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

Bakalářská práce

Využití neuronových sítí v sémantice slov

Plzeň 2018 Hoang Ngoc Hung

Místo této strany bude zadání práce.

Prohlášení

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

V Plzni dne 2. května 2018

Hoang Ngoc Hung

Poděkování

Rád bych poděkoval vedoucímu bakalářské práce Ing. Michalu Konkolovi, Ph.D. za odborné vedení, cenné rady a čas, který mi věnoval při řešení dané problematiky.

Abstract

This thesis is focused on semantic models especially on *skip-gram* method from *word2vec* library. For this method, an extension based on syllabifying an input word was introduced. Improved Lansky algorithm was used to syllabify words. The created model was tested on standard data in English language but it did not achieve better results than the original *skip-gram* method. This thesis proved that syllables can be used to create semantic models.

Abstrakt

Tato práce se věnuje sémantickým modelům a především metodě *skip-gram* z knihovny *word2vec*. Pro tuto metodu bylo vytvořeno rozšíření založené na rozdělení vstupního slova na slabiky. K slabikování byl použit *vylepšený Lánského algoritmus*. Práce byla otestována na standardních datech v anglickém jazyce. Rozšíření nedosáhlo lepších výsledků než původní metoda *skip-gram*, ale prokázalo se, že i slabiky se dají použít k vyjádření sémantiky.

Obsah

1 Úvod

V dnešní době žijeme v rozmachu informačních a výpočetních technologií. Lidé denně využívají k práci počítače, chytré telefony apod. a komunikaci s těmito zařízeními si představují v co možná nejjednodušší formě. Tou je pro lidi přirozená řeč. Aby bylo možné přirozenou řeč zpracovávat, musí jí počítač nejprve porozumět.

Sémantická analýza slov je jednou z hlavních úloh zpracování přirozeného jazyka (NLP). Pomocí reprezentace významu slov můžeme zkoumat podobnosti slov. Tyto reprezentace je dále možné použít v dalších úlohách NLP, jakými jsou např. strojový překlad (angl. *machine translation*) a rozpoznávání pojmenovaných entit (angl. *named entity recognition*) [4]. První pokusy o vytvoření reprezentací slov, kterým by rozuměly počítače, proběhly bez úspěchu již během minulého století. Chyběl jim dostatečný výpočetní výkon a především data, se kterými mohly pracovat. Dnes již tyto problémy nemáme, výpočetní výkon je veliký a dat je díky *Internetu* obrovské množství.

Drtivá většina algoritmů pro tvorbu reprezentace sémantického významu používá statistiku výskytů slov v různých kontextech. Mapují slova do vektorového prostoru takovým způsobem, že podobná slova se nacházají blízko sebe. Knihovna *word2vec*, která patří mezi nejúspěšnější, byla vytvořena týmem vedena Tomášem Mikolovem [10]. Její úspěšnost je založena na jednoduché architektuře neuronové sítě a velice rychlému učení sítě, díky kterému je možné použít veliké množství trénovacích dat. Tato práce se zabývá rozšířením této knihovny.

V následující kapitole jsou uvedeny základní teoretické znalosti v oblasti umělých neuronových sítí. V kapitole *Sémantické modely* je uvedeno několik sémantických modelů a podrobně popsána knihovna *word2vec*. Kapitola *Evaluace sémantických modelů* popisuje různé způsoby vyhodnocení úspěšnosti sémantických modelů. V kapitole *Implementace a optimalizace skip-gramu* jsou popsány jednotlivé optimalizační prvky metody *skip-gram*. V kapitole *Navržený model* je popsáno rozšíření metody *word2vec*. Kapitola *Experimenty* obsahuje popis jednotlivých experimentů a jejich výsledky. Na závěr jsou shrnuty výsledky práce.

2 Umělá neuronová síť

Umělá neuronová síť je matematickým modelem biologických struktur neuronů u živých organismů. Skládá se z umělých neuronů, které jsou propojeny pomocí synapsí a uspořádany do vrstev. Každý umělý neuron má pouze jeden výstup a libovolný počet vstupů.

Každý neuron je propojen s neurony v předchozí vrstvě, které mu předávají signál. Spojení mezi neurony mají váhy (tzv. synaptické váhy), které se během učení sítě upravují, aby síť dosahovala co nejlepších výsledků.

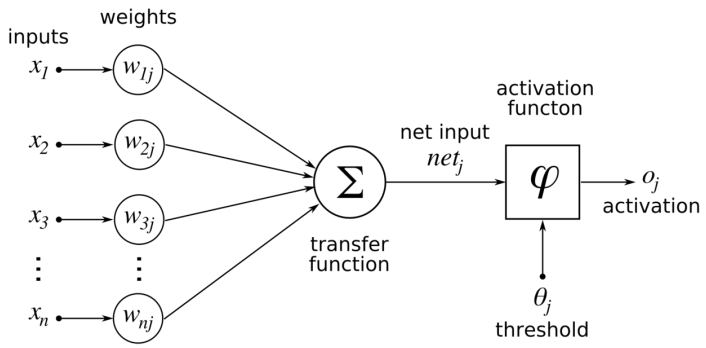
Umělá neuronová síť je vhodná na úlohy z oblasti klasifikace, aproximace a predikace.

2.1 Model umělého neuronu

Existuje celá řada modelů, které umělý neuron popisují. Jedním z nejpoužívanějších je McCulloch-Pitsův model neuronu, viz obr. 2.1. Jednotlivé neurony jsou popsány uspořádanou trojicí

$$(w, \varphi, \Theta),$$

kde $w = (w_1, .., w_n) \in \mathbb{R}$ je vektor synaptických vah, $\varphi : \mathbb{R} \longrightarrow \mathbb{R}$ je aktivační funkce neuronu a $\Theta \in \mathbb{R}$ je práh [15].



Obrázek 2.1: Model umělého neuronu

Vstup je reprezentován vektorem $x = (x_1, .., x_n) \in \mathbb{R}$, každá složka vektoru má vlastní váhu, která ji přisuzuje významnost. Vážená suma složek vektoru

společně s prahem tvoří *vnitřní potenciál* neuronu ξ, vypočítán podle (2.1).

$$\xi = \sum_{i=1}^{n} w_i x_i + \theta \tag{2.1}$$

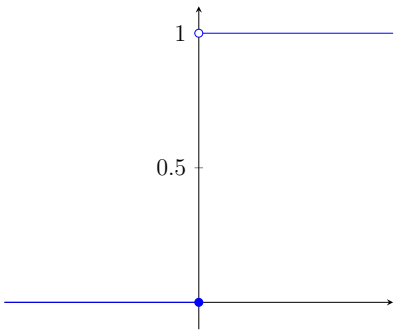
Výstup neuronu je získán použitím aktivační funkce na vnitřní potenciál neuronu dle rovnice (2.2) [20].

$$o = \varphi(\sum_{i=1}^{n} w_i x_i + \theta) \tag{2.2}$$

2.2 Aktivační funkce

Aktivační funkce mohou být různé pro různé úlohy, volí se na základě povahy vstupních dat. V nejjednodušší podobě je aktivační funkce binární, neuron se může nacházet buď v pasivním nebo aktivním stavu. Touto funkcí může být např. Heavisideova (skoková) funkce, obr. 2.2, definovaná předpisem (2.3).

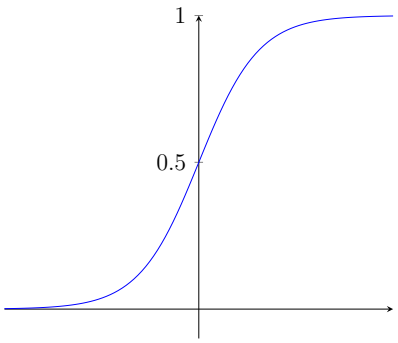
$$f(x) = \begin{cases} 0 & pro\, x < 0 \\ 1 & pro\, x \geq 0 \end{cases} \tag{2.3}$$



Obrázek 2.2: Heavysideova funkce (skoková)

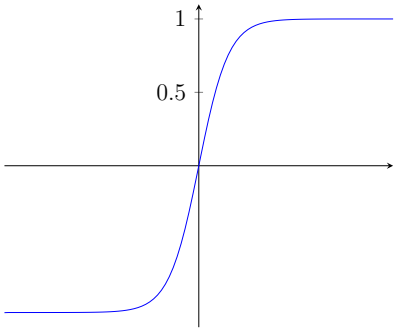
Častěji jsou ale voleny funkce spojité, které jsou diferenciovatelné na celém oboru reálných čísel. Derivace je vyžadována kvůli algoritmu **zpětného šíření chyby**, který je popsán v sekci 2.3.1. Příkladem takové funkce je logistická funkce (sigmoida) definovaná předpisem (2.4), obr. 2.3, nebo hyperbolický tangens (2.5), obr. 2.4.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.4}$$



Obrázek 2.3: Logistická funkce (sigmoida)

$$f(x) = tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{2.5}$$



Obrázek 2.4: Hyperbolický tangens

2.3 Učení neuronové sítě

Proces učení (trénování) je ve své podstatě optimalizační proces, který najde pro danou trénovací množinu dat takovou konfiguraci synaptických vah a prahů, kdy je průměrná chyba sítě přes celou trénovací množinu dat minimální.

Používají se dvě strategie učení sítě a to:

- učení s učitelem, kdy se síť snaží snížit rozdíl mezi požadovaným výstupem a aktuálním výstupem konfigurací vah sítě [7],
- učení bez učitele, kdy síť nezná požadovaný výstup.

2.3.1 Algoritmy učení neuronové sítě

Nejčastěji používaným algoritmem k učení umělých neuronových sítí je algoritmus zpětného šíření (angl. *backpropagation algorithm*). Používá se k učení vícevrstvých sítí se spojitou nelineární diferencovatelnou aktivační funkcí (např. sigmoida, viz kapitola 2.2). Algoritmus minimalizuje **chybovou funkci** (angl. *cost function*), a to prostřednictvím adaptace synaptických vah. Patří do skupiny gradientních metod.

