Automatická korekcia textu pomocou neurónových sietí

DIPLOMOVÁ PRÁCA

Pavol Kaiser

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY
KATEDRA INFORMATIKY

Študijný odbor: Informatika

Vedúci záverečnej práce: Ing. Igor Farkaš

BRATISLAVA 2007

Čestne vyhlasujem, že som túto diplomovú prácu
vypracoval samostatne s použitím citovaných zdrojov.

Abstrakt

Pavol Kaiser, Automatická korekcia textu pomocou neurónových sietí,
Diplomová práca, Univerzita Komenského v Bratislave,
Fakulta matematiky fyziky a informatiky, Katedra Informatiky, 2007
Vedúci: Ing. Igor Farkaš

Práca sa snaží ukázať možnosti neurónových sietí pri oprave textu. Konkrétnejšie sa venuje sieti Merge Self-Organizing Map (MSOM). Sieť MSOM je schopná zachytiť kontext a vďaka tomu je vhodná na korekciu textu, kde je znalosť kontextu veľmi podstatná. V práci je ukázané, ako je možné túto sieť natrénovať na korpuse jazyka a následne takto natrénovanú sieť použiť pri oprave textu. Definovaná je funkcia energie reťazca, pričom táto funkcia sa počíta na základe výstupov siete. Opravený reťazec je možné nájsť ako mutáciu pôvodného reťazca s najvyššou energiou. Ukazuje tiež, ako sa dajú riešiť problémy s časovou náročnosťou a sčasti i nepresnosťou tejto siete.

Kľúčové slová: neurónová sieť, MSOM, korekcia textu.

Predhovor

História umelých neurónových sietí siaha do roku 1943, kedy McCulloch a Pitts predstavili prvý model neurónovej siete [7]. Napriek tomu skutočný rozmach umelých neurónových sietí nastal až v druhej polovici 80-tych rokov minulého storočia [8]. Potenciál umelých neurónových sietí ešte nie je dostatočne preskúmaný a preto by som aj ja svojou prácou rád pomohol ukázať ich možnosti.

Vo svojej práci sa pokúsim ukázať, ako sa dajú neurónové siete využiť na natrénovanie jazyka a zákonitostí ktoré v ňom platia. Aby som demonštroval moje výsledky, navrhnem spôsob ako pomocou takto natrénovanej siete možno opravovať chyby v texte v danom jazyku.

Problém porozumenia počítačového programu nejakému textu sa stále nedarí dostatočne riešiť. Zdarné vyriešenie tohto problému sľubuje úžasné možnosti do budúcnosti (stroje, ktoré by dokázali komunikovať s človekom a reagovať na ľudskú reč).

S týmto problémom priamo súvisí problém rozpoznávania ľudskej reči (speech recognition), ako aj optického rozpoznávania symbolov – OCR (optical character recognition). V súčasnosti poznáme niektoré metódy, ktoré riešia uvedené dva problémy, ale faktom zostáva, že tieto metódy nie sú neomylné. Pri aplikácii týchto metód často dochádza k chybám v rozpoznaných textoch/reči. Pritom ak je text/reč chybne rozpoznaná, aj porozumenie danému textu/reči je oveľa ťažšie.

Ďalšou možnou aplikáciou opravy textu sú textové editory, pri ktorých využívaní sa ľudia často nevyhnú rôznym chybám a preklepom.

Poznáme niekoľko spôsobov ako odhaľovať prípadné chyby v texte. Tou najjednoduchšou je porovnávanie jednotlivých slov so slovami v slovníku. Pomocou takejto metódy je možné rozpoznať niektoré chyby, ale zďaleka nie všetky. Príčinou je závislosť daného slova na kontexte. V anglickom i slovenskom jazyku môžeme nájsť mnoho slov, z ktorých aj pri zamenení jedného symbolu vznikne slovo, ktoré daný jazyk obsahuje. Ak je však takto poškodené slovo vo vete, spravidla sa nové slovo do kontextu vety nehodí.

Zachytenie kontextu je nevyhnutným predpokladom na správne opravenie textu. Napriek tomu, že sú neurónové siete pomerne časovo náročné, umožňujú zachytiť kontext vo vete. Vďaka tejto skutočnosti majú neurónové siete potenciál dosahovať lepšie výsledky ako bežné spôsoby opravy textu.

Obsah

1 Úvod

Problém korekcie textu sa dá riešiť viacerými spôsobmi, z ktorých ani jeden nedosahuje 100% úspešnosť. Prvou možnosťou je použiť takzvané databázové riešenie, pri ktorom si vytvoríme databázu (slovník) slov patriacich do daného jazyka a následne porovnávame jednotlivé slová v texte s touto databázou. Pre prípadné poškodené slovo môžeme nájsť správne slovo v slovníku ako slovo s najmenšou vzdialenosťou od poškodeného slova. Takéto riešenie však môže podávať veľmi nepresné výsledky, keďže žiadnym spôsobom nezachytáva slová nachádzajúce sa okolo poškodeného slova, čím sa veľmi ľahko môže stať, že slovo je nesprávne opravené.

Druhým spôsobom je štúdium gramatiky a lexiky daného jazyka. Nájsť a odvodiť čo najviac zákonitostí, ktoré vplývajú na tvorbu slov a viet. Dané pravidlá potom implementovať do expertného systému. Pri tomto riešení nie je problém s novými slovami, ktoré systém predtým nepoznal. Vyžaduje si však pomerne komplikovanú analýzu jazyka a je náročné pri ňom prejsť z jedného jazyka na iný, s inými pravidlami. Okrem toho sa ľudská reč nedá úplne zachytiť pomocou pravidiel matematickej logiky.

Oprave poškodeného textu sa na Slovensku venoval Kočalka v [2]. V tejto práci využil fraktálnu reprezentáciu kontextu jazyka. Pri tejto metóde sú všetky reťazce v jazyku nad abecedou s n symbolmi reprezentované polohou v hyperkocke rozmeru $\lceil \log_2 n \rceil$. Vrcholy hyperkocky reprezentujú jednotlivé symboly abecedy. Pri hľadaní fraktálnej reprezentácie konkrétneho reťazca sa postupuje nasledovne:

1. Určí sa bod X_0, ktorý je umiestnený v strede hyperkocky; i:=0, j:=0
2. Bod X_i spojíme s vrcholom hyperkocky reprezentujúcej symbol na i-tom mieste v reťazci. Bod X_{i+1} určíme ako stred vzniknutej úsečky.
3. i:=i+1
4. Ak ešte nie sme na konci reťazca choď na bod 2 ináč vráť bod X_i, ktorý je fraktálnou reprezentáciou daného reťazca

Takto sa vytvorí fraktálna reprezentácia pre všetky podreťazce z trénovacej množiny o určitej dĺžke. Kočalka použil vo svojej práci dĺžku 10. Fraktálne reprezentácie sa musia uložiť v databáze. Pri opravovaní textu sa opäť vytvárajú fraktálne reprezentácie pre

jednotlivé podreťazce, pričom namiesto posledného symbolu sa vyskúšajú všetky symboly v abecede a za korektné sa považujú tie, ktorých fraktálna reprezentácia sa už vyskytuje vo vopred vytvorenej databáze alebo je najbližšie k nejakému údaju z databázy. Túto metódu kombinoval autor vo svojej práci so simulovaným žíhaním, pomocou ktorého našiel najkorektnejší reťazec. Výsledky práce sú pomerne nejasné, keďže neuvádza úspešnosť pri oprave textu.

Kapalla v práci [1] sa venoval dopĺňaniu diakritiky do slovenského textu. V tejto práci využil viacvrstvovú neurónovú sieť s učením pomocou spätného šírenia chýb (backpropagation). Vo svojej práci navyše kombinoval výsledky neurónovej siete s databázovým riešením spomenutým vyššie. Autor uvádza úspešnosť 96-98%. Táto metóda je však pomerne časovo náročná. V práci bola použitá len na dopĺňanie diakritiky, čo vo všeobecnosti nie je veľmi náročná úloha. V bežnom texte sa však môžu vyskytnúť oveľa rozmanitejšie chyby a všeobecné riešenie pomocou backpropagation by bolo neúnosne časovo náročné.

Iný zaujímavý spôsob využitia neurónových sietí, ktorý chcem vo svojej práci využiť ja, sa črtá v samoorganizujúcej sa mape (SOM – self-organizing map), ktorej sa venoval najmä Kohonen [3]. SOM je založená na učení bez učiteľa (t.j. algoritmus učenia nemá informáciu o požadovaných aktivitách výstupných neurónov v priebehu trénovania). Samoorganizácia je založená na tzv. Hebbovom pravidle učenia: *"When an axon of cell A excites cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells so that A's efficiency as one of the cells firing B is increased."*

SOM pozostáva z N neurónov, pričom ku každému prislúcha n rozmerný váhový vektor $w_i = (w_{i1} ... w_{in})$. Pre vstup $x = (x_1 ... x_n)$ sa výstup neurónu vypočíta ako

$$y_i = \sum w_{ij} x_i$$

Pri trénovaní siete dávame na vstup jednotlivé vzory a pre každý nájdeme víťazný neurón – neurón, ktorý najviac reaguje na vstup x. Teda $i^* = \arg\max(w_i x)$, kde i* je index víťazného neurónu. Po nájdení víťaza nasleduje adaptácia váh – učenie. To zabezpečí, že váhové vektory víťazného neurónu a jeho topologických susedov sa posunú smerom k vstupu podľa vzťahu

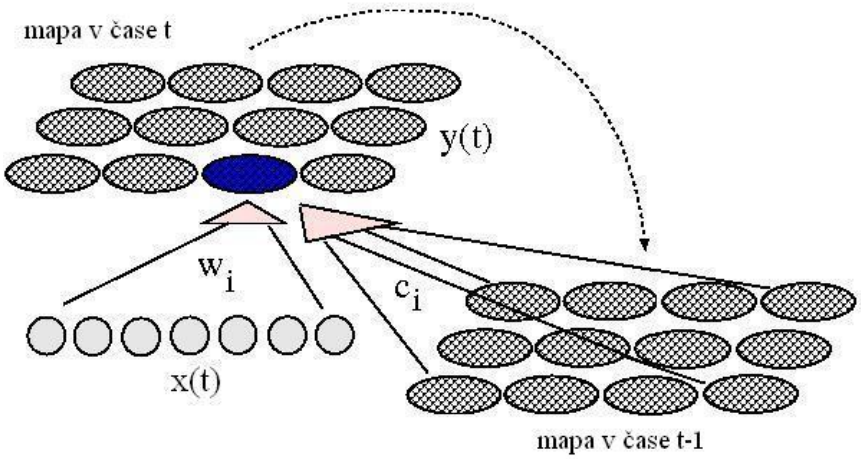
$$w_i(t+1) = w_i(t) + \gamma(t) \cdot h(i^*, i) \cdot (x(t) - w_i(t))$$

Funkcia $\gamma(t)$ predstavuje rýchlosť učenia, ktorá s časom klesá k nule. Funkcia okolia $h(i^*,i)$ definuje rozsah kooperácie medzi neurónmi, t.j. do akej miery budú váhové vektory v okolí víťaza adaptované. Často používanou funkciou je gaussovské okolie

$$h(i^*,i) = \exp\left(-\frac{d_E^2(i^*,i)}{\sigma^2(t)}\right)$$

kde $d_E^2(i^*,i)$ predstavuje euklidovskú vzdialenosť neurónov i* a i v mriežke. Parameter $\sigma(t)$ s časom klesá k nule, čím sa zabezpečuje zmenšovanie okolia počas učenia.

Pre korekciu textu je dôležité zachytiť kontext v ňom. Pre túto úlohu sú vhodné niektoré implementácie SOM, napríklad RecSOM alebo MSOM.



Obr. 1. RecSOM: Každý neurón v mape je asociovaný s váhovým vektorom *w*, ktorý má rovnaký rozmer ako vstup. Ďalej je asociovaný s kontextovým vektorom *c*, s ktorým sa spája mapa z predchádzajúceho časového kroku.

V RecSOM (Recursive SOM) [6] je každý neurón asociovaný s dvoma váhovými vektormi - w_i asociovaný so vstupom podobne ako v pôvodnej SOM a c_i asociovaný s kontextom $y(t-1) = (y_1(t-1)...y_n(t-1))$, obsahujúcim aktivácie $y_i(t-1)$ z predchádzajúceho kroku.

Aktivita neurónu sa počíta ako

$$y_i(t) = \exp(-d_i(t)),$$

kde

$$d_i(t) = \alpha \cdot \|x(t) - w_i\|^2 + \beta \cdot \|y(t-1) - c_i\|^2,$$

Parametre $\alpha > 0$ a $\beta > 0$ sú parametre, ktoré ovplyvňujú efekt vstupu a kontextu.

Víťaz v sieti je neurón s najvyššou aktivitou $i^*(t) = \arg\min(d_i)$.

Oba váhové vektory sa upravujú podobne ako v pôvodnej SOM.

$$\Delta w_i = \gamma \cdot h(i^*(t), i) \cdot (x(t) - w_i),$$

$$\Delta c_i = \gamma \cdot h(i^*(t), i) \cdot (y(t-1) - c_i).$$

V MSOM (Merge SOM) [5] je taktiež každý neurón asociovaný s dvoma váhovými vektormi. Tu platí

$$d_i(t) = (1-\alpha) \cdot \|x(t) - w_i\|^2 + \alpha \cdot \|cd(t) - c_i\|^2$$

Parameter α určuje vplyv vstupu a kontextu, $cd(t)$ je kontext deskriptor

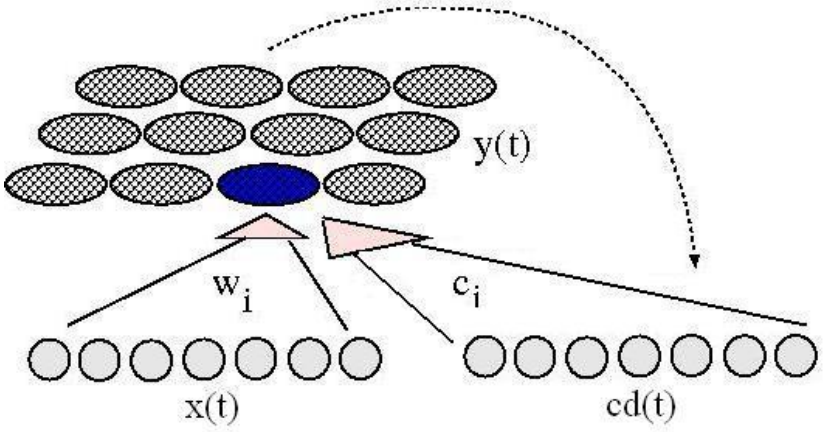
$$cd(t) = (1-\beta) \cdot w_{i^*(t-1)} + \beta \cdot c_{i^*(t-1)},$$

kde $i^*(t-1)$ je víťazný neurón z predchádzajúceho kroku. Obvyklá hodnota pre parameter β je 0.5. Váhy sa upravujú opäť Hebbovým pravidlom učenia

$$\Delta w_i = \gamma_1 \cdot h(i^*(t), i) \cdot (x(t) - w_i),$$

$$\Delta c_i = \gamma_2 \cdot h(i^*(t), i) \cdot (cd(t) - c_i).$$

γ_1 a γ_2 sú rýchlosti učenia. Zvyčajnou hodnotou je $\gamma_1 = \gamma_2 = 0.05$



Obr. 2. MSOM: Každý neurón v mape je asociovaný s váhovým vektorom *w*, ktorý má rovnaký rozmer ako vstup. Ďalej je asociovaný s kontextovým vektorom *c*, s ktorým sa spája kontext deskriptor *cd*. Ten popisuje kontext.

RecSOM aj MSOM sa ukazujú ako vhodné pre riešenie danej problematiky vďaka schopnosti zachytiť kontext. Ich slabinou je iste časová náročnosť. V tomto sa ukazuje ako lepšia MSOM, pri ktorej je rozmer váhové ho vektora asociovaného s kontextom rovný rozmeru vstupu, zatiaľ čo u RecSOM je to počet neurónov v sieti.

V práci bude veľmi dôležité definovať vzdialenosť dvoch reťazcov – mieru podobnosti dvoch reťazcov. Podobnosti dvoch reťazcov sa venuje Kohonen v [4]. Tu uvádza vzdialenosť dvoch reťazcov ako *Levenshteinovu vzdialenosť* (Levenshtein distance) alebo *Feature distance*. Levenshteinova vzdialenosť (LD) pre reťazce A a B je definovaná ako

$$LD(A,B) = \min\{a(i) + b(i) + c(i)\}$$

Tu reťazec B dostaneme z reťazca A pomocou $a(i)$ výmen, $b(i)$ vsunutí a $c(i)$ vymazaní jednotlivých symbolov. Existuje nekonečne veľa kombinácií $\{a(i), b(i), c(i)\}$ a kombináciu s minimálnou Levenshteinovou vzdialenosťou je možné nájsť napríklad dynamickým programovaním.

Keďže rozdielne typy chýb sa môžu vyskytovať s rozdielnou pravdepodobnosťou, je porovnávanie zvyčajne lepšie, ak sa jednotlivé operácie uvažujú s rôznymi váhami. To vedie k *váhovanej Levenshteinovej vzdialenosti* (weighted Levenshtein distance - WLD)

$$WLD(A,B) = \min\{pa(i) + qb(i) + rc(i)\},$$

kde koeficienty p, q, r je možné zistiť z takzvanej *confusion matrix* abecedy, ako prevrátenú hodnotu pravdepodobnosti pre príslušnú chybu.

Feature distance (FD) sa definuje pomocou N-gramov. N-gram je podreťazec N po sebe nasledujúcich symbolov (zvyčajne sa používa N=2 – bigram alebo digram, či N=3 – trigram). “Okno”, z ktorého sa zostaví N-gram by sa malo pohybovať po celom reťazci tak, aby sa jednotlivé N-gramy prekrývali.

Nech n_a, n_b je počet N-gramov v reťazcoch A a B a nech e je počet zhodných N-gramov. Feature disatnce (FD) je definovaná ako

$$FD(A,B) = \max(n_a, n_b) - e$$

Táto metóda podáva o niečo slabšie výsledky ako WLD, ale je rýchlejšia.

2 Trénovanie siete

V tejto kapitole stručne popíšem, ako som postupoval pri trénovaní neurónovej siete na korpuse jazyka. Zdôvodním tu výber siete, parametrov a zhodnotím niektoré výsledky siete.

2.1 Výber jazyka

Za jazyk, na ktorom budem sieť trénovať, som si vybral angličtinu. Dôvodom tohto výberu bol menší počet symbolov (26) oproti iným jazykom. Okrem základných písmen anglickej abecedy som pri trénovaní pridal do symbolovej sady aj medzeru. Snažil som sa vyhnúť iným symbolom v texte ako aj veľkým písmenám preto, aby som čo najviac urýchlil už i tak pomalé trénovanie siete.

Trénoval som sieť na knihe reprezentujúcej krásnu literatúru. Všetky veľké písmená boli prevedené na ich malé ekvivalenty. Všetky symboly okrem 26 znakov anglickej abecedy a medzery boli vyhodené. Trénovací text pozostával po týchto úpravách z 102 804 symbolov.

2.2 Výber siete

Pre natrénovanie siete na jazyku pripadá do úvahy niekoľko druhov neurónových sietí. Podmienkou je, aby táto sieť dokázala zachytiť kontext na vstupe. Zachytenie kontextu je v jazyku kľúčové pre porozumenie textu a odhalenie prípadných chýb v ňom. Existuje niekoľko druhov neurónových sietí, ktoré sú schopné zachytiť kontext. Ja som si vybral sieť MSOM, pretože dosahuje pri zachytení kontextu veľmi dobré výsledky v porovnaní s ostatnými sieťami pri slušnej časovej náročnosti. Časová zložitosť MSOM je $O(N \cdot n)$, ak N je počet neurónov v sieti a n je rozmer vstupu. Pre porovnanie zložitosť RecSOM je $O(N \cdot n + N^2)$. Pritom MSOM dosahuje pri rovnakom počte neurónov porovnateľné výsledky ako RecSOM. Neurónovú sieť MSOM navrhli Barbara Hammer

a Marc Strickert. Veľmi dobre je popísaná v [5] a v tejto kapitole sa budem na ich prácu viackrát odvolávať.

2.3 Počet neurónov v sieti

Samozrejme, že platí čím viac neurónov v sieti, tým lepšie sú výsledky siete. Obmedzení sme len výpočtovou silou dnešných počítačov. Pre sieť MSOM sa ukazuje ako ešte reálna veľkosť 10000 neurónov. Sieť som trénoval ako štvorec. Teda pre 10000 neurónov by to bola sieť 100x100 neurónov. Pre porovnanie som trénoval aj sieť s 1600 neurónmi (t.j. 40x40 neurónov).

2.4 Voľba parametrov

Parametre γ_1, γ_2 určujú rýchlosť učenia. Ak je ich hodnota príliš vysoká, učenie siete nie je celkom presné. Váhy neurónov sa veľmi rýchlo menia a nie je možné jemné doladenie siete. Ak je hodnota týchto parametrov veľmi malá, učenie siete zase trvá príliš dlho. Je preto treba nájsť vhodnú rovnováhu medzi týmito dvoma parametrami. Strickert a Hammer odporúčajú $\gamma_1 = \gamma_2$ a vo svojich pokusoch používajú hodnoty okolo 0.05. Preto som rovnakú hodnotu použil vo svojich pokusoch aj ja.

Hodnotu parametra β som nechal na 0.5. Táto hodnota de facto vyjadruje, ako rýchlo sieť zabúda svoj kontext. Strickert a Hammer odporúčajú hodnotu 0.5 ako najvhodnejšiu. Navyše tento parameter úzko súvisí s parametrom α, takže jeho výber sa dá ovplyvniť aj zmenou tohto parametra.

Hodnota parametra α de facto vyjadruje ako významný má byť vplyv kontextu oproti symbolu, ktorý práve prišiel na vstup. Na začiatku trénovania sieť nemá v podstate žiadnu informáciu o tom, ako nakladať s kontextom. Preto je vhodné parameter α nastaviť na malé kladné číslo. Postupne, ako sa sieť učí, je vhodné parameter α zvyšovať. Ale pokiaľ zvyšovať tento parameter tak, aby sme dosiahli najlepšie výsledky ? Cieľom je rovnomerné rozloženie aktivity neurónov. Nechceme, aby v jednej oblasti siete boli príliš aktívne

neuróny a v inej zase neaktívne. Hammer a Strickert preto navrhli, aby sa α zvyšovala tak, aby maximalizovala entropiu aktivácie neurónov.

Na začiatku nastavíme $\alpha = 0.05$. Ďalej hodnotu α zvyšujeme o malú hodnotu, povedzme 0.01, pokiaľ sa entropia zvyšuje. Tým sa dosahuje väčší vplyv kontextu. Ak sa entropia zníži, musíme naopak znížiť hodnotu parametra α. Ako výsledok by sme mali dosiahnuť na konci trénovania, že aktivita neurónov je rovnomerne rozdelená po celej sieti. Pri mojich pokusoch sa parameter α ustálil na hodnote 0.4.

Posledným z parametrov je parameter σ. Tento parameter určuje veľkosť okolia okolo víťazného neurónu, v ktorom sú váhy neurónov upravované. Jeho hodnota by mala byť na začiatku vysoká a časom by mala klesať, aby sa zabezpečila špecializácia neurónov v malej oblasti. Na začiatku som tento parameter nastavil na hodnotu rovnú 0.4 násobku rozmeru siete. To znamená pre sieť 100x100 to bola hodnota 40 a pre sieť 40x40 to bolo 0.16. Následne som nechal túto hodnotu klesať až k 0.5.

Hodnoty parametrov α a σ som nechal klesať k svojim finálnym hodnotám v prvej polovici trénovania. V druhej polovici boli tieto hodnoty ustálené a tak bolo umožnené precízne natrénovanie siete a doladenie detailov.

2.5 Záver

Pri trénovaní som nechal sieť prejsť trénovacou množinou dvakrát. Teda pri prvom prejdení sa ešte upravovali parametre α a σ a neskôr už boli stabilizované. Pridanie ešte jedného cyklu sa ukázalo už ako zbytočné. Trénovanie siete bolo veľmi časovo náročné. Natrénovanie siete 100x100 trvalo na bežnom počítači asi 5 hodín. Po natrénovaní je jasne vidieť špecializáciu oblastí siete na podobné vstupy. Podrobné vyhodnotenie výsledkov sa však ukáže až pri aplikácii siete na problém opravy textu.

3 Základné definície

Aby sme mohli úspešne opravovať text potrebujeme vedieť pre ľubovoľný reťazec určiť pravdepodobnosť, s akou je korektný. Teda potrebujeme si definovať funkciu, ktorá by vedela zo skupiny reťazcov vybrať ten, ktorý je s najväčšou pravdepodobnosťou správny.

Definicia 3.1: Nech Σ je abeceda symbolov. Nech $e : \Sigma^* \rightarrow R$ je funkcia s vlastnosťou

$$\forall s, s' \in \Sigma^* \; e(s) > e(s')$$

$$\Leftrightarrow$$

pravdepodobnosť, že reťazec s je korektný je vyššia ako

pravdepodobnosť, že reťazec s' *je korektný*

Funkciu *e* budeme nazývať energetickou funkciou nad abecedou Σ. Hodnotu *e(s)* nazveme energetickou hodnotou reťazca *s* alebo skrátene len energia *s*.

Aby bolo opravovanie reťazca *s* korektné, je potrebné nielen nájsť reťazec v danom jazyku, ktorý má najvyššiu energiu. Je potrebné, aby opravený reťazec bol zároveň dostatočne podobný na pôvodný reťazec. Môžeme si zadefinovať vzdialenosť dvoch reťazcov.

Definicia 3.2: Nech Σ je abeceda symbolov. Nech $s = a_1, a_2 ... a_n$ je reťazec nad touto abecedou t.j. $s \in \Sigma^*$ a $a_1, a_2 ... a_n \in \Sigma$.

(a) Hovoríme, že reťazec $s' = a_1 ... a_{i-1} b a_{i+1} ... a_n$ vznikol z reťazca *s* zámenou i-teho symbolu symbolom b nad abecedou Σ, ak $b \in \Sigma \wedge b \neq a_i$.

(b) Hovoríme, že reťazec $s' = a_1 ... a_{i-1} b a_i a_{i+1} ... a_n$ vznikol z reťazca *s* vsunutím symbolu *b* na i-te miesto nad abecedou Σ, ak $b \in \Sigma$.

(c) Hovoríme, že reťazec $s' = a_1 ... a_{i-1} a_{i+1} ... a_n$ vznikol z reťazca *s* vymazaním i-teho symbolu.

Mutáciou reťazca *s* je ľubovoľný reťazec $s' \in \Sigma^*$, pre ktorý platí $s \neq s'$. Pre každý reťazec s' existuje postupnosť reťazcov $s_1 ... s_n$ takých, že

$$s = s_1 \wedge s' = s_n \wedge$$

$$\forall i \in \{1 \ldots n-1\}\ s_{i+1} \textit{ vznikol z } s_i \textit{ zámenou, vsunutím alebo vymazaním symbolu.}$$

Hovoríme, že reťazec s' vznikol z reťazca s postupnosťou operácií zámeny, vsunutia a vymazania. Takýchto postupností existuje pre každú dvojicu nekonečne veľa.

Definicia 3.3: Nech Σ je abeceda symbolov. Levenshteinovou vzdialenosťou nazveme funkciu $LD : \Sigma^* \times \Sigma^* \rightarrow R$ definovanú ako

$$LD(s,s') = \min\{n \mid n = a+b+c \wedge \quad \textit{reťazec} \quad s' \quad \textit{vznikol} \quad \textit{z reťazca} \quad \textit{s postupnosťou a zámen, b vsunutí a c vymazaní symbolu}\}$$

Nech $s, s', s'' \in \Sigma^*$ sú reťazce a nech $LD(s,s') < LD(s,s'')$. Budeme hovoriť, že reťazec s' je bližšie k reťazcu s ako reťazec s'' a že reťazec s'' je od reťazca s ďalej ako reťazec s'.

Levenshteinovu vzdialenosť (ďalej len „vzdialenosť“) dvoch reťazcov možno nájsť pomocou dynamického programovania. Ak hodnota vzdialenosti pre ich prefixové podreťazce (podreťazce začínajúce prvým symbolom) dĺžky i-1 a j-1 je *d*1, vzdialenosť prefixových podreťazcov dĺžky i a j bude najviac *d*1, ak sú symboly na i-tom a j-tom mieste rovnaké a najviac *d*1+1, ak sú symboly na i-tom a j-tom mieste rozdielne. Ak hodnota vzdialenosti prefixových reťazcov dĺžky i a j-1 je *d*2, vzdialenosť prefixových podreťazcov dĺžky i a j bude najviac *d*2+1. Ak hodnota vzdialenosti prefixových reťazcov dĺžky i-1 a j je *d*3, vzdialenosť prefixových podreťazcov dĺžky i a j bude najviac *d3*+1. Teda vzdialenosť prefixových podreťazcov dĺžky i a j nájdeme ako minimum z *d*3+1, *d*2+1 a *d*1 alebo *d*1+1 (podľa toho či sú symboly na i-tom a j-tom mieste totožné, ako je uvedené vyššie). Algoritmus pre výpočet vzdialenosti je uvedený nižšie.

Ďalšie možnosti ako definovať funkciu vzdialenosti reťazcov je váhovaná Levenshteinova vzdialenosť (weighted Levenshtein distance - WDL).

Definicia 3.4: Nech Σ je abeceda symbolov. Váhovanou Levenshteinovou vzdialenosťou nazveme funkciu $WLD : \Sigma^* \times \Sigma^* \rightarrow R$ definovanú ako

$$WLD(s,s') = \min\{n \mid n = p \cdot a + q \cdot b + r \cdot c \wedge \quad \textit{reťazec} \quad s' \quad \textit{vznikol} \quad \textit{z reťazca s postupnosťou a zámen, b vsunutí a c vymazaní symbolu}\},$$ kde p, q, r sú konštanty.

Kohonen v [4] definuje hodnoty konštánt *p*, *q*, *r* ako prevrátené hodnoty pravdepodobnosti pre príslušnú chybu. Takéto hodnoty je zrejme možné zistiť náročnou analýzou veľkého množstva textu. Za určitých podmienok ale vieme približne odhadnúť aj bez takejto analýzy aspoň približné hodnoty týchto konštánt.

```
begin
   D(0,0) := 0;
   for i := 1 step 1 until length(s) do D(i,0) := D(i-1,0) + 1;
   for i := 1 step 1 until length(s‘) do D(0,j-1) + 1;
   for j := 1 step 1 until length(s) do
      for j := 1 step 1 until length(s‘) do
      begin
         if s(i) == s‘(j) m1 := D(i-1,j-1) + 1
         else m1 := D(i-1,j-1);
         m2 := D(i,j-1) + 1;
         m3 := D(i-1,j) + 1;
         D(i,j) := min(m1,m2,m3);
      end;
   LD := D(length(s),length(s’));
end;
```

Algoritmus pre výpočet Levenshteinovej vzdialenosti

Ak predpokladáme, že opravovaný text bol napísaný na klávesnici, môžeme hodnoty konštánt určiť ako vzdialenosť príslušných kláves na klávesnici v prípade zámeny. V prípade vsunutia maximálna vzdialenosť zo vzdialeností klávesy zodpovedajúcej vsunutému symbolu od klávesy zodpovedajúcej symbolu naľavo alebo napravo od vsunutého symbolu. V prípade vymazania by táto hodnota bola defaultne 1. Ak bol text získaný pomocou rozpoznávania reči alebo OCR, môžu sa konštanty určiť ako pravdepodobnosti príslušných chýb, ktoré vieme, že pri rozpoznávaní reči alebo OCR môžu nastať.

Ak by sme chceli uvažovať aj o slovoslede v texte, môžeme definovať vzdialenosť úplne ináč. Ak sú vymenené dve slová v reťazci, nevypočítame vzdialenosť ako Levenshteinovu vzdialenosť, ale určíme nejakú konštantu.

Definicia 3.5: Nech e je energetická funkcia nad abecedou Σ. Nech d je funkcia vzdialenosti reťazcov nad abecedou Σ. Nech s je reťazec nad touto abecedou. Definujme funkciu $e_s : \Sigma^* \to R$ tak, že $\forall s' \in \Sigma^*$ platí:

(a) $e_s(s') = e_s(s'')\ ak\ e(s') = e(s'') \wedge d(s', s) = d(s'', s)$

(b) $e_s(s') < e_s(s'')\ ak\ e(s') < e(s'') \wedge d(s', s) \geq d(s'', s)$

(c) $e_s(s') > e_s(s'')\ ak\ e(s') > e(s'') \wedge d(s', s) \leq d(s'', s)$

Funkciu e_s nazveme energetickou funkciou nad abecedou Σ vzhľadom na reťazec s. Hodnotu $e_s(s')$ nazveme energetickou hodnotou reťazca s' vzhľadom na reťazec s alebo skrátene len energia s' vzhľadom k s.

Z vlastností funkcie e_s vyplýva, že z dvoch reťazcov s rovnakou energiou ale rozdielnou vzdialenosťou od reťazca s bude ich energia vzhľadom k s vyššia u toho bližšieho. Zároveň, ak je vzdialenosť dvoch reťazcov od reťazca s rovnaká, ich energia vzhľadom k s bude vyššia u toho s vyššou energiou e. Sú to presne vlastnosti, ktoré potrebujeme.

3.1 Implementácia funkcií e a e_s

Pomocou MSOM sa nám podarilo zachytiť určité vlastnosti jazyka. Takto natrénovaná sieť dokáže pre nejaký reťazec predikovať, aký by mohol byť nasledujúci symbol v reťazci. Túto vlastnosť môžeme využiť pri implementácii energetickej funkcie.

Každý neurón v MSOM má v sebe tabuľku, kde má uložené pravdepodobnosti výskytu jednotlivých symbolov abecedy ako nasledujúceho symbolu v reťazci za predpokladu, že daný neurón bol aktivovaný.

Definicia 3.6: Nech je daná MSOM a nech $N_1 ... N_n$ sú jednotlivé neuróny v sieti. Nech Σ je abeceda symbolov v jazyku, na ktorom bola sieť trénovaná. Nech funkcia $P_i : \Sigma \to \langle 0,1 \rangle$ je funkciou pravdepodobnosti a nech hodnota $P_i(a)$ označuje pravdepodobnosť, že ak je aktivovaný neurón N_i pri prechádzaní reťazca od jeho prvého symbolu neurónovou sieťou, nasledujúci symbol bude a. Nech $s = a_1 a_2 ... a_m$ je reťazec nad

abecedou Σ t.j. $s \in \Sigma^*$ a $a_1, a_2 ... a_m \in \Sigma$. Nech $i(1)...i(m)$ sú indexy neurónov, ktoré boli aktivované pri prechádzaní reťazca, t.j. $i(j)$ je index neurónu, ktorý bol aktivovaný po posunutí symbolu a_j sieti (teda neurón s týmto indexom predikuje symbol na pozícii j+1). Potom hodnotu $P_{i(j)}(a_j)$ nazveme ľavá čiastková energia *i*-teho symbolu reťazca *s*.

Veta 3.1: Nech $s = a_1 a_2 ... a_m$ je reťazec nad abecedou Σ t.j. $s \in \Sigma^*$ a $a_1, a_2 ... a_m \in \Sigma$. Nech $l_1 ... l_m$ sú ľavé čiastkové energie jednotlivých symbolov reťazca. Teda g_j je ľavá čiastková energia *j*-teho symbolu reťazca *s*. Potom

$$e(s) = \frac{\sum_{j=1}^{m-1} l_j}{m-1}$$

je energetickou funkciou nad abecedou Σ pre množinu všetkých reťazcov ktoré sú dostatočne dlhé.

Platnosť predchádzajúcej vety vyplýva priamo z implementácie MSOM a jej vlastností. Otázkou zostáva, čo znamená slovné spojenie dostatočne dlhý. Ako vieme, to ktorý neurón sa v sieti aktivuje, závisí od aktivácie neurónov v predchádzajúcich krokoch. Je teda zrejmé, že pre prvý symbol v reťazci nemáme predstavu, aké mohli byť aktivácie pred týmto symbolom. Preto to, ktorý neurón sa aktivoval, nie je pre nás dostatočne relevantné a teda ani pravdepodobnosti pre výskyt nasledujúcich symbolov, ktoré nám určuje tento neurón, nie sú dostatočne dôveryhodné.

Ako upraviť funkciu *e* tak, aby bola presnejšia aj pre kratšie reťazce ? Možnou implementáciou môže byť napríklad:

$$e_1(s) = \frac{\sum_{j=k}^{m-1} l_j}{m-k},$$

kde k>0 je vlastne index symbolu v reťazci, od ktorého začneme brať predikciu aktivovaného neurónu za dostatočne dôveryhodnú.

Ďalšou možnou implementáciou je:

$$e_2(s) = \frac{\sum_{j=1}^{m-1} f(j) \cdot l_j}{\sum_{j=1}^{m-1} f(j)},$$

kde $f : N \to \langle 0,1 \rangle$ je neklesajúca funkcia. Predpoklad, že funkcia *f* je neklesajúca vychádza z toho, že presnosť predikcie ďalšieho symbolu pomocou neurónovej siete neklesá pri rastúcom prefixe, ktorý sme poskytli neurónovej sieti. Ak funkciu f definujeme ako $f(i) = \begin{matrix} 0 \; ak \;\; i \le k \\ 1 \; ak \;\; i > k \end{matrix}$ tak je vlastne funkcia e_2 totožná funkcii e_1.

Je zrejmé, že funkcia e_2 je funkciou, ktorá pri vhodnej implementácii funkcie *f* je energetickou funkciou pre ľubovoľne dlhé reťazce. Tomu, aká je najvhodnejšia funkcia *f,* sa budeme venovať neskôr.

Zatiaľ sme hovorili len o neurónovej sieti natrénovanej na texte zľava doprava (teda smerom akým text čítame). Podrobnejšie štúdium jazyka však ukazuje, že rovnako ako sú pre určenie nejakého symbolu v texte podstatné symboly pred ním, sú podstatné aj symboly za ním. Budeme hovoriť o ľavom a pravom kontexte. Ukážme si to na nasledujúcom príklade:

Tento príklad ukazuje význa? ľavého kontextu.

Aký symbol by sme doplnili v predchádzajúcej vete namiesto otáznika? Skúsme sa pozerať len na symboly nachádzajúce sa za otáznikom. Pri takomto pohľade nie je možné určiť, aký symbol by bolo vhodné doplniť namiesto otáznika. Ak sa však pozrieme na text pred otáznikom, je nám okamžite jasné, že za otáznik treba doplniť písmeno *m*. Pozrime sa na ešte jeden príklad.

Tento príklad ?kazuje význam pravého kontextu.

Aký symbol by sme doplnili tentokrát za otáznik? Ak sa budeme pozerať len na symboly pred otáznikom, ako sme to spravili v predchádzajúcom príklade, nepomôžeme si. Letmý pohľad na text napravo od otáznika však ukazuje, že na mieste otáznika má byť písmeno *u*.

Predchádzajúce príklady naznačujú, že v energetickej funkcii bude potrebné zohľadňovať význam pravého aj ľavého kontextu. Vo funkcii e_2, ktorú sme definovali vyššie, uvažujeme len o ľavom kontexte – uvažujeme MSOM trénovanú na texte zľava doprava. Význam pravého kontextu nás privádza k potrebe natrénovať neurónovú sieť aj sprava doľava. Pomocou takto natrénovanej siete potom môžeme definovať energetickú funkciu zohľadňujúcu pravý kontext. Budeme hovoriť o pravej a ľavej sieti, podľa toho ktorý kontext sieť zachytáva.

Veta 3.2: Nech je daná MSOM a nech $N_1^B ... N_n^B$ sú jednotlivé neuróny v sieti. Nech Σ je abeceda symbolov v jazyku, na ktorom bola sieť trénovaná. Nech funkcia $P_i^B : \Sigma \to \langle 0,1 \rangle$ je funkciou pravdepodobnosti a nech hodnota $P_i^B(a)$ označuje pravdepodobnosť, že ak je aktivovaný neurón N_i^B pri prechádzaní reťazca od jeho posledného symbolu smerom k prvému, nasledujúci symbol bude *a*. Nech $s = a_1 a_2 ... a_m$ je reťazec nad abecedou Σ, t.j. $s \in \Sigma^*$ a $a_1, a_2 ... a_m \in \Sigma$. Nech $i_1 ... i_m$ sú indexy neurónov, ktoré boli aktivované pri prechádzaní reťazca, t.j. i_j je index neurónu, ktorý bol aktivovaný po posunutí symbolu a_{m+1-j} sieti (teda neurón s týmto indexom predikuje symbol na pozícii m-j).

Potom hodnotu $P_{i(j)}^B(a_j)$ nazveme pravá čiastková energia *i*-teho symbolu reťazca *s*.

Veta 3.3: Nech $s = a_1 a_2 ... a_m$ je reťazec nad abecedou Σ t.j. $s \in \Sigma^*$ a $a_1, a_2 ... a_m \in \Sigma$. Nech $p_1 ... p_m$ sú pravé čiastkové energie jednotlivých symbolov reťazca. Teda p_j je pravá čiastková energia *j*-teho symbolu reťazca *s*. Potom

$$e_2^B(s) = \frac{\sum_{j=1}^{m-1} f(m-j) \cdot p_j}{\sum_{j=1}^{m-1} f(j)}$$

je tiež energetickou funkciou nad abecedou Σ pre množinu všetkých reťazcov pre vhodnú neklesajúcu funkciu *f*.

Teda máme dve energetické funkcie, ktoré zohľadňujú význam ľavého a pravého kontextu každá zvlášť. Vhodné by bolo zaviesť jednu energetickú funkciu, ktorá by zohľadňovala význam oboch kontextov:

$$e_3(s) = e_2(s) + e_2^B(s)$$

Zaveďme si teraz energetickú funkciu vzhľadom k nejakému danému reťazcu $s \in \Sigma^*$.

Veta 3.4: Nech Σ je abeceda symbolov. Nech $s' \in \Sigma^*$ potom

$$e_s(s') = e_3(s') - c \cdot d(s', s),$$

kde *c* je konštanta, je energetickou funkciou nad abecedou Σ vzhľadom k reťazcu *s*.

Overením podmienok (a), (b), (c) z definície energetickej funkcie vzhľadom k reťazcu *s* ľahko overíme, že veta je platná. Najvhodnejšiu hodnotu konštanty *c* zistíme pokusmi neskôr.

4 Prvé pokusy o korekciu textu

V tejto kapitole uvediem výsledky mojich prvých pokusov o korekciu textu a ako som postupoval.

4.1 Spôsob počítania energie

Na výpočet energie som použil funkciu e_3 spomínanú v predchádzajúcej kapitole. Funkciu f, ktorá tu vystupuje, som definoval nasledovne: $f(i) = \begin{matrix} 0\ ak\ \ i \le k \\ 1\ ak\ \ i > k \end{matrix}$. Konštantu k je potrebné určiť tak, aby sme pre každý symbol s indexom väčším ako k vedeli s istotou povedať, že predpoveď víťazného neurónu je dostatočne dôveryhodná.

Môžeme pozorovať, že sieť dokáže zachytiť určitú hĺbku kontextu. Ak zmeníme jeden symbol v reťazci pochopiteľne niekoľko nasledujúcich víťazných neurónov bude iných než pre pôvodný reťazec. O niekoľko symbolov ďalej však začnú byť víťazné neuróny opäť totožné s víťaznými neurónmi v pôvodnom reťazci. Symbol nachádzajúci sa v kontexte hlbšie je totiž pri hľadaní víťaza menej významný ako symboly, ktoré prišli na vstup až po ňom.

Definicia 4.1: Nech je daná natrénovaná neurónová sieť MSOM. Nech k je minimálne číslo také, že ak zmutujem v ľubovoľnom reťazci jeden symbol, tak na (k+1)-om symbole od zmutovaného sa aktivuje vždy rovnaký neurón ako v pôvodnom reťazci. Toto číslo nazveme maximálna hĺbka kontextu siete.

Práve maximálna hĺbka kontextu siete sa ukazuje ako vhodná hodnota pre konštantu k vystupujúcu vo funkcii f. Práve preto, že nezávisle na tom či pred alebo za daný reťazec vložíme ešte nejaké iné symboly, na symboloch, ktoré sú vzdialené aspoň k od začiatku pôvodného reťazca, vždy vyhrá rovnaký neurón v ľavej sieti a rovnako i pre symboly, ktoré sú vzdialené aspoň k od konca pôvodného reťazca, vždy vyhrá rovnaký neurón v pravej sieti. Hodnotu parametra k musíme určiť pokusmi. Pri mojich pokusoch som pre obe siete - s 10000 neurónmi i s 1600 neurónmi dosiahol hodnotu $k = 4$.

Ako funkciu vzdialenosti dvoch reťazcov som použil Levenshteinovu vzdialenosť. V súčasnej situácii, keď sú už i tak veľké problémy s časovou náročnosťou výpočtu, nie je rozumné uvažovať zložitejšie funkcie vzdialenosti.

4.2 Časová zložitosť

Pri mojich pokusoch sa ako veľká prekážka ukázala rýchlosť, či skôr pomalosť siete. Vybral som z trénovacieho textu jednu vetu a pokúsil sa o vypočítanie energií jej mutácií. Táto veta pozostávala z 88 symbolov. Vypočítanie energie pre túto vetu trvalo asi 6 sekúnd v sieti s 10000 neurónmi.

My však potrebujeme vypočítať energiu nielen pre túto vetu, ale aj pre všetky jej mutácie. Ak zoberieme len také mutácie tejto vety, kde je len jeden zmutovaný symbol, dostaneme celkový počet mutácií 4779. Presnejšie $88 \cdot 26 = 2288$ zámen, $89 \cdot 27 = 2403$ vsunutí a 88 vymazaní symbolu. Všeobecne pre reťazec dĺžky n je takýchto mutácií $n \cdot 26 + (n+1) \cdot 27 + n$. Samozrejme, že skutočný počet jedinečných mutácií môže byť o niečo menší, pretože niektorými mutáciami môžu vzniknúť rovnaké reťazce. Celkový počet jedinečných mutácií potom závisí už od konkrétneho reťazca.

Ak by sme teda chceli vypočítať energiu pre všetky mutácie vyššie spomínanej vety, trval by tento výpočet $4779 \cdot 6 = 28674$ sekúnd, čo je asi 8 hodín. Pritom stále vypočítame len energiu pre mutácie reťazca s jedným zmutovaným symbolom. To znamená, že za žiadnych okolností nemáme šancu odhaliť, či nie je v reťazci viac chýb. Ak by sme chceli počítať energiu aj pre reťazec s viacerými mutáciami, bol by výpočet už neúnosne náročný. Je preto jasné, že je treba nejako vyriešiť časovú zložitosť tohto výpočtu.

Pozrime sa, aká je zložitosť počítania energií mutácií reťazca (a teda problému korekcie textu) všeobecne. Ak máme reťazec dĺžky n, vieme jeho energiu vypočítať v čase $O(n)$. To zodpovedá tomu, že potrebujeme vypočítať čiastkové energie pre každý z jeho n symbolov.

Ako som už vyššie demonštroval, počet mutácií reťazca s jedným zmutovaným symbolom oproti pôvodnému reťazcu je $O(n)$. Teda zložitosť výpočtu energií pre všetky takéto mutácie je $O(n^2)$. Ak by sme chceli mať aj mutácie s dvoma zmutovanými

symbolmi, bola by zložitosť celého výpočtu $O(n^3)$. Takýto výpočet je už absolútne nezvládnuteľný.

4.3 Výsledky korekcie

Aby som získal predstavu o presnosti siete, nechal som zbehnúť tento náročný výpočet a sledoval som výstup. Keď som nechal zbehnúť výpočet v sieti s 1600 neurónmi, dostal som 36 mutácií s vyššou energiou ako pôvodný reťazec. V sieti s 10000 neurónmi bolo takýchto mutácií už len 11, ale aj to je stále priveľa. Cieľom by malo byť, aby ani jedna nekorektná mutácia korektného reťazca nemala vyššiu energiu ako pôvodný správny reťazec. Pri zvýšení počtu neurónov z 1600 na 10000 sa výsledky siete veľmi nezlepšili. Z týchto výsledkov je zrejmé, že pri vyššom počte neurónov by sme už nedosiahli oveľa lepší výsledok. K zvýšeniu počtu neurónov nás už nepustí ani časová zložitosť.

4.4 Záver

Prvé pokusy ukázali, že MSOM nie je vhodná na korekciu textu a to najmä z dvoch hlavných dôvodov:

1. je príliš pomalá
2. nedokáže sa naučiť jazyk dostatočne dobre

Tieto problémy sa pokúsim v nasledujúcich kapitolách riešiť aspoň čiastočne.

5 Zlepšenie výsledkov siete

Pre lepšiu názornosť sa pozrime na reťazec, s ktorým som pracoval v predchádzajúcej kapitole, ako aj na jeho mutácie s vyššou energiou.

Pôvodný reťazec a jeho energia:
it was on the corner of the street that he noticed the first sign of something peculiar: 0.8946361

Mutácie s vyššou energiou:
it was on the corner of the street that he noticed the first fign of something peculiar: 0.8949074
it was on the corner of the street that he notied the first sign of something peculiar: 0.8950534
it was on the corner of the street that he notice the first sign of something peculiar: 0.8962081
it was on the corner of the street that he notived the first sign of something peculiar: 0.8964384
it was on the corner of the street that he noticed the first tign of something peculiar: 0.8966991
it was on the corner of the street that he noticmed the first sign of something peculiar: 0.8984091
it was on the corner of the street that he noticked the first sign of something peculiar: 0.9019380
it was on the corner of the street that he noticer the first sign of something peculiar: 0.9026988
it was on the corner of the street that he notimed the first sign of something peculiar: 0.9040620
it was on the corned of the street that he noticed the first sign of something peculiar: 0.9041193
it was on the corner of the street that the noticed the first sign of something peculiar: 0.9188260

Ako vidieť, problém je najmä v slove *noticed*. Až 7 z týchto reťazcov je zmutovaných práve v tomto slove. Ďalšie dve mutácie sú na prvom písmene slova *sign*. Skúšal som, či sa tieto problémy nedajú vyriešiť tým, že do siete pridám nový neurón, ktorému by som nastavil váhy tak, aby vyhral pri týchto problematických slovách a zvýšil energiu správneho reťazca a zároveň znížil energiu nesprávnej mutácie.

Jedno z problematických miest je napríklad *sign*. Mutácie, kde je *s* zamenené za *f* alebo *t*, majú vyššiu energiu, čo ale nechceme. Preto som skúsil pridať do pravej siete neurón, ktorý mal váhy nastavené tak, že jeho zložka zodpovedajúca písmenu *i* bola 1 a zvyšné zložky boli 0. Jeho kontextové váhy som nastavil rovné kontext deskriptoru, ktorý bol v sieti v momente, keď mal prísť symbol *i* na vstup. Vo výsledku tento neurón dával výstup 0 pri posunutí *s* na vstup a teda tento neurón vyhral. Zároveň som povedal tomuto neurónu, že predikuje symbol *s*. Tým sa energia pôvodného reťazca zvýšila. Nová energia pôvodného reťazca bola 0.9052094. Energia reťazca so zmutovaním na *fign* bola 0.8927928

a reťazca so zmutovaním na *tign* bola 0.8966991. Teda sme dosiahli to, že pôvodný reťazec má vyššiu energiu.

Črtá sa tu možnosť pridať do siete malý počet neurónov, ktoré by zvyšovali energiu správnych reťazcov tak, aby potom ich mutácie mali nižšiu energiu. Takýmto neurónom by sa nastavili váhy podobne ako som to uviedol v predchádzajúcom príklade, aby vyhrali na doteraz problematických miestach v správnom reťazci. Otázkou zostáva, či neurón pridať do pravej alebo ľavej siete alebo hneď do obidvoch. Do ktorej siete je najlepšie ho pridať možno zistiť pokusom tak, že neurón pridáme do oboch a vypočítame energiu pôvodného reťazca a problematickej mutácie. Podľa výsledkov vyberieme, v ktorej sieti nový neurón necháme. Väčšinou bude stačiť pridať neurón len do jednej siete. Z problematických mutácií uvedených vyššie bolo treba len na jednom mieste pridať neurón do oboch sietí a to pri poslednej mutácii, ktorá mala priveľkú energiu na to, aby sa dala zvládnuť pridaním neurónu len do jednej siete. Pridaním 5 neurónov do ľavej siete a dvoch neurónov do pravej siete sa mi podarilo dosiahnuť, že vyššie uvedený reťazec mal vyššiu energiu ako všetky jeho mutácie so zmutovaným jedným symbolom.

Teda po fáze učenia pomocou samoorganizácie by mohla nasledovať fáza učenia s učiteľom, kde by sa pridávali neuróny, tak ako je opísané vyššie. V momente, keď som robil vyššie uvedený pokus, nedokázal som určiť koľko neurónov by bolo treba pridať do siete, aby mali napríklad reťazce z trénovacej množiny vyššiu energiu ako ich mutácie s jedným zmutovaným symbolom. Nebolo to možné kvôli vysokej časovej náročnosti výpočtu energie. Po využití metód na zrýchlenie siete, uvedených v nasledujúcej kapitole, som zistil, že tento počet by musel byť priveľký. V trénovacom texte o dĺžke 102 804 symbolov bolo až 103 097 mutácií s vyššou energiou. Takýto vysoký počet som nečakal vychádzajúc z počtu problematických mutácií vo vyššie uvedenom reťazci. Vzhľadom na tento vysoký počet som usúdil, že pridávanie neurónov nie je schodnou cestou, pretože by ich bolo treba pridať príliš veľa a celý výpočet by sa opäť veľmi spomalil. Možno v budúcnosti po zvýšení výpočtovej sily počítačov bude pridávanie neurónov schodnejšou cestou a ukáže sa ako vhodné pre jemné dolaďovanie siete.

6 Zrýchlenie siete a korekcie

Zrýchlenie siete je nevyhnutné, pretože v pôvodnej podobe nie je schopná výpočtu v rozumnom čase. Predstavím dva spôsoby ako zrýchliť priamo výpočet siete, ktoré sa dajú použiť všeobecne pri ľubovoľnej SOM a ľubovoľnom probléme riešenom pomocou SOM. Navyše jeden spôsob ako zrýchliť výpočet pre náš konkrétny problém – korekciu textu.

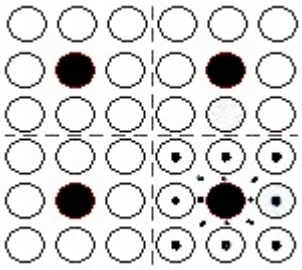
6.1 Využitie organizácie mapy

Počas trénovania sa neuróny v SOM organizujú do skupiniek. Susediace neuróny majú podobné váhy a reagujú na podobné vstupy. Preto, ak poznáme výstup jedného neurónu, vieme do istej miery predpovedať, aký bude výstup jeho topologických susedov. To vedie k myšlienke nepočítať výstup pre všetky neuróny v sieti, ale len pre niektoré. Ak je aktivita neurónu veľmi malá, tak ani jeho susedia nebudú mať dostatočne vysokú aktivitu na to, aby vyhrali. Preto pre jeho susedov nie je nutné počítať výstup.

Za predpokladu, že susedné neuróny majú podobné váhy, by stačilo počítať výstup len pre každý tretí neurón v riadku a v stĺpci a teda pre každý deviaty neurón v sieti. Ak by sme ukázali, že váhy sú podobné aj pre neuróny vo vzdialenosti 2, stačilo by počítať výstup dokonca len pre každý 25-ty neurón v sieti. A podobne, ak by sme ukázali, že vzdialenosť, pri ktorej sú váhy neurónov ešte podobné, je k, stačilo by počítať výstup len pre jeden z $(2 \cdot k + 1)^2$ neurónov.

Výstupy pre zvyšné neuróny sa budú počítať, len ak sú vo vzdialenosti najviac k od niektorého z neurónov, ktorý mal nízku aktivitu. Celkovo sa teda výpočet siete zrýchli o niečo menej ako $(2 \cdot k + 1)^2$ krát.

Skúmal som sieť MSOM veľkosti 100x100 natrénovanú na anglickom texte, aby som zistil v akej vzdialenosti sú výstupy neurónov ešte podobné. Moje pokusy ukázali, že táto vzdialenosť dosahuje miestami hodnotu 2, ale vo všeobecnosti je to len 1. Teda zrýchlenie, ktoré je možné dosiahnuť pre naše potreby, je asi 9-násobné. Je to určité zlepšenie, avšak určite nie dostatočné pre naše potreby. Oprava textu by aj s využitím tejto metódy bola stále neúnosne časovo náročná.



Obr. 3 Využitie organizácie mapy: Výstup sa počíta len pre neuróny vo vzdialenosti 3 (čierne) Pre neuróny s vysokou aktivitou sa potom vypočíta výstup aj pre susedné neuróny (s bodkou)

6.2 Odstránenie neurónov bez šance na výhru

Pri procese samoorganizácie vznikajú v sieti oblasti neurónov reagujúce na podobné vstupy. Neuróny, ktoré sa nachádzajú rovnako ďaleko od dvoch takýchto oblastí, sa potom snažia prispôsobiť obom oblastiam. Ich váhy vyjadrujú akýsi kompromis medzi váhami v jednej oblasti a váhami v druhej oblasti. Takéto neuróny teda uviaznu kdesi medzi a nie sú schopné výhry. Nemá zmysel počítať výstup pre neurón, o ktorom už dopredu vieme, že nevyhrá. Preto je rozumné takéto neuróny po natrénovaní vyhodiť zo siete.

Čím je v sieti väčší počet neurónov, tým je percento takýchto neurónov vyššie. V sieti s 1600 neurónmi bolo neaktívnych 50% neurónov. V sieti s 10000 neurónmi bolo neaktívnych 79% neurónov. Teda ak zo siete s 10000 neurónmi vyhodíme po natrénovaní neaktívne, zrýchli sa výpočet asi 5-násobne. Existuje určité riziko, že vyhodíme i neuróny, ktoré by boli aktívne pre nejaký reťazec, ktorý sa v našej testovacej množine nenachádzal, ale ak bola trénovacia množina dostatočne veľká a reprezentatívna, je toto riziko akceptovateľné pre zrýchlenie, ktoré dosiahnem.

6.3 Hashovanie neurónov

Už sme spomenuli, že susedné neuróny v sieti majú podobné váhy. Ak by sme si zaindexovali pre niektoré vstupy pozície, kde sa nachádzajú neuróny, ktoré reagujú na tieto vstupy najlepšie, mohli by sme hľadanie urýchliť tým, že by sme sa rovno pozreli na tieto

neuróny. Teda nemuseli by sme počítať výstup pre neuróny, ktorých váhy sa výrazne odlišujú od daného vstupu a teda nemajú šancu vyhrať.

Najvhodnejšie je určiť si funkciu, ktorá nám roztriedi neuróny podľa váh do niekoľkých skupín. Použitím rovnakej funkcie potom zistíme, do ktorej skupiny patrí daný vstup. Víťazný neurón potom budeme hľadať v tejto skupinke neurónov. Celkový výpočet siete sa nám tak môže zrýchliť až *n* násobne pri rozdelení do *n* skupín. Okrem počtu skupín bude celkové zrýchlenie závisieť aj od rovnomernosti rozdelenia neurónov do týchto skupín.

Táto metóda je veľmi podobná hashovaniu, ktoré mi aj bolo inšpiráciou, preto budem v ďalšom texte hovoriť o hashovaní neurónov alebo hashovanej sieti. Napriek veľkej podobnosti sú tu určité rozdiely v porovnaní s hashovaním.

Podobne ako pri hashovaní potrebujeme hashovaciu funkciu. Pričom táto funkcia musí mať nasledujúce vlastnosti:

1. Musí byť vypočítateľná v krátkom konštantnom čase.
2. Pre rovnaký vstup musí vrátiť vždy rovnakú hodnotu (t.j. nesmie mať náhodný charakter).

Na rozdiel od klasickej hashovacej funkcii musí navyše platiť:

3. Musí rozdeľovať neuróny do skupín čo najviac rovnomerne, aby sa nestalo, že v jednej skupine je veľa neurónov a v ostatných málo, čo by znamenalo, že by sme nedosiahli veľké zrýchlenie.
4. Neuróny, ktoré majú hashovacou funkciou pridelenú rovnakú hodnotu (sú v rovnakej skupine), majú pre rovnaký vstup podobné výstupy.
5. Neuróny, ktoré majú hashovacou funkciou pridelenú rozdielnu hodnotu (sú v rozdielnych skupinách), dávajú pre rovnaký vstup výrazne rozdielne výstupy

Oproti klasickému hashovaniu nie je potrebné uvažovať o riešení kolízií. Jednoducho sú neuróny s rovnakou hodnotou hashovacej funkcie v jednej skupine. Je len potrebné v zmysle bodu 3 dodržať, aby bol počet kolízií približne rovnaký pre každú hodnotu funkcie. Teda, aby počet neurónov v jednotlivých skupinách bol približne rovnaký.

Dôležité je i dodržanie bodu 4. Ak by mali neuróny v jednej skupine príliš rozdielne výstupy pre rovnaký vstup znamenalo by to, že pri prehľadávaní danej skupinky neurónov musím vypočítať výstup aj pre neuróny, ktoré nemajú žiadnu šancu vyhrať. V takom

prípade by stálo za úvahu rozdelenie tejto skupiny na dve (alebo viac) tak, aby bol splnený bod 4, čím by sa urýchlil výpočet.

Ak nie je dodržaný bod 5, znamená to, že máme v rozdielnych skupinách neuróny, ktoré dávajú podobné výstupy pre rovnaký vstup. To by mohlo znamenať, že hodnota hashovacej funkcie by daný vstup zaradila do inej skupiny ako tej, v ktorej by sa nachádzal víťazný neurón. Potom by nemuselo byť jednoduché nájsť víťaza pre daný neurón. Neskôr si ukážeme, že túto podmienku možno za určitých okolností porušiť a pritom stále dosiahnuť zlepšenie v rýchlosti výpočtu.

Neuróny v našej sieti môžme rozdeliť podľa váhových vektorov a prípadne aj podľa kontextového vektora. Skúmanie siete MSOM 100x100 neurónov natrénovanej na anglickom texte ma priviedlo na myšlienku rozdeliť neuróny podľa indexu najvyššieho prvku vo váhovom vektore. Takéto rozdelenie de facto zodpovedá rozdeleniu neurónov do skupín podľa symbolu, na ktorý reagujú. Pomocou tohto kritéria môžme zaviesť hashovaciu funkciu:

$$h_1(c, w) = \arg\max(w_i),$$

kde c je kontextový vektor a w je váhový vektor neurónu, prípadne vstupu.

Takéto rozdelenie spĺňa podmienky 1 a 2. Podmienka 3 je splnená tiež v dostatočnej miere. Je síce zrejmé, že v niektorých skupinách bude viac neurónov ako v iných, čo je spôsobené tým, že niektoré symboly sa v jazyku vyskytujú častejšie ako iné. Napríklad symbol *x* sa vyskytuje menej často a preto skupina grupujúca neuróny reagujúce na tento symbol bude menšia ako skupina neurónov reagujúcich na symbol *e*.

Podmienka 4 je tiež splnená, keďže najvyššia zložka váhového vektora najvýraznejšie vplýva na výstup. Môžu sa tu vyskytnúť isté problémy, ktoré prediskutujeme v ďalšom odseku, keďže body 4 a 5 úzko súvisia.

Skúmajme splnenie podmienky 5. Keďže na vstup vždy dávam vektor, ktorého jedna zložka je 1 a zvyšné zložky sú nulové, väčšina neurónov má jednu zložku výrazne vyššiu (blížiacu sa k 1) ako ostatné (blížiace sa k 0). V sieti však možno nájsť i neuróny, ktorých najvyššia a druhá najvyššia zložka váhového vektora sú približne rovnaké. Preto sa ponúka otázka, či sa nemôže stať, že pre vstup, ktorého hodnota hashovacej funkcie nás odkáže na jednu skupinku neurónov, sa víťazný neurón nachádza v inej skupine. Najväčšia šanca, že niečo takéto nastane je v prípade, keď sú hodnoty najvyššej a druhej najvyššej váhovej zložky neurónu približne rovnaké. Vtedy je ich hodnota približne 0.45. To znamená, že hodnota výstupu tohto neurónu bude určite viac ako 0.3025. Výstupy víťazných neurónov

však nedosahujú takéto vysoké čísla a teda ani nemôže takýto neurón zvíťaziť. Takýto neurón by mal šancu zvíťaziť jedine v sieti s malým počtom neurónov alebo v nedostatočne natrénovanej sieti.

Pomocou funkcie h_1 rozdelíme neuróny v sieti, ktorej vstup má rozmer *n*, na *n* skupín. Pri sieti trénovanej na anglickom jazyku to znamená 27 skupín a teda potenciálne až 27-násobné zrýchlenie. Toto zrýchlenie je síce slušné, ale skúsme sa pozrieť, či nie je možné zadefinovať hashovaciu funkciu, ktorá by dosahovala väčšie množstvo hodnôt, čím by sme dosiahli vyššie zrýchlenie.

Ďalšia možnosť ako rozdeliť neuróny v MSOM je podľa vektoru s kontextovými váhami. Opäť sa najlogickejšie javí možnosť rozdeliť neuróny podľa indexu najvyššej zložky kontextového vektora. Tým dosiahneme rozdelenie na ďalších 27 skupín a v kombinácii s predchádzajúcim rozdelením až 729 skupín.

$$h_2(c, w) = 27 \cdot \arg\max(w_i) + \arg\max(c_i)$$

Opäť je zrejmé, že podmienky 1 a 2 sú splnené. Podmienka 3 je podobne ako v predošlom splnená dostatočne. Takéto rozdelenie de facto zodpovedá rozdeleniu podľa posledných dvoch symbolov v reťazci, ktorý prišiel doteraz na vstup. Prípadné vyššie počty v niektorej zo skupiniek sú výsledkom vyššieho výskytu niektorých dvojíc symbolov (napríklad *pe*) oproti iným (*xx*). Podmienka 4 je tiež splnená, keďže najvyššia zložka váhového vektora a najvyššia zložka kontextového vektora najvýraznejšie vplývajú na výstup.

Problémom zostáva podmienka 5. Ako ukážem, môže sa stať, že víťazný neurón nemá rovnakú hodnotu funkcie h_2 ako vstup. Ale ukážem tiež, že takáto situácia nenastane veľmi často a ako spoľahlivo zistiť, kedy sa môže víťazný neurón nachádzať v inej skupine a ktorá skupina to je tak, aby sme stále dosiahli výrazné zrýchlenie.

Už vyššie sme si rozobrali problémy týkajúce sa najvyššej zložky váhového vektora, preto sa teraz sústredím len na problémy súvisiace s najvyššou zložkou v kontextovom vektore. V ideálnom prípade, ak by v sieti mal vždy víťazný neurón minimálny výstup, t.j. $d_i = 0$, bol by kontextový vektor každého takéhoto víťazného neurónu daný nejakou (pre každý neurón inou) funkciou $F : N \to \{1..n\}$, kde *N* je množina prirodzených čísel a *n* je rozmer vstupu (a teda aj rozmer kontextového vektora). Zložka kontextového vektora $c = (c_1..c_n)$ by vyzerala nasledovne:

$$c_i = \sum_j (1-\beta)\cdot\beta^{j-1}\cdot(F(j)=i),$$[1]

kde β je parameter neurónovej siete tak ako bolo uvedené vyššie.

Toto tvrdenie vychádza zo vzorcov na výpočet kontext deskriptora a kontextových váh.

$$cd(t) = (1-\beta)\cdot w_{i^*(t-1)} + \beta\cdot c_{i^*(t-1)}$$

$$\Delta c = \gamma_2 \cdot h(i^*, i)\cdot(cd(t) - c)$$

Ako sme už povedali, váhový vektor neurónu má práve jednu zložku blížiacu sa k 1 a zvyšné blížiace sa k 0. V ideálnom prípade má váhový vektor neurónu práve jednu zložku rovnú 1 a zvyšné rovné 0. Nech je kontextový váhový vektor víťazného neurónu z predchádzajúceho kroku daný funkciou F_{t-1} a *k*-ta zložka jeho váhového vektora je rovná 1. Potom kontext deskriptor bude daný ako:

$$cd(t)_i = \sum_j (1-\beta)\cdot(j=k) + (1-\beta)\cdot\beta^j\cdot(F_{t-1}(j)=i).$$

Zo vzorca na úpravu kontextových váh vyplýva, že *c* = *cd*. Teda môžeme definovať funkciu $F_t : N \to \{1..n\}$:

$$F_t(1) = k$$

$$F_t(j) = F_{t-1}(j-1)$$

Kontextový vektor víťazného neurónu v nasledujúcom kroku by bol teda opäť daný funkciou F_t.

Zobrazenie F ukazuje kontext, pri ktorom neurón víťazí. Ak Σ je abeceda symbolov dávaných na vstup a $\omega : \Sigma \to \{1..n\}$ je očíslovanie týchto symbolov, ktoré sme použili pri trénovaní siete a posledné symboly dané na vstup siete boli $s_1, s_2, s_3 ...$, potom by v ideálnom prípade zvíťazil neurón, ktorého kontextový vektor je daný funkciou $F(i) = \omega(s_i)$. Teda takýto ideálny prípad by zrejme mohol nastať iba pre nekonečne veľkú sieť. Napriek tomu i pre siete s konečnou veľkosťou môžeme pozorovať, že väčšina víťazných neurónov sa snaží mať svoj kontextový vektor viac či menej presne daný podobnou funkciou, ako by to bolo v ideálnom prípade.

Dá sa povedať, že okrem funkcie F je kontextový vektor daný i funkciou šumu $\Omega : \{1..n\} \to R$;

[1] Výraz $(F(j)=i)$ sa vyhodnotí ako 1 ak rovnosť platí a ako 0 ak rovnosť neplatí.

$$c_i = \Omega(i) + \sum_j (1-\beta) \cdot \beta^{j-1} \cdot (F(j) = i)$$

Šum je závislý najmä na tom aké sú kontextové vektory víťaziacich neurónov, ktoré sa nachádzajú v okolí neurónu. Čím bližšie pri sebe sa nachádzajú dva neuróny, ktoré víťazia, tým viac si navzájom ovplyvňujú svoje váhy. Priemerné vzdialenosti víťaziacich neurónov možno zmenšiť zväčšením siete. Aj z pokusov vidno, že čím je vyšší počet neurónov v sieti, tým je šum pre víťazné neuróny menší. Hodnoty šumu sa pohybujú okolo 0 a pre ideálny prípad sú práve 0.

Pri trénovaní siete som použil hodnotu $\beta = 0.5$. V ideálnom prípade by teda zložka $c_{F(1)}$ bola najvyššia spomedzi všetkých zložiek kontextového vektora víťazného neurónu. Pričom väčšinou by bola táto zložka okolo 0.5 Druhá najvyššia zložka by zase bola $c_{F(2)}$ a jej hodnota by bola väčšinou okolo 0.25. Pri pokusoch so sieťou s 10 000 neurónmi tieto skutočnosti platili pre drvivú väčšinu víťazných neurónov.

Teda vzdialenosť medzi najvyššou a druhou najvyššou zložkou kontextového vektora víťaziaceho neurónu je väčšinou asi 0.25. Už sme hovorili o tom, že najvyššia zložka váhového vektora je približne 1. Zo vzorca na výpočet kontext deskriptora vidíme, že kontext deskriptor bude mať vždy jednu zložku okolo 0.5, nech je to *i*-ta zložka, a väčšinou jednu zložku okolo 0.25. Ak zoberieme neurón, ktorého *i*-ta zložka kontextového vektora nie je najvyššia, bude výstup tohto neurónu väčšinou väčší ako $(0.5\text{-}0.25)^2 = 0.0625$. Počet neurónov, ktoré majú výstup vyšší ako 0.0625, bol pri mojich pokusoch veľmi malý.

Pozrime sa na praktické výsledky neurónovej siete s 10 000 neurónmi. Len 0.46% víťazných neurónov malo pri pokusoch výstup vyšší ako 0.0625. Už z toho vidno, že je málo pravdepodobné, aby vyhral neurón, ktorého index najvyššej zložky kontextového vektora nie je zhodný s indexom najvyššej zložky kontext deskriptora. Podiel takýchto víťazných neurónov je 1.05%. Teda v 1.05% prípadoch sa stane, že hodnota funkcie h_2 pre vstup nie je zhodná s hodnotu funkcie h_2 pre víťazný neurón. V takomto množstve prípadov by sme pri využití hashovania nenašli vždy správny víťazný neurón, ak by sme ho hľadali len v skupine neurónov s rovnakou hodnotou funkcie h_2 ako má vstup.

I tomuto malému percentu prípadov, kedy by sme nenašli správny víťazný neurón, je možné zabrániť. Najskôr nájdeme v skupinke neurónov s rovnakou hodnotou funkcie h_2 ten s najnižším výstupom. Nech jeho výstup je $d_{\min}$. Ak je daná skupina prázdna (nie je v nej žiaden neurón), uvažujme ďalej $d_{\min} = \infty$.

Nech cd_i je najvyššia zložka kontext deskriptora a nech cd_j je druhá najvyššia zložka kontext deskriptora. Skúmajme, aký môže byť najnižší výstup neurónu, ktorý nemá rovnakú hodnotu funkcie h_2 ako kontext deskriptor. Zrejme je to v prípade, keď sú prvá a druhá najvyššia zložka kontextového vektora tohto neurónu vymenené. Teda hľadáme minimum funkcie $f(c_i, c_j) = (c_i - cd_i)^2 + (c_j - cd_j)^2$ *za podmienky* $cd_i \geq cd_j \wedge c_i \leq c_j$. Táto funkcia dosahuje minimum pre $c_i = c_j = \frac{cd_i + cd_j}{2}$.

Ak teda $d_{\min} > f\left(\frac{cd_i + cd_j}{2}, \frac{cd_i + cd_j}{2}\right) = 2 \cdot \left(\frac{cd_i - cd_j}{2}\right)^2$ je možné, že neurón, ktorému tento výstup náleží, nie je víťazný. Ak sa víťazný neurón nenachádza v skupinke neurónov s rovnakou hodnotou funkcie h_2, je najpravdepodobnejšie, že bude práve v skupine, kde sú najvyššia a druhá najvyššia zložka kontextového vektora vymenené. V takomto prípade je potrebné vypočítať hodnoty výstupov aj pre neuróny v tejto skupine.

Pokusy ukazujú, že výstup víťazného neurónu spĺňa vyššie uvedenú nerovnosť v 2.01% prípadoch. V asi polovici z týchto prípadov sa víťazný neurón naozaj nachádza v druhej skupine. Napriek tomu, že existuje istá pravdepodobnosť, že sa víťazný neurón nenachádza ani v druhej skupine, pri mojich pokusoch takýto prípad nastal, len ak sa v prvej ani v druhej skupine nenachádzal žiaden neurón. Takýchto prípadov bolo 0.15% a vtedy som prezrel všetky skupiny s rovnakou hodnotu funkcie h_1.

Zrýchlenie výpočtu siete pomocou hashovania neurónov závisí od počtu rôznych hodnôt hashovacej funkcie (počtu skupín neurónov), ako aj od rovnomernosti rozdelenia neurónov do týchto skupín. Teoreticky je maximálne zrýchlenie siete pomocou funkcie h_2 729-násobné. Pri mojich pokusoch sa mi podarilo dosiahnuť 194-násobné zrýchlenie, čo je spôsobené nerovnomerným rozdelením neurónov.

6.4 Zrýchlenie korekcie pomocou nezávislých mutácií

Zrýchlenia popísané vyššie je možné použiť na SOM prípadne MSOM riešiacu ľubovoľný problém. Zrýchlenie prezentované v tomto odseku sa týka priamo problému korekcie textu a nie je mienené ako zrýchlenie MSOM riešiacej iný problém.

Pri počítaní energie reťazca závisí čas počítania od dĺžky tohto reťazca. Pričom táto závislosť nie je lineárna ako sme si už vyššie povedali. Ukazuje sa teda, že bude veľkým problémom opravovať dlhé reťazce. Kočalka v [2] mal rovnaký problém a rozhodol sa ho riešiť tým, že reťazec rozdelí na menšie úseky pevnej dĺžky a potom opravuje jednotlivé úseky zvlášť. Tento prístup nie je zlý, ale dochádza pri ňom k veľa nepresnostiam. Pokúsim sa preto ukázať iný efektívnejší pohľad na náš problém.

Pri počítaní energie zmutovaného reťazca nie je potrebné vypočítať čiastkovú energiu pre každý symbol v reťazci. Práve toto hovorí definícia maximálnej hĺbky kontextu. Nemá zmysel počítať čiastkovú energiu pre symboly vzdialené od miesta mutácie viac ako je maximálna hĺbka kontextu. Tieto čiastkové energie budú mať rovnakú hodnotu. Pri počítaní energie mutácie teda vypočítame len čiastkovú energiu symbolov vzdialených od miesta mutácie nanajvýš o maximálnu hĺbku kontextu. Pri zvyšných symboloch sa aktivujú opäť rovnaké neuróny ako to bolo pri pôvodnom reťazci. Stačí si preto zapamätať čiastkové energie pre každý symbol pri výpočte energie pôvodného reťazca a ich sumu.

Definicia 6.1: Nech $s = a_1, a_2 ... a_n$ je pôvodný reťazec. Nech $p_1 ... p_n$ sú pravé čiastkové energie a nech $l_1 ... l_n$ sú ľavé čiastkové energie reťazca *s*, pričom p_i je pravá čiastková energia *i*-teho symbolu reťazca *s* a l_i je ľavá čiastková energia *i*-teho symbolu reťazca *s*. Nech *k* je maximálna hĺbka kontextu siete.

a) Nech reťazec $s' = a_1, a_2 ... a_{i-1} b a_{i+1} ... a_n$ vznikol z reťazca *s* zamenením *i*-teho symbolu reťazca za symbol *b*. Nech $p'_1 ... p'_n$ sú pravé čiastkové energie a nech $l'_1 ... l'_n$ sú ľavé čiastkové energie reťazca s', pričom p'_i je pravá čiastková energia *i*-teho symbolu reťazca s' a l'_i je ľavá čiastková energia *i*-teho symbolu reťazca s'. Potom sumačný rozdiel energií reťazcov *s* a s' je definovaný nasledovne:

$$diff(s, s') = \sum_{j=i-1}^{i+k} (l'_j - l_j) + \sum_{j=i+1}^{i-k} (p'_j - p_j)$$

b) Nech reťazec $s' = a_1, a_2 ... a_i b a_{i+1} ... a_n$ vznikol z reťazca *s* vsunutím symbolu *b* za *i*-ty symbol. Nech $p'_1 ... p'_{n+1}$ sú pravé čiastkové energie a nech $l'_1 ... l'_{n+1}$ sú ľavé čiastkové energie reťazca s', pričom p'_i je pravá čiastková energia *i*-teho

symbolu reťazca s' a l'_i je ľavá čiastková energia i-teho symbolu reťazca s'. Potom sumačný rozdiel energií reťazcov s a s' je definovaný nasledovne:

$$diff(s,s') = \sum_{j=i}^{i+k}(l'_j - l_j) + l'_{i+k+1} + \sum_{j=i+1}^{i-k-1}(p'_j - p_j) + p'_{i+2}$$

c) Nech reťazec $s' = a_1, a_2 ... a_{i-1} a_{i+1} ... a_n$ vznikol z reťazca s vymazaním i-teho symbolu. Nech $p'_1 ... p'_{n-1}$ sú pravé čiastkové energie a nech $l'_1 ... l'_{n-1}$ sú ľavé čiastkové energie reťazca s', pričom p'_i je pravá čiastková energia i-teho symbolu reťazca s' a l'_i je ľavá čiastková energia i-teho symbolu reťazca s'. Potom sumačný rozdiel energií reťazcov s a s' je definovaný nasledovne:

$$diff(s,s') = \sum_{j=i-1}^{i+k-1}(l'_j - l_j) - l_{i+k} + \sum_{j=i+1}^{i-k}(p'_j - p_j) - p_{i+1}$$

Veta 6.1: Nech $s = a_1, a_2 ... a_n$ je pôvodný reťazec. Nech $e_{sum}(s)$ je suma čiastkových energií reťazca. Nech k je maximálna hĺbka kontextu siete. Nech reťazec s' vznikol z reťazca s zmutovaním jedného symbolu. Potom energiu reťazca možno vypočítať nasledovne:

$$e(s') = \frac{e_{sum}(s) + diff(s,s')}{|s'|}, \text{ kde } |s'| \text{ je dĺžka reťazca } s'$$

Platnosť predchádzajúcej vety vyplýva priamo z definície maximálnej hĺbky kontextu, ako aj z diskusie pred ňou.

Keď sme bez využitia maximálnej hĺbky kontextu chceli vypočítať energiu mutácie reťazca s jedným zmutovaným symbolom, časová zložitosť pre tento výpočet bola $O(n)$. Ako vidno z predchádzajúcej vety, ak sme si už vypočítali energiu pôvodného reťazca, s využitím maximálnej hĺbky kontextu môžeme energiu zmutovaného reťazca vypočítať v čase $O(k)$, kde k je malá konštanta.

Bez využitia maximálnej hĺbky kontextu sme vedeli vypočítať energiu všetkých mutácií reťazca s jedným zmutovaným symbolom v čase $O(n^2)$ (n spôsobmi vyberieme symbol, ktorý budeme mutovať a vypočítame čiastkovú energiu pre každý symbol v reťazci). Spôsobom popísaným vyššie sa táto zložitosť zníži na $O(k \cdot n) = O(n)$ (vyberieme

n spôsobmi symbol, ktorý budeme mutovať a vypočítame čiastkové energie pre symboly vzdialené od neho najviac o hĺbku maximálneho kontextu). Teda zložitosť je len lineárna.

Ak sme predtým chceli počítať energiu všetkých mutácií reťazca s dvoma zmutovanými symbolmi, časová zložitosť pre tento výpočet bola $O(n^3)$. Ak by mal byť počet mutácií tri, bolo by to až $O(n^4)$. Využitím maximálnej hĺbky kontextu môžeme tieto zložitosti znížiť na $O(n^2)$ resp. $O(n^3)$.

Definicia 6.2: Nech je daná neurónová sieť MSOM a nech *k* je maximálna hĺbka kontextu tejto siete. Potom mutácie v reťazci vzdialené aspoň *k* symbolov nazývame nezávislé mutácie.

Pojem nezávislých mutácií je veľmi