morphology.

[28] Peter Wiemer-Hastings, K Wiemer-Hastings, and A Graesser. Latent semantic analysis. In *Proceedings of the 16th international joint conference on Artificial intelligence*, pages 1–14. Citeseer, 2004.

Appendix A - zdrojový kód

Zdrojový kód k diplomovej práci, popis spustenia a odkaz na testovacie dáta je dostupný na GitHube: `https://github.com/mixie/word_vectors/`