Existuje celá řada chybových funkcí. Chybová funkce je rovna nejčastěji střední kvadratické chybě (angl. *MSE - mean squared error*) mezi požadovaným a skutečným výstupem, příkladem takové funkce je

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(t_i - y_i)^2 = \sum_{i=1}^{n}\frac{e_i^2}{n}, \tag{2.6}$$

kde n je počet trénovacích vzorů, t_i je očekávaný výstup, y_i je výstup sítě a e_i je chyba sítě na i-tém vzoru.

Dalším možnou chybovou funkcí je *cross-entropy*, která je definována takto:

$$CE(y, \hat{y}) = -\sum_{i=1}^{V} y_i log(\hat{y}_i), \tag{2.7}$$

kde y je vektor požadovaných hodnot a $\hat{y}$ je vektor predikcí. K minimalizaci chybové funkce se používají gradientní metody [19].

Algoritmus má tři etapy

1. dopředné (angl. *feedforward*) šíření signálu tréninkového vzoru,
2. zpětné šíření chyby,
3. oprava prahů a synaptických vah.

Po průchodu je výstup porovnán s požadovanou hodnotou, je spočítaná chyba, a ta se zpětně přepočítává do předchozích vrstev a synaptické váhy jsou opraveny [19]. Proces se iterativně opakuje dokud není chyba menší než požadovaná hodnota nebo se nepřesáhl počet iterací.

2.3.2 Problémy s učením

Jednou z nevýhod neuronové sítě je pomalý proces trénování a obrovské množství trénovacích dat, které je třeba k naučení sítě. Dalšími jsou *přeučení*

a *nedoučení* sítě.

Přeučení

Při učení sítě může dojít k efektu, kterému se říká přeučení (angl. *overfitting*), kdy síť efektivně minimalizovala cenovou funkci na trénovací množině, ale ztratila schopnost generalizace. Přeučená síť neumí zobecňovat a selhává na validačních datech (data nenacházející se v trénovací množině).

Nedoučení

Stav, kdy se síť naučí s velkou hodnotou chybové funkce, se nazývá nedoučení (angl. *underfitting*). Takto naučená síť dosahuje špatných výsledků jak na trénovací množině, tak na množině validační.

3 Sémantické modely

V této kapitole budou popsány výpočetní modely patřící do oblasti nazývané *distribuční sémantika*, která se zabývá metodami vytvářejících reprezentace sémantického významu slov z velkého objemu dat.

Modely jsou založené na *distribuční hypotéze*, která říká: „slovo můžeme charakterizovat pomocí slov nacházejících se v jeho okolí“. Slova sémanticky si podobná by se tedy měly nacházet v podobných kontextech. Tyto výpočetní modely obvykle shromažďují statistiky ohledně kontextu pro každé slovo. Shromážděné statistky se využívají k vytvoření mnohorozměrných vektorů reprezentující sémantický význam jednotlivých slov. Podobnost dvou slov se díky tomu jednoduše zjistí porovnávním jejich vektorů pomocí vzdálenosti nebo podobnosti vektorů (viz kapitola 4.1) [4].

3.1 HAL

Hyperspace Analogue to Language (HAL) [9] je jedním ze základních sémantických modelů. Model HAL využívá dvou důležitých předpokladů. Prvním je rozdíl mezi pravým a levým kontextem, proto statistiky těchto kontextů musí být oddělené. Druhým předpokladem je význam vzdálenosti mezi slovy (ve větě), vzdálenější slova mají menší vliv na sémantický význam než slova bližší.

Model vytváří matici sousednosti M o rozměrech $|W| \times |W|$, kde $|W|$ je počet unikátních slov v korpusu. Na pozici $m_{i,j}$ se nachází hodnota, která vyjadřuje úroveň společného výskytu slov w_i a w_j, přesněji výskyt slova w_j v levém kontextu slova w_i a výskyt slova w_i v pravém kontextu slova w_j [4]. Pro každé slovo v tréninkovém korpusu se prozkoumává jeho okolí o fixní velikosti. Každý výskyt slova w_j v levém kontextu slova w_i přičte hodnotu váhové funkce $f(\Delta k)$, kde Δk je vzdálenost mezi slovy, na pozici $m_{i,j}$. Příkladem váhové funkce může být (3.1).

$$f(x) = |window| - x \tag{3.1}$$

Vektorovou reprezentaci slova můžeme získat spojením sloupcového a řádkového vektoru daného slova. Takový vektor má příliš velkou dimenzi,

a proto se využívají různé techniky redukce dimenze. Většina slov se společně nevyskytuje, proto je matice M velice řídká.

3.2 LSA

Latentní sémantická analýza (LSA) (angl. *latent semantic analysis*) vytváří vektorovou reprezentaci slova na základě vztahu mezi množinou dokumentů a slovy, která se v dokumentech nachází. Dokumentem může být věta, odstavec nebo jakýkoliv jiný logický celek.

LSA nejdříve sestaví matici A, kde každé unikátní slovo je reprezentováno jednou řádkou a každý sloupec jeden dokument. Na pozici i a j se nachází počet výskytů slova w_i v dokumentu d_j [5]. Hodnoty frekvencí slov jsou váženy, obvykle se využívá metoda *tf-idf*.

Metoda *tf-idf* ohodnocuje slova podle důležitosti pro daný dokument. Výpočet relevance slova se vypočítá pomocí hodnot *tf* (angl. *term frequency*) a *idf* (angl. *inverse document frequency*). Hodnota *tf* vyjadřuje, jak často se slovo vyskytuje v dokumentu. Obvykle se tato hodnota normalizuje, aby nedocházelo k nadhodnocování dlouhých dokumentů, ve kterých se hledaný výraz může vyskytovat častěji než v kratších. Četnost slova získáme takto:

$$tf_{i,j} = \frac{n_{i,j}}{\sum_{k=1}^{N} n_{k,j}}, \tag{3.2}$$

kde $n_{i,j}$ je počet výskytů slova t_i v dokumentu d_j o N slovech.

Relevance slova je vyjádřená hodnotou *idf*. Vyskytuje-li se často slovo v dokumentech, nenese tolik informace a není tak důležité jako slova vyskytující se méně. Hodnotu důležitosti slova spočítáme:

$$idf_i = log\frac{|D|}{|\{j : t_i \in d_j\}|}, \tag{3.3}$$

kde |D| je velikost množiny dokumentů a $|\{j : t_i \in d_j\}|$ je počet dokumentů, které obsahují slovo i. Výsledná hodnota *tf-idf* se získá vynásobením hodnot *tf* a *idf*.

Matice A má obvykle velkou dimenzi, a proto se dále pomocí metody rozkladu na singulární čísla (angl. *singular value decomposition* - SVD) redukuje

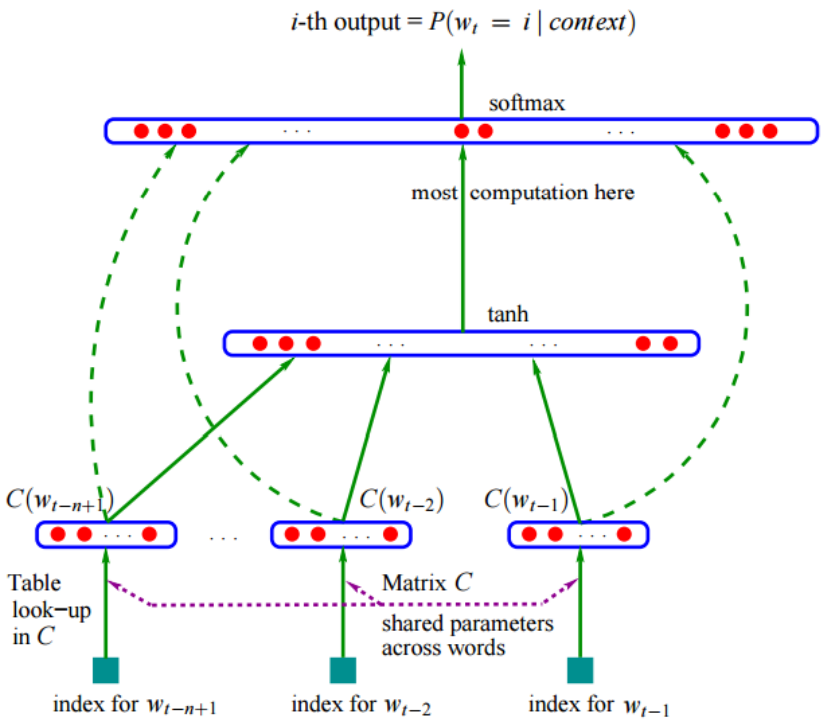
dimenze. SVD rozloží matici A na tři matice dle (3.4).

$$A = U\Sigma V^T, \tag{3.4}$$

kde U a V jsou ortogonální matice a Σ je diagonální matice s nezápornými reálnými čísly na diagonále. Sémantický význam slova je reprezentován řádkou pro dané slovo v matici A.

3.3 RNN model

Prvním ze sémantických modelů, který se používal k vytváření mnohorozměrných vektorů k reprezentaci významu slov, byl jazykový model využívající *rekurentní neuronové sítě* [1], dále RNN. Výhoda této architektury sítě je možnost využívat vnitřní stav, tzv. paměť, ke zpracování sekvence vstupů. Základní model RNN je na obr. 3.1.



Obrázek 3.1: Rekurentní neuronová síť, obrázek pochází z [1]

RNN podávalo velice dobré výsledky i přes malé množství trénovacích dat.

Učení sítě je ale výpočetně velice náročné, proto se jiné architektury neuronových sítí osvědčili lépe.

3.4 Unified Neural Network

V této sekci bude popsána architektura konvoluční neuronové sítě navržena Ronanem Collobertem [2]. Jedná se o síť, která na vstupu přijímá větu a na výstupu jsou *part-of-speech* tagy, tagy pojmenovaných entit, sémantické role, sémantická podobnost slov a pravděpodobnost, že věta dává smysl (gramaticky a sémanticky) daná jazykovým modelem. Tato architektura tedy optimalizuje několik úloh najednou. Celá síť se učí na všech úkolech společně.

Part-of-speech tagging Úloha spočívá v přiřazování značek slovních druhů jednotlivým slovům ve větách. Těmto značkám se říká *POS tag*, který může obsahovat informace o slovním druhu, a další informace, např. u slovesa jeho čas, osobu, vid.

Rozpoznání pojmenovaných entit Rozpoznání pojmenovaných entit (angl. *named entity recognition*) má za úkol nalezení a klasifikování pojmenovaných entit, které byly v textu zmíněny, do předem definovaných kategorií jako jsou osoby, organizace, lokace.

Jazykový model Jazykový model určuje u jednotlivých slov a jejich spojení pravděpodobnost výskytu v textu. Jazykový model vždy pracuje s konkrétním jazykem, který má konkrétní slovník a vlastní soubor pravidel. Síť se snaží určit gramatickou a sémantickou správnost tri-gramů, což je posloupnost tří po sobě jdoucích slov.

Značkování sémantických rolí Značkování sémantických rolí (angl. *semantic role labeling*) je proces, který přiřazuje značky slovům nebo frázím ve větách podle jejich sémantického významu. Je založen na nalezení větných členů souvisejících s přísudkem ve větě a zařazení těchto členů do jednotlivých rolí.

Sémantický vztah slov Cílem je určit, jestli dvě slova mají mezi sebou sémantický vztah (synonyma, holonyma, atd.).

3.5 GloVe

Globální vektory (angl. *global vectors*) [12] je jednou z nejnovějších metod pro vytváření mnohorozměrných vektorů. GloVe podobně jako HAL sbírá statistiky výskytu slova v celém trénovacím korpusu a zkoumá slova, která se vyskytují v podobném kontextu. Model se vytváří pomocí matice sousednosti.

Během učení se využívá log-bilineární regresní model tak, aby skalární součin vektorů byl roven pravděpodobnosti společného výskytu.

3.6 Word2Vec

Knihovna *Word2Vec* vytváří ze slov mnohorozměrné vektory pomocí neuronových sítí. Je velice populární díky její rychlosti trénování, která je výrazně rychlejší než další metody založené na neuronových sítí. Dá se tak trénovat na větším objemu dat, což vede k vyšší kvalitě reprezentace slov. Vysoce optimalizovaný kód dovoluje vytvoření modelu z 6 miliard slov s dimenzí vektoru 1000 za pouhé dva dny [10]. Žádná z předchozích metod nebyla schopna při trénování přesáhnout hranici dimenze vektoru o velikosti 100, proto metoda *word2vec* dosahuje výrazně lepších výsledků než její předchůdci.

Natrénované vektory v sobě nesou informace o některých lingvistických jevech. Jedním z těchto jevů je také analogie slov (angl. *word analogy*), viz kapitola 4.3.

Knihovna *word2vec* má dvě různé architektury sítě, které budou popsány v následujících kapitolách.

3.6.1 Continuous bag-of-words

Continuous bag-of-words, dále CBOW, je jednou z architektur knihovny *word2vec*. Model se na základě předem dané velikosti okna kontextu snaží určit slovo aktuální (střední). Architektura se nazývá *bag-of-words*, protože pořadí slov v kontextovém okně neovlivňuje výslednou projekci [10]. Architektura sítě je naznačena na obr. 3.2.

Projekční (skrytá) vrstva h je vektor, který se vypočítá pomocí takto:

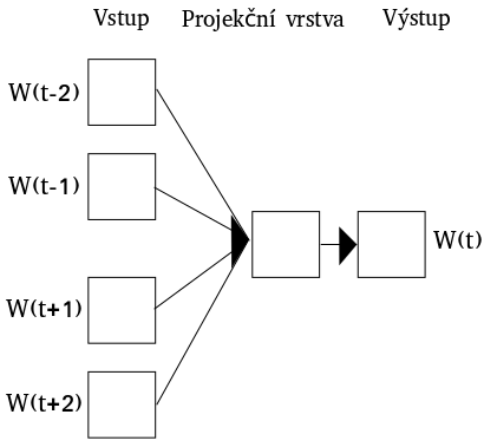
$$h = \frac{1}{C} \cdot (v_{w_1} + v_{w_2} + ... + v_{w_C}), \tag{3.5}$$

kde v_w jsou vektorý vstupních slov.

Výpočet skóre u_j slova w_j ze slovníku V se počítá dle (3.6).

$$u_j = v'^T_{w_j} h, \tag{3.6}$$

kde v'_{w_j} je výstupní vektor slova w_j.



Obrázek 3.2: Architektura sítě metody *continuous bag-of-words*

Pravděpodobnost výstupního slova w_j podmíněného kontextem o šířce C získáme aplikováním funkce *softmax* dle (3.7) [16].

$$y_j = p(w_j | w_{I,1}, w_{I,2}, .., w_{I,C}) = \frac{exp(u_j)}{\sum_{j'=1}^{V} exp(u_{j'})} \tag{3.7}$$

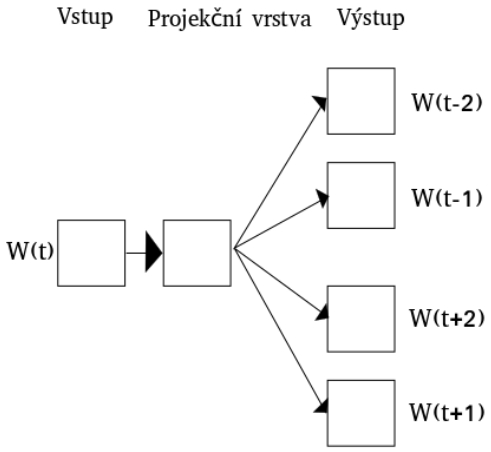
Složitost učení této architektury je

$$Q = N \times D + D \times log_2(V), \tag{3.8}$$

kde N je velikost kontextového okna, D je velikost dimenze vektoru slov a V je velikost slovníku.

3.6.2 Skip-gram

Druhou možnou architekturou je *skip-gram*, který je velice podobný CBOW, ale namísto určování středního slova v kontextovém okně se snaží předpovědět kontext podle aktuálního slova. Achitektura skip-gram je základem této práce, a proto bude popsána podrobněji než architektura předchozí. Architektura sítě je naznačena na obr. 3.3.



Obrázek 3.3: Architektura sítě metody *skip-gram*

Na vstupu je pouze jedno slovo, projekční vrstva h se vypočítá dle (3.9).

$$h = v_{w_I} \tag{3.9}$$

Pravděpodobnost předpovězení kontextu o šířce C dle středního slova w_I je:

$$p(w_{c,j} = w_{O,c}|w_I) = y_{c,j} = \frac{exp(u_{c,j})}{\sum_{j'=1}^{V} exp(u_{j'})}, \tag{3.10}$$

kde w_{c_j} je j-té možné slovo na pozici c v kontextu, $w_{O,c}$ je správné slovo na pozici c, w_I je vstupní slovo, y_{c_j} je výstup slova w_j na pozici c. Jelikož výstupní vrstva sdílí váhy, pro skóre slov platí (3.11) [16].

$$u_{c,j} = u_j = {v'_{w_j}}^T \cdot h, \text{ pro } c = 1, 2, \ldots, C, \tag{3.11}$$

kde ${v'_{w_j}}^T$ je výstupní vektor slova w_j.

Chybová funkce modelu je

$$E = -\sum_{c=1}^{C} u_{j_c^*} + C \cdot log \sum_{j=1}^{V} exp(u'_j), \tag{3.12}$$

kde j_c^* je index správného slova na pozici c v kontextu. Zderivujeme podle $u_{j_c^*}$ a dostáváme

$$\frac{\partial E}{\partial u_{j_c^*}} = y_j - t_j = e_j, \tag{3.13}$$

kde $t_j = 1$ pouze tehdy, když j-té slovo je správné slovo kontextu, jinak $t_j = 0$. Pro gradientní sestup vah matice W', která je maticí vah výstupní vrstvy, platí rovnice (3.14).

$$w_{ij}^{'(new)} = w_{ij}^{'(old)} - \eta \cdot \sum_{c=1}^{C} (y_{c,j} - t_{c,j}) \cdot h_i \tag{3.14}$$

Výsledný gradientní sestup na adaptaci vstupních synaptických vah je definován vztahem (3.15).

$$w_{ij}^{'(new)} = w_{ij}^{'(old)} - \eta \cdot \sum_{j=1}^{V} \sum_{c=1}^{C} (y_{c,j} - t_{c,j}) \cdot w'_{ij} \cdot x_j \tag{3.15}$$

Jak je vidět z rovnice, gradientní sestup je výpočetně velice náročný, protože počítá sumu přes celou velikost slovníku. K zrychlení výpočtu byly navrhnuty techniky *hierarchický softmax* a *negativní vzorkování*, který bude popsán v další kapitole. Složitost učení architektury je

$$Q = C \times (D + D \times log_2(V)), \tag{3.16}$$

kde C je maximální vzdálenost slov. Pro každé trénovací slovo se vybírá náhodné číslo R z intervalu $\langle 1; C \rangle$. Navyšování maximální vzdálenosti slov vede k větší kvalitě natrénovaných vektorů, ale zvyšuje se tím výpočetní složitost.

3.6.3 Negativní vzorkování

Učení neuronové sítě je kvůli velikosti slovníku výpočetně náročný proces. Technika *negativního vzorkování* [11] (angl. *negative sampling*) tento problém řeší tak, že se pro jeden tréninkový vzor neadaptují všechny synaptické váhy, ale pouze malé procento z nich.

Během jedné iterace se upraví váhy pro aktuální výstupní slovo, tzv. *pozitivní vzorek* (angl. *positive sample*), a pro předem daný počet slov různých od pozitivního vzorku, tzv. *negativní vzorky* (angl. *negative samples*). Ve vstupní vrstvě sítě se aktualizují váhy pouze pro vstupní slovo.

Výběr negativních vzorků se dělá pomocí pravděpodobnostího rozdělení značené $P_n(w)$, které se nazývá rozdělení šumu. Knihovna *word2vec* používá modifikované unigramové rozdělení. Pravděpodobnost vyběru slova jako negativní vzorek se vypočítá dle (3.17).

$$P(w_i) = \frac{f(w_i)^{\frac{3}{4}}}{\sum_{j=0}^{n} f(w_j)^{\frac{3}{4}}}, \tag{3.17}$$

kde $f(w_i)$ je frekvence slova w_i. Více frekeventovaná slova mají tedy větší pravděpdobnost se stát negativním vzorkem.

Při využití techniky negativního vzorkování vypadá chybová funkce následovně

$$E = -log\ \sigma({\mathbf{v}'_{w_0}}^T \cdot \mathbf{h}) - \sum_{w_j \in W_{neg}} log\ \sigma(-{\mathbf{v}'_{w_j}}^T \cdot \mathbf{h}), \tag{3.18}$$

kde w_O je aktuální výstupní slovo (pozitivní vzorek), $\mathbf{v}'_{w_O}$ je jeho výstupní vektor, $\mathbf{h}$ je výstup projekční vrstvy a $W_{neg} = \{w_j | j = 1, \ldots, K\}$ je množina slov (negativní vzorky) vybraná pravděpodobnostním rozdělením $P_n(w)$ [16].

3.6.4 Podvzorkování

Jednou z dalších možných technik k zefektivnění procesu učení je *podvzorkování* (angl. *subsampling*). Zrychluje proces učení zmenšením množství trénovacích vzorů předložených neuronové síti. Některá slova se mohou vyskytovat ve velkých korpusech až milionkrát (např. spojky „i“ a „a“). Tato slova poskytují méně informační hodnoty než slova méně se vyskytující. Například společný výskyt slov „Francie“ a „Paříž“ poskytuje modelu mnohem větší přínos než společný výskyt slova „Francie“ a spojky „a“, protože se téměř každé slovo s touto spojkou ve větě vyskytne. Vektorová reprezentace frekventovaných slov se po více jak milionech tréninkových vzorech výrazně nemění [11].

Nerovnováha mezi neobvyklými a frekventovanými slovy se tedy vyvažuje

pomocí podvzorkování. Každé slovo w_j se zahodí s pravděpodobností vypočtené podle vzorce (3.19).

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}, \tag{3.19}$$

kde $f(w_i)$ je četnost výskytu slova w_i a t je zvolený práh, obvykle se volí hodnota 10^{-5}.

Technika podvzorkování výrazně zrychluje proces učení a dokonce vylepšuje přesnost natrénovaných vektorů neobvyklých slov.

4 Evaluace sémantických modelů

V této kapitole budou popsány způsoby vyhodnocení úspěšnosti sémantických modelů. Nejdříve uvedu základní vzorce a následně popíšu samotné metody.

4.1 Sémantická podobnost slov

Význam jednotlivých slov je vyjádřen vektorem ve vektorovém prostoru s vysokou dimenzí. Slova podobná nebo slova související se v tomto prostoru nachází blízko sebe. Výpočet vzdálenosti dvou vektorů lze určit pomocí *euklidovské vzdálenosti*

$$d(a,b) = \sqrt{\sum_{i=1}^{N}(a_i - b_i)^2}, \tag{4.1}$$

kde a a b jsou vektory s dimenzí N. Pro výpočet sémantické podobnosti slov se častěji používá *kosínová podobnost* definovaná vztahem (4.2).

$$cos(a,b) = \frac{a \cdot b}{||a|| \cdot ||b||} = \frac{\sum_{i=1}^{N} a_i \cdot b_i}{\sqrt{\sum_{i=1}^{N}(a_i)^2} \cdot \sqrt{\sum_{i=1}^{N}(b_i)^2}}, \tag{4.2}$$

kde a a b jsou vektory s dimenzí N. Podobnost slov je rovna kosínu úhlu, který jejich vektory svírají.

4.2 Korelace výsledků

Úspěšnost modelu se může měřit pomocí korelace s daty ohodnocenými lidmi. Obvykle se používají dvě korelace, *Pearsonova korelace* a *Spearmanova korelace*.

Výsledná korelace nabývá hodnot od -1 do 1. Rostoucí absolutní hodnota korelace vyjadřuje rostoucí lineární závislost veličin.

4.2.1 Pearsonova korelace

Pearsonova korelace [14] vyjadřuje lineární vazbu dvou veličin pomocí vzorce pro korelaci.

$$\rho = \frac{cov(X,Y)}{\sigma(X) \cdot \sigma(Y)}, \tag{4.3}$$

kde $cov(X,Y)$ je kovariance definována vztahem (4.4) a $\sigma(X)$ je směrodatná odchylky veličiny X definovaná vztahem (4.5).

$$cov(X,Y) = E(X \cdot Y) - E(X) \cdot E(Y), \tag{4.4}$$

$$\sigma(X) = \sqrt{E(X^2) - E^2(X)}, \tag{4.5}$$

kde $E(X)$ je střední hodnota diskrétní veličiny X definována vztahem (4.6).

$$E(X) = \sum_i x_i P(x_i) \tag{4.6}$$

Pearsonův korelační koeficient pro výpočet korelace mezi vektory lze vypočítat podle (4.7).

$$r = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=0}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=0}^{n}(Y_i - \bar{Y})^2}}, \tag{4.7}$$

kde $\bar{X}$ a $\bar{Y}$ jsou aritmetické průměry.

4.2.2 Spearmanova korelace

Spearmanova (pořadová) korelace udává statistickou závislost mezi pořadím jednotlivých složek vektorů. Vzestupně uspořádáme jednotlivé složky x_i a y_i obou vektorů podle velikosti a přiřadíme jim pořadová čísla p_i a q_i. Hodnota korelace se vypočítá podle (4.8).

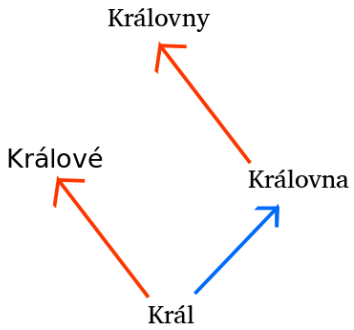
$$\rho = 1 - \frac{6\sum_{i=1}^{n}(p_i - q_i)}{n(n^2-1)}, \tag{4.8}$$

kde n je počet složek vektoru.

4.3 Analogie slov

Analogie slov (angl. *word analogy*) [8] je jedním z lingvistických jevů pomocí kterého se dá určit úspěšnost sémantického modelu. V úloze analogie slov nám jsou předloženy dva páry slov, které sdílejí vztah (např. „muž:žena“ a „král:královna“). Identita čtvrtého slova („královna“) není známa, úkolem modelu je toto slovo vyvodit z prvních tří slov. Model tedy odpovídá na otázku typu: „muž se má k ženě jako král k — ?“. Dále se tato úloha analogie slov bude označovat jako $a : A, b : B$.

Vektorová reprezentace slov zachovává lineární závislost sémantických a syntaktických vlastností těchto slov. Například odečtením vektoru slova „král“ od vektoru slova „králové“ získáme vektor, který představuje vektor přechodu od jednotného čísla k číslu množnému. Tato vlastnost je vidět na obr. 4.1.



Obrázek 4.1: Analogie slov

Vztah mezi slovy je tedy vyjádřen rozdílem jejich vektorů (4.9).

$$vec_{kralove} - vec_{kral} \approx vec_{kralovny} - vec_{kralovna} \tag{4.9}$$

Model řeší úlohu $a : A, b : B$, kde B neznáme, nalezením příslušných vektorů vec_a, vec_A, vec_b a následným výpočtem $y = vec_A - vec_a + vec_b$. y je vektorová reprezentace slova, které by mělo být správnou odpovědí. Vypočteným vektorem ale nemusí být žádné slovo reprezentováno, proto se najde vektor, který má největší kosínovou podobnost k vektoru y.

$$w = argmax(cos(vec_w, y), \tag{4.10}$$

kde w je slovo, které model vyvodil a je řešením úlohy analogie slov.

4.4 Podobnost a souvislost slov

Slova si mohou být podobná a mohou spolu také souviset. Pojmy se lehce zamění, přesto vyjadřují výrazně odlišný vztah mezi slovy. Jsou-li si dvě slova velice podobná, tak při jejich záměně se výrazně nezmění význam věty. Souvislost vyjadřuje pravděpodobnost, že se slova vyskytnou ve stejné větě, popř. kontextu. Slova *kolo* a *bicykl* si jsou podobná a slova *kolo* a *silnice* spolu souvisejí. Ve větě *„Dojíždí do práce na kole“* je možné zaměnit slovo *kolo* za podobné slovo *bicykl* bez změny významu věty. Záměna za souvislé slovo *silnice* ale není možná, protože věta by se stala nesmyslnou. Podobnost a souvislost slov lze využít pro vyhodnocení úspěšnosti sémantických modelů. Existuje mnoho datasetů, které obsahují velký počet dvojic podstatných jmen, přídavných jmen nebo sloves. Tyto dvojice jsou následně profesionálními anotátory ohodnoceny předem danou stupnicí podle toho jak spolu slova souvisejí nebo si jsou podobná.

5 Implementace a optimalizce skip-gramu

V kapitole bude popsána volně dostupná implementace metody *skip-gram* v jazyce `C`[1]. Zdrojový kód je vysoce optimalizovaný, ale jak bývá zvykem programů napsaných v jazyce `C`, tak je také velice nečitelný. Program využívá knihovnu `pthread.h`, proto není přenositelný a nelze ho zkompilovat na operačních systémech, které nesplňují standard `POSIX`[2].

5.1 Vytvoření slovníku

Metoda nejdříve potřebuje slovník slov, pro která je třeba natrénovat mnohorozmněrné vektory. Programu se může předat již vytvořený slovník jako textový soubor, ale obvyklé je vytvořit slovník z trénovacího korpusu. Pro každé unikátní slovo je připravena struktura, viz fragment kódu 5.1, kde je uchovaný samotný řetězec a počet jeho výskytů.

```
struct vocab_word {
   long long cn; // pocet vyskytu
   char *word;
}
struct vocab_word *vocab; // slovnik dyn. pole
int *vocab_hash; // hashovaci tabulka
```

Fragment kódu 5.1: Základní struktury vytváření slovníku

Jednotlivé struktury jsou uložené v dynamickém poli, který reprezentuje vytvářený slovník. K urychlení vyhledávání slova se používá hashovací tabulka, která umožňuje vyhledání slova s průměrnou složitostí *O(1)*. Hashovací tabulka je implementována prostým polem dat typu `int`, v tabulce se uchovávají pouze indexy do slovníku (dynamického pole). Velikost hashovací tabulky je typicky 30 milionů, přesto může dojít ke kolizi, která je řešena sekvenčním nalezením nejbližšího volného místa od pozice dané hashovací funkcí.

[1] https://github.com/dav/word2vec

[2] Jednotné rozhraní, které zajišťuje přenositelnost programů, především `unixové` operační systémy

Vytvořený slovník je celý načtený v paměti, proto je nutné, aby slovník nebyl prostorově příliš velký. Slovník se nejdříve seřadí podle četnosti výskytu a následně jsou vyřazena slova, která se vyskytly v celém trénovacím korpusu méně než předem daný počet (může být uživatelem zadán). Tato slova jsou nejčastěji překlepem, nesmyslným výrazem nebo slovem nepodstatný pro vytvářený model.

5.2 Učení modelu

Jak již bylo zmíněno, rychlost učení metody *word2vec* je jeden z důvodů, proč vytváří kvalitnější mnohorozměrné vektory než metody ostatní. Architektura *skip-gram* je schopná natrénovat model z 1,6 miliardy slov o dimenzi vektoru 300 za pouhé dva dny. Učení je urychleno jak technikami *negativního vzorkování* a *podvzorkování*, tak paralelizací procesu učení sítě.

5.2.1 Paralelizace učení

Proces učení lze paralelizovat pomocí vláken. Počet vláken může uživatel zadat dle své potřeby. Každému vláknu je přidělena část trénovacího korpusu a všechna vlákna přistupují ke společným globálním proměným. Přesto se synchronizace mezi vlákny neřeší, nejsou implementovány žádné synchronizační prvky. Chyba vzniklá neošetřenou synchronizací je malá a na výsledný vektor nemá tak výrazný vliv, aby bylo třeba zajistit správnou synchronizaci vláken. Aktualizace jsou velice časté, proto k opravení chyby obvykle dojde později během učení[3]. Vynechání synchronizace vláken vede k rychlejšímu procesu učení.

5.2.2 Negativní vzorkování

Technika *negativního vzorkování* používá modifikované unigramové rozdělení pro výběr negativních vzorků. Výpočet pravděpodobnosti slova dle (3.17) je kvůli knihovní funkci `pow` pro výpočet mocniny příliš pomalý a rychlost učení by nebyla tak výrazně urychlena. Modifikované unigramové rozdělení je tedy předem předpočítané do pole obsahující pouze indexy slov. Ve fragmentu kódu 5.2 je ukázáno vyplnění pole indexy slov. Výběr negativního vzorku se poté provede pomocí generátoru náhodných čísel. Vygeneruje se náhodné číslo od 0 do velikosti pole, pomocí kterého indexujeme do daného pole, na této pozici se nachází index našeho negativního vzorku.

[3]https://groups.google.com/forum/!#topic/word2vec-toolkit/iDTvPEOgFD0

```
double d1, power = 0.75, train_words_pow = 0;
for (a = 0; a < vocab_size; a++) train_words_pow +=
    pow(vocab[a].cn, power);
// i je index slova
i = 0;
// vypocet velikosti casti rozdeleni slova s indexem i
d1 = pow(vocab[i].cn, power) / train_words_pow;
for (a = 0; a < table_size; a++) {
  table[a] = i;
  // presahnuti casti rozdeleni pro dane slovo
  if (a / (double)table_size > d1) {
    i++;
    d1 += pow(vocab[i].cn, power) / train_words_pow;
  }
}
```

Fragment kódu 5.2: Vyplnění pole indexy slov

5.2.3 Podvzorkování

Implementace techniky *podvzorkování* nepoužívá k vypočtení pravděpodobnosti vynechání slova vzorec (3.19), ale jeho modifikaci:

$$P(w_i) = (\sqrt{\frac{f(w_i)}{t \cdot c}} + 1) \cdot \frac{t \cdot c}{f(w_i)}, \tag{5.1}$$

kde $f(w_i)$ je frekvence slova w_i, c je celkový počet unikátních slov ve slovníku a t je zvolený práh uživatelem.

Následně se vygeneruje náhodné číslo od 0 do 1, je-li pravděpodobnost slova w_i menší než vygenerované čisło, tak je slovo w_i vynechané. Implementace *podzvorkování* je ukázána ve fragmentu kódu 5.3.

```
real ran = (sqrt(vocab[word].cn / (sample * train_words)) + 1) *
    (sample * train_words) / vocab[word].cn;
next_random = next_random * (unsigned long long)25214903917 + 11;
if (ran < (next_random & 0xFFFF) / (real)65536) continue;
```

Fragment kódu 5.3: Implementace podvzorkování

5.2.4 Aktivační funkce

V této implementaci je aktivační funkcí funkce velice podobná sigmoidě. Podobně jako u *negativního vzorkování* jsou hodnoty funkce předpočítány, aby nedošlo k zbytečnému zpomalení počítáním aktivační funkce. Je předpočítáno 1000 hodnot od -6 do 6 a uloženy do pole `expTable`.

```
for (i = 0; i < EXP_TABLE_SIZE; i++) {
  expTable[i] = exp((i / (real)EXP_TABLE_SIZE * 2 - 1) * MAX_EXP);
  expTable[i] = expTable[i] / (expTable[i] + 1);
}
```

Fragment kódu 5.4: Předpočítání hodnot aktivační funkce

6 Navržený model

Cílem této práce je navrhnout rozšíření pro již existující sémantický model. Byla vybrána metoda *skipgram* z knihovny *word2vec*, která se v praxi využívá a dosahuje velice dobrých výsledků. Jsou volně dostupné dvě implementace metody, jedna je v jazyce `C` a druhá v jazyce `Python`. Rozhodl jsem se pro implementaci v jazyce `C`, jelikož se jedná o původní implementaci metody a kvůli mým zkušenostem s tímto jazykem.

V základní verzi metody *word2vec* nese sémantický význam celé slovo. Nabízí se proto otázka, jestli sémantický význam není zakódován i v menší jednotce než samotné slovo. Nejmenší možnou jednotkou je písmeno, ale to není praktické, protože se nachází ve velkém množství významově různých slovech. Další možnou jednotkou je slabika, která bude popsána v další kapitole.

6.1 Slabika

Slabika je nejjednodušší a nejmenší možnou organizační jednotkou řeči. Je to část slova vyslovená jedním nárazem dechového proudu. Slova tedy nevyslovují po hláskách, ale po slabikách. V promluvě přejímá fonetickou funkci, tvoří rytmus řeči.

Základem slabiky je *jádro* (nucleus), které je nejčastěji tvořené samohláskou. Před a za ní se mohou nacházet tzv. *svahy* (pretura, koda), které jsou tvořeny jednou nebo více souhláskami. Schéma slabiky vypadá následovně:

[pretura] – jádro – [koda]

V jazycích, které mají bohatou morfologii, má dělení slov na předpony, kmen, přípony, apod. vliv na slabičný charakter slova. Například změnou předpony u slovesa „*jet*“ se výrazně liší jeho význam „*zajet*“, „*přejet*“. Poměrně s malým množstvím slabik lze vytvořit velké množství gramaticky a sémanticky různých slov. Díky tomu je slabika vhodnou jednotkou k vytvoření sémantického modelu. Jednotlivé slabiky nesou sémantickou informaci.

6.2 Algoritmus slabikování

Slabikování je proces, který rozdělí slovo na jednotlivé slabiky [3]. Má několik pravidel, ale velké množství výjimek. Například slovo „*sestra*“ lze vyslabikovat třemi různými způsoby a to *se-stra*, *ses-tra* nebo *sest-ra*. Všechny tři varianty jsou z teoretického hlediska správné[1]. Vytvořit naprosto správný algoritmus slabikování proto není prakticky možné.

Přístupy k slabikování jsou různé, mohou být založené na statistice, na pravidlech nebo na nalezených vzorech v slabikách. Níže popsaný algoritmus patří do skupiny založených na pravidlech.

Vylepšený Lánského algoritmus [3, 6] je jednoduchý algoritmus slabikování. Používá jednoduchou definici slabiky. Slabika je sekvence hlásek a obsahuje právě jednu maximální sekvenci samohlásek. Algoritmus je založený na nalezení této maximální sekvenci samohlásek, které se říká samohlásková skupina, a následném přidávání souhlásek k těmto nalezeným skupinám. Algoritmus má pět pravidel:

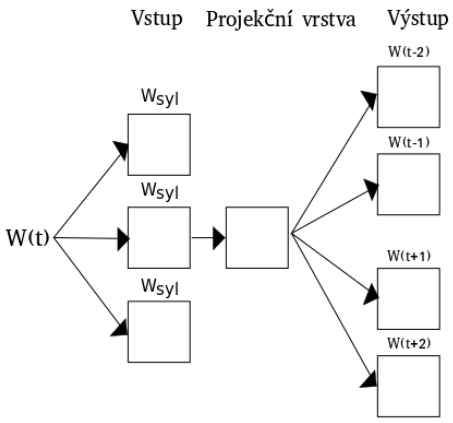
1. Vše po poslední samohlásce (samohláskové skupině) patří k poslední slabice.
2. Vše před první samohláskou (samohláskovou skupinou) patří k první slabice.
3. Je-li počet souhlásek mezi samohláskami sudý ($2n$), jsou rozděleny na dvě stejné poloviny (n/n), první polovina patří k levé samohlásce (samohláskové skupině) a druhá k pravé samohlásce (samohláskové skupině).
4. Je-li počet souhlásek mezi samohláskami lichý ($2n+1$), jsou rozděleny na dvě poloviny ($n/n+1$), první polovina patří k levé samohlásce (samohláskové skupině) a druhá k pravé samohlásce (samohláskové skupině).
5. Je-li mezi samohláskami pouze jedna souhláska, patří k levé samohlásce (samohláskové skupině).

Tato metoda je velice rychlá a dosahuje vysoké úspěšnosti. Díky její rychlosti je vhodným algoritmem, protože výrazně nesnižuje rychlost učení sítě.

[1] https://is.muni.cz/elportal/estud/ff/js07/fonetika/materialy/ch08s06.html

6.3 Popis rozšíření

Rozšíření modelu *word2vec* je založené na architektuře *skip-gram*. Vstupní slovo je rozděleno do slabik, na výstupu je (jak ve *skip-gramu*) předpovězený kontext na základě vstupního slova. Hlavní rozdíl je ve vstupních vektorech, které reprezentují slabiky. Neuronová síť rozšíření je naznačena na obr. 6.1.



Obrázek 6.1: Architektura sítě rozšířené metody *skip-gram*

Model musí vytvořit jak slovník slov k natrénování, tak slovník slabik. K vytvoření slovníku slabik nedochází během vytváření slovníku slov, ale až po dokončení tvorby slovníku slov proto, aby se do slovníku slabik nepřidaly slabiky ze slov, která se ve výsledném modelu nenacházejí. Vytvoření slovníku slov a slovníku slabik je ukázano ve fragmentu kódu 6.1.

```
while (1) { // vytvoreni slovniku slov
  ReadWord(word, fin);
  if (feof(fin)) break;
  a = AddWordToVocab(word, &vocab, vocab_hash);
  fscanf(fin, "%lld%c", &vocab[a].cn, &c);
  i++;
}
SortVocab(vocab, vocab_hash); // serazeni a redukce sl.
for (a = 0; a < vocab_size; a++) {
 // vytvoreni slovniku slabik ze slovniku slov
    SyllabicateAndAdd(vocab[a].word, vocab[a].cn);
}
SortVocab(syllab_vocab, syllab_vocab_hash);// serazeni sl. slab.
```

Fragment kódu 6.1: Vytvoření slovníků

Algoritmus slabikování obvykle používá regulární výraz k nalezení samohláskové skupiny, ten se ale moc neosvědčil a proces učení byl výrazně pomalejší. K vyhledání byla proto implementována vyhledávací tabulka ukázaná ve fragmentu kódu 6.2. Pomocí tabulky došlo k minimálnímu zpomalení učení. Algoritmus slabikování je implementován pouze pro anglický jazyk.

```
char vowels[256];

vowels['a'] = 1;
vowels['e'] = 1;
vowels['i'] = 1;
vowels['o'] = 1;
vowels['u'] = 1;
```

Fragment kódu 6.2: Vyhledávací tabulka samohlásek

Projekční vrstva h je vypočítána takto:

$$h = (v_{syl_1} + v_{syl_2} + ... + v_{syl_n}), \tag{6.1}$$

kde v_{syl} jsou vstupní vektory slabik získaných ze vstupního slova a n je počet slabik ve vstupním slově. Výpočet projekční vrstvy je ukázán ve fragmentu kódu 6.3.

```
for (c = 0; c < layer1_size; c++) syl_syn0[c] = 0;
for (c = 0; c < layer1_size; c++)/* adding syllable vectors */
  for (b = 0; b < a; b++) {
    l1 = layer1_size * syl->indexes[b];
    syl_syn0[c] += syn0[l1 + c];
}
```

Fragment kódu 6.3: Výpočet projekční vrstvy h

Následný výpočet pravděpodobnosti předpovězení kontextu o šířce C se vypočítá dle vztahu (3.10), který byl popsán v kapitole 3.6.2. Toto rozšíření využívá technik *negativního vzorkování* a *podvzorkování*. Narozdíl od původního *skip-gramu*, kde se aktualizuje pouze jeden vektor vstupního slova, se musí aktualizovat všechny vektory vstupních slabik. Aktualizace vstupních vektorů je ukázána ve fragmentu kódu 6.4.

```
int k;
for (k = 0; k < layer1_size; k++) {
  for (c = 0; c < syl.count; c++) {
    syn0[syl.indexes[c] * layer1_size + k] += neu1e[k];
  }
}
```

Fragment kódu 6.4: Aktualizace vstupních vektorů slabik

V původním modelu *word2vec* se natrénované vektory získají ze vstupních vektorů slov. V rozšíření reprezentují vstupní vektory sémantický význam slabik, proto se natrénované vektory reprezentující sémantický význam slov získávají z výstupních vektorů.

7 Experimenty

Kapitola se věnuje popisu jednotlivých experimentů prováděných na rozšířené metodě *skip-gram*. Dále se v kapitole nacházejí výsledky těchto experimentů a jejich shrnutí.

7.1 Trénovací korpus

Během všech experimentů byla síť učena na korpusu v anglickém jazyce. Jedná se o volně dostupný korpus[1] obsahující téměř miliardu slov. Korpus je již předzpracovaný, jsou předem např. odstraněna málo obvyklé znaky (např. @, #) a odstraněny mnohonásobné mezery. V korpusu se nacházejí pouze alfanumerické znaky a interpunkční znaménka.

7.2 Testovací data

Úspěšnost vytvořených vektorů slov byla testována pomocí testovacích dat založených na podobnosti dvojic slov a analogii slov.

7.2.1 Word Similiarity 353

Jedná se datovou sadu obsahující 353 dvojic slov, které ohodnotili anotátoři hodnotami na stupnici od 0 do 10 dle míry souvislosti a podobnosti. V datasetu se nacházejí podstatná jména, přídavná jména a slova. Datovou sadu je možné rozdělit do dvou polovin, jedna vyjadřující míru souvislosti, druhá míra podobnosti.

7.2.2 Google Word Analogy

Společnost *Google* vydala datovou sadu[2] obsahující přes 20 000 příkladů *analogie slov* pomocí kterých se dá otestovat sémantická a syntaktická úspěšnost modelu. Sémantická úspěšnost se testuje na tématech typu hlavní města světa, rodina a měna. Syntaktická úspěšnost je testována na gramatických tvarech sloves a přídavných jmen.

[1] http://www.statmt.org/lm-benchmark/1-billion-word-language-modeling-benchmark-r13output.tar.gz

[2] https://github.com/dav/word2vec/blob/master/data/questions-words.txt

7.2.3 Rubenstein & Goodenough

Jeden ze základních evaluačních datasetů [17]. Obsahuje 65 párů podstatných jmen ohodnocených dle podobnosti stupnicí od 0 do 4.

7.2.4 Rare Words

Dataset *Rare Words* [18] byl vytvořený na Stanfordovo univerzitě v Kalifornii. Obsahuje 2034 párů slov, která jsou vzácná resp. málo obvyklá.

7.3 Knihovna gensim

Gensim je *open source* knihovna pro jednoduchou práci s různými algoritmy sémantického modelování [13]. Je implementovaná ve skriptovacím jazyce *Python*. Knihovna klade důraz na jednoduchost, robustnost a efektivitu. Přestože jsou algorimy napsány v jazyce *Python* jsou velice rychlé díky knihovně *NumPy*, která zefektivňuje práci s poli a maticemi. V knihovně *gensim* je implementován *word2vec*, *LSA*, a mnoho dalších algoritmů vytvářejících vektory reprezentující význam slova. Během experimentů byla knihovna použita pro výpočet *Pearsonovo* a *Spearmanovo* korelace modelu s datasetem *WordSim353*, *Rare Words* a *Rubenstein & Goodenough*.

7.4 Parametry sítě

Metoda *skip-gram* nabízí mnoho tzv. hyperparametrů, které uživatel může libovolně upravovat. Konfigurace těchto parametrů mění jak rychlost, tak kvalitu učení. Hodnoty následujících parametrů jsou výchozí a byly použity během experimentů není-li uvedeno jinak:

- dimenze vektoru – 200,
- rychlost učení (*angl. learning rate*) – 0,025,
- velikost kontextového okna – 8,
- práh podvzorkování – 10^{-4},
- počet negativních vzorků – 25,
- počet iterací – 1.

7.5 Výsledky experimentů

Všechny experimenty probíhaly na stejném počítači s operačním systémem *Mac OS*. Experimenty byly zaměřeny především za účelem zjištění optimálních hyperparametrů pro rozšířenou metodu *skip-gram*. Klíčovým hyperparametrem, který výrazně ovlivňuje kvalitu výsledných vektorů, je rychlost učení. Autoři metody *skip-gram* uvádějí pro původní metodu optimální rychlost učení hodnotu $0,025$. Ta se může ale pro vytvořené rozšíření metody velice lišit.

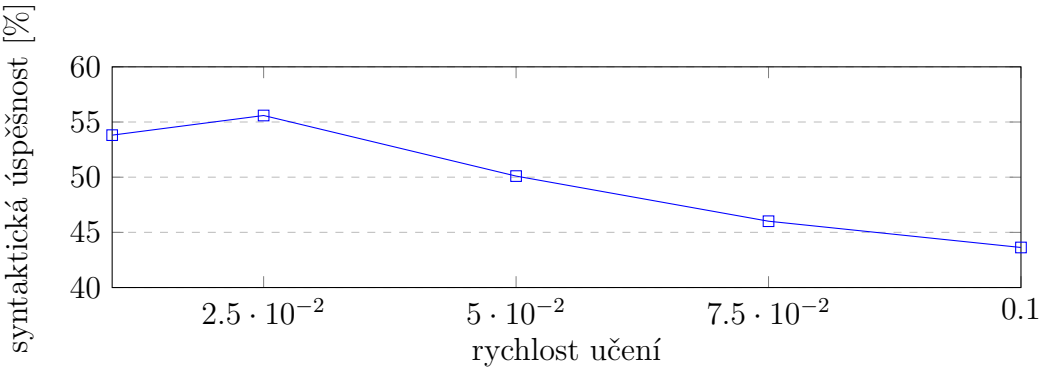
V tabulkách 7.1 a 7.2 jsou uvedeny výsledky úspěšnosti rozšířeného modelu při měnící se rychlosti učení. Vytvořený model byl otestován na datasetech *WordSim353*, *Google Word Analogy*, *Rubenstein & Goodenough* a *Rare Words*. Použity byly všechny výchozí hodnoty, měnila se pouze zmíněná rychlost učení. Velikost trénovacího korpusu byla 300 milionů slov. Z grafu (viz obr. 7.1) vidíme klesající syntaktickou úspěšnost při rostoucí rychlosti učení nad hodnotu $0,025$.

rychlost učení	dataset podobnosti		dataset souvislosti	
	PC	SC	PC	SC
0,01	0,589	0,553	0,450	0,463
0,025	**0,637**	**0,612**	**0,491**	**0,499**
0,05	0,608	0,589	0,474	0,482
0,075	0,572	0,548	0,458	0,471
0,1	0.572	0,546	0,469	0,486

Tabulka 7.1: Dataset WordSim353. Úspěšnost modelu při měnící se rychlosti učení. Ostatní hyperparametry jsou výchozí.

rych. učení	Rare Words		RG65		Google Word An.	
	PC	SC	PC	SC	SEM[%]	SYN[%]
0,01	0,323	**0,361**	0,525	0,488	48,82	53,81
0,025	0,319	0,329	**0,562**	**0,546**	**56,94**	**55,59**
0,05	0,334	0,329	0,542	0,519	56,79	50,10
0,075	0,344	0,339	0,552	0,534	55,20	46,01
0,1	**0,347**	0,342	0,523	0,498	52,96	43,63

Tabulka 7.2: Datasety Rare Words, Rubenstein & Goodenough a Google Word Analogy. Úspěšnost modelu při měnící se rychlosti učení. Ostatní hyperparametry jsou výchozí.



Obrázek 7.1: Klesající syntaktická úspěšnost při zvyšující se rychlosti učení

V tabulce 7.3 a 7.4 jsou opět ukázány výsledky modelů při měnící se rychlosti učení s dimenzí vektoru 300. Z těchto tabulek jsou velice zajímavé výsledky modelu při rychlosti učení $0,1$, model s touto hodnotou divergoval a nebyl schopný natrénovat vektory. Rychlost učení $0,025$ dosahuje nejlepších výsledků, proto ji budeme považovat za optimální.

rychlost učení	dataset podobnosti		dataset souvislosti	
	PC	SC	PC	SC
0,01	0,597	0,561	0,453	0,466
0,025	**0,640**	**0,612**	0,482	0,496
0,05	0,624	0,607	**0,493**	**0,504**
0,075	0,597	0,578	0,489	0,499
0,1	0,355	0,352	0,171	0,169

Tabulka 7.3: Dataset WordSim353. Úspěšnost modelu při měnící se rychlosti učení. Dimenze vektorů 300. Ostatní hyperparametry jsou výchozí.

rych. učení	Rare Words		RG65		Google Word An.	
	PC	SC	PC	SC	SEM[%]	SYN[%]
0,01	0,303	0,346	0,491	0,456	44,73	55,05
0,025	0,308	0,322	**0,577**	**0,584**	**58,17**	**57,43**
0,05	0,317	0,316	0,551	0,531	57,83	52,02
0,075	**0,336**	**0,330**	0,583	0,557	55,95	46,27
0,1	0,113	0,107	0,172	0,212	0,24	0,08

Tabulka 7.4: Úspěšnost modelu při měnící se rychlosti učení. Dimenze vektorů 300. Ostatní hyperparametry jsou výchozí.

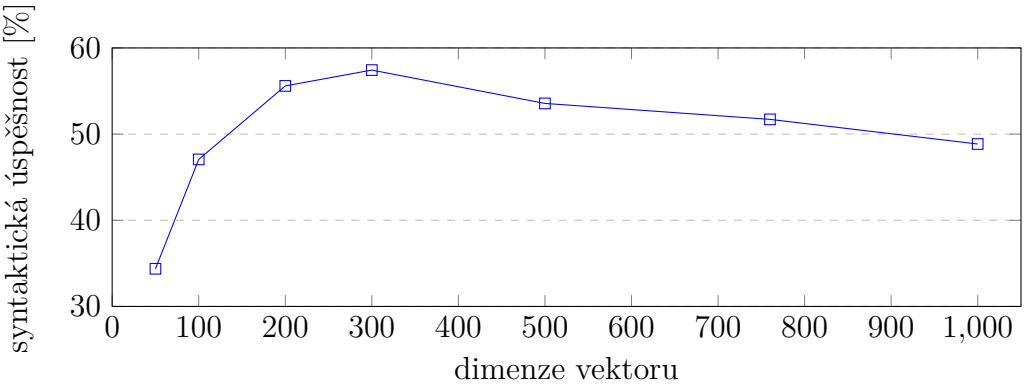
V tabulce 7.5 a 7.6 jsou výsledky při rostoucí dimenzi. Rychlost učení byla zvolena hodnota $0,025$, kterou jsme určili jako optimální. Výsledky jsou velice zajímavé, z obou tabulek vidíme, že žádná dimenze vektoru není nejúspěšnější ve většině datasetů jako tomu bylo u rychlosti učení. Dimenze vektoru 50 dosáhla nejlepších výsledků v datasetu se vzácnými slovy, dimenze 1000 zase měla nejlepší úspěšnost v datasetu $WordSim353$, přestože se stejnou dimenzí dosáhla výrazně nižší úspěšnosti v datasetu s analogiemi slov, kde nejlepších výsledků dosáhla dimenze 300. Na grafu (viz obr. 7.2) je zobrazena úspěšnost modelu v datasetu $WordSim353$ podobnosti při měnící se dimenzi vektoru, na grafu (viz obr. 7.3) je zobrazena syntaktická úspěšnost při měnící se dimenzi vektoru.

rychlost učení	dataset podobnosti		dataset souvislosti	
	PC	SC	PC	SC
50	0,549	0,546	0,425	0,442
100	0,589	0,568	0,469	0,479
200	0,637	0,612	0,491	0,499
300	0,634	0,612	0,482	0,496
500	0,630	0,607	0,484	0,502
750	0,643	0,624	0,492	0,508
1000	**0,644**	**0,635**	**0,493**	**0,510**

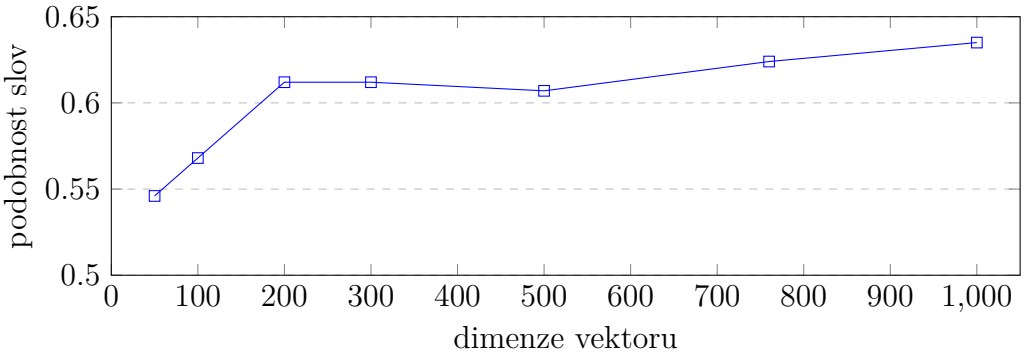
Tabulka 7.5: Dataset Wordsim353. Úspěšnost modelu při měnící se dimenzi vektorů. Rychlost učení $0,025$. Ostatní hyperparametry jsou výchozí.

rych. učení	Rare Words		RG65		Google Word An.	
	PC	SC	PC	SC	SEM[%]	SYN[%]
50	**0,341**	**0,346**	0,462	0,441	37,79	34,37
100	0,336	0,342	0,532	0,494	51,22	47,06
200	0,319	0,329	0,562	0,546	56,94	55,59
300	0,308	0,322	**0,577**	**0,584**	**58,17**	**57,43**
500	0,281	0,297	0,561	0,544	54,81	53,66
760	0,264	0,282	0,545	0,510	51,60	51,71
1000	0,252	0,269	0,556	0,523	50,14	48,85

Tabulka 7.6: Datasety Rare Words, Rubenstein & Goodenough a Google Word Analogy. Úspěšnost modelu při měnící se dimenzi vektorů. Rychlost učení $0,025$. Ostatní hyperparametry jsou výchozí.



Obrázek 7.2: Syntaktická úspěšnost modelu při měnící se dimenzi vektoru.



Obrázek 7.3: Úspěšnost modelu na datasetu WordSim353 při měnící se dimenzi vektoru. Hodnoty jsou Spearmanova korelace.

Nakonec byly porovnány původní metoda *skip-gram* a vytvořené rozšíření této metody. Byly vybrány hodnoty $0,025$ jako rychlost učení a 300 jako dimenze výsledných vektorů. Pro zajímavost byly vytvořeny vektory ze vstupní vrstvy, kde se nacházejí vektory slabik. Každé slovo ze slovníku se rozslabikovalo, následně se pro každou slabiku nalezl příslušný vektor ve vstupní vrstvě a tyto vektory se nakonec sečetly. Výsledný vektor byl použit jako reprezentace daného slova. Výsledky jsou v tabulkách 7.7 a 7.8. Původní *skip-gram* dosahuje téměř ve všech datasetech lepších výsledků, jedinou výjimkou je dataset *Rubenstein & Goodenough*. Vektory vytvořené pomocí vstupní vrstvy nedosáhly nijak dobrých výsledků.

model	dataset podobnosti		dataset souvislosti	
	PC	SC	PC	SC
skip-gram	**0,649**	**0,618**	**0.505**	**0.515**
slabiky	0,634	0,612	0,482	0,496
vstup. vrstva	0,464	0,450	0,410	0,396

Tabulka 7.7: Porovnání *skip-gramu*, jeho rozšíření (slabiky) a vektorů vytvořených ze vstupní vrstvy na datasetu WordSim353.

model	Rare Words		RG65		Google Word An.	
	PC	SC	PC	SC	SEM[%]	SYN[%]
skip-gram	**0,344**	**0,355**	**0,591**	0,551	**61,50**	**62,97**
slabiky	0,308	0,322	0,577	**0,584**	58,17	57,43
vstup. vrstva	0,282	0,262	0,229	0,226	19,85	48,53

Tabulka 7.8: Porovnání *skip-gramu* a jeho rozšíření (slabiky) a vektorů vytvořených ze vstupní vrstvy na datasetu Rare Words, Rubenstein & Goodenough a Google Word Analogy.

8 Závěr

V rámci této práce jsem se seznámil s neuronovými sítěmi a s několika sémantickými modely, některé z nich jsem popsal v teoretické části. Podrobně jsem nastudoval model *word2vec* a především její metodu *skip-gram*, pro kterou jsem navrhl rozšíření pomocí slabik. Rozšíření bylo otestováno na standardních testovacích datech v anglickém jazyce.

Rozšíření metody *skip-gram* má odlišnou architekturu sítě, na vstupu sítě se nenachází pouze jedno slovo, ale slabiky získané ze vstupního slova. Vytvořené rozšíření bylo otestováno na základních evaluačních datasetech. Během experimentů byl model otestován s různou konfigurací hyperparametrů. Nejdříve byl model otestován při měnící se rychlosti učení, optimální hodnota rychlosti učení byla $0,025$. Následně se testovala úspěšnost modelu s různými dimenzemi výsledných vektorů. Žádná hodnota dimenze vektoru se neosvědčila jako optimální, protože model měl největší úspěch v různých datasetech s různými dimenzemi vektoru. Model byl poté porovnán s původní metodou *skip-gram*. Vytvořený model zaostával v úloze analogie slov za původní metodou o $3,33\%$ v sémantické úspěšnosti a $5,54\%$ v syntaktické, v datasetech podobnosti a souvislosti již nebyl o tolik horší, v datasetu *Rubenstein & Goodenough* dokonce dosáhl lepších výsledků. Vytvořené rozšíření bohužel nepřekonalo původní metodu, ale dokázali jsme fakt, že i v slabikách je uložen sémantický význam, kterého lze využít.

Budoucí rozšíření této práce by mohlo spočívat ve změně slabikovacího algoritmu *vylepšený Lánského algoritmus* za efektivnější. Dalším možným rozšířením je experimentace se slabikami. Ty by se mohly na vstupu různě kombinovat, např. spojení dvou po sobě jdoucích slabik.

Seznam zkratek

HAL ..... Hyperspace Analogue to Language

MSE ..... Mean Squared Error

CE ..... Cross-Entropy

SVD ..... Singular value decomposition

LSA ..... Latent Semantic Analysis

POS ..... Part-of-speech

NLP ..... Natural language processing

CBOW ..... Continuous Bag-of-Words

Literatura

[1] Bengio, Y. et al. A Neural Probabilistic Language Model. *Journal of Machine Learning Research*. 2003, 3, s. 1137–1155. Dostupné z: http://dblp.uni-trier.de/db/journals/jmlr/jmlr3.html#BengioDVJ03.

[2] Collobert, R. – Weston, J. A unified architecture for natural language processing: Deep neural networks with multitask learning, 2008.

[3] Hejtmánek, J. Using Syllables as Acoustic Units for Spontaneous Speech Recognition. In Sojka, P. et al. (Ed.) *Text, Speech and Dialogue*, s. 299–305, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-15760-8.

[4] Konkol, M. – Brychcín, T. – Konopík, M. Latent semantics in Named Entity Recognition. *Expert Systems with Applications*. 2015, 42, 7, s. 3470 – 3479. ISSN 0957-4174. doi: http://dx.doi.org/10.1016/j.eswa.2014.12.015. Dostupné z: http://www.sciencedirect.com/science/article/pii/S0957417414007933.

[5] Landauer, F. P. W. . L. D. T. K. *Introduction to Latent Semantic Analysis* [online]. Colorado Edu. [cit. 2018/03/26]. Dostupné z: http://lsa.colorado.edu/papers/dp1.LSAintro.pdf.

[6] Lansky, J. – Zemlicka, M. Text Compression: Syllables. 2005.

[7] LeCun, Y. – Bengio, Y. – Hinton, G. E. Deep learning. *Nature*. 2015, 521, 7553, s. 436–444. doi: 10.1038/nature14539. Dostupné z: https://doi.org/10.1038/nature14539.

[8] Levy, O. – Goldberg, Y. Linguistic Regularities in Sparse and Explicit Word Representations. In Morante, R. – Yih, W. (Ed.) *CoNLL*, s. 171–180. ACL, 2014. Dostupné z: http://dblp.uni-trier.de/db/conf/conll/conll2014.html#LevyG14. ISBN 978-1-941643-02-0.

[9] Lund, K. – Burgess, C. Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instruments, & Computers*. Jun 1996, 28, 2, s. 203–208. ISSN 1532-5970. doi: 10.3758/BF03204766. Dostupné z: https://doi.org/10.3758/BF03204766.

[10] Mikolov, T. et al. Efficient Estimation of Word Representations in Vector Space. *CoRR*. 2013, abs/1301.3781. Dostupné z: http://dblp.uni-trier.de/db/journals/corr/corr1301.html#abs-1301-3781.

[11] Mikolov, T. et al. Distributed Representations of Words and Phrases and their Compositionality. In Burges, C. J. C. et al. (Ed.) *Advances in Neural Information Processing Systems 26*. Curran Associates, Inc., 2013. s. 3111–3119. Dostupné z: https://arxiv.org/pdf/1310.4546.pdf.

[12] Pennington, J. – Socher, R. – Manning, C. D. Glove: Global vectors for word representation. In *In EMNLP*, 2014.

[13] Řehůřek, R. – Sojka, P. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, s. 45–50, Valletta, Malta, May 2010. ELRA. http://is.muni.cz/publication/884893/en.

[14] Reif, J. *Metody matematické statistiky*. Západočeská univerzita, 2004. ISBN 80-7043-302-7.

[15] Rojas, R. *Neural Networks, A Systematic Introduction* [online]. Springer. [cit. 2018/03/15]. Dostupné z: https://page.mi.fu-berlin.de/rojas/neural/neuron.pdf.

[16] Rong, X. word2vec Parameter Learning Explained. *CoRR*. 2014, abs/1411.2738. Dostupné z: http://arxiv.org/abs/1411.2738.

[17] Rubenstein, H. – Goodenough, J. B. Contextual correlates of synonymy. *Commun. ACM*. 1965, 8, 10, s. 627–633. Dostupné z: http://dblp.uni-trier.de/db/journals/cacm/cacm8.html#RubensteinG65.

[18] Luong, M. – Socher, R. – Manning, C. D. Better word representations with recursive neural networks for morphology. In *In Proceedings of the Thirteenth Annual Conference on Natural Language Learning. Tomas Mikolov, Wen-tau*, 2013.

[19] Tučková, J. *Algoritmy a struktury neuropočítačů* [online]. České vysoké učení technické, Fakulta elektrotechnická. [cit. 2018/03/16]. Učební texty. Dostupné z: http://amber.feld.cvut.cz/ssc/ssc-cv/bpg.pdf.

[20] Volná, E. *Neuronové sítě 1* [online]. Ostravská univerzita, Přírodovědná fakulta, 2002. [cit. 2018/03/15]. Učební texty. Dostupné z: http://www1.osu.cz/~volna/Neuronove_site_skripta.pdf.

A Uživatelská příručka

K jednoduchému přeložení a sestavení programu jsou třeba nástroje *Make* a *GCC*. Ty jsou součástí *GNU*, který je obvykle již v unixových operačních systémech.

Přeložení a spuštění

Program lze jednoduše přeložit pomocí nástroje *Make*. V adresáři se zdrojovými soubory stačí zadat příkaz:

```
make
```

Spustitelný program se vytvoří v adresáři `bin`. Program se spouští příkazem:

```
./word2vec <argumenty>
```

Program má velké množství možných argumentů, u nepovinných argumentů je v závorce uvedena výchozí hodnota:

- `-size <velikost>` – dimenze natrénovaných vektorů `(100)`,
- `-train <soubor>` – cesta k trénovacímu korpusu,
- `-save-vocab <soubor>` – uložení slovníku slov,
- `-save-syl-vector <soubor>` – uložení vektorů slabik,
- `-save-syl-vocab <soubor>` – uložení slovníku slabik,
- `-read-vocab <soubor>` – použití předem vytvořeného slovníku,
- `-debug <int>` – úroveň debugovacího výpisu `(2)`,
- `-binary <int>` – binární výstup natrénovaných vektorů `(0)`,
- `-alpha <float>` – rychlost učení `(0,025)`,
- `-output <soubor>` – soubor s natrénovanými vektory,
- `-window <int>` – velikost okna `(5)`,

- `-sample <float>` – práh podvzorkování 10^{-4},
- `-negative <int>` – počet negativních vzorků `(5)`,
- `-threads <int>` – počet vláken `(12)`,
- `-min-count <int>` – práh minimálního výskytu slova `(5)`.

B Obsah přiloženého DVD

Přiložené DVD má následující adresářovou strukturu:

- **src** - Obsahuje zdrojové kódy a soubor *Makefile* potřebný pro nástroj *make*.
- **bin** - Cílový adresář nástroje *make*, po přeložení jednotlivých programů obsahuje binární soubory.
- **docs** - Obsahuje zdrojové kódy textu a tuto práci ve formátu PDF.
- **scripts** - Skripty pro jednoduché otestování vytvořené práce.
- **datasets** - Použité testovací datasety.
- **data** - Obsahuje jeden trénovací korpus k otestování vytvořeného rozšíření.
- **readme** - Textový soubor obsahující krátký popis DVD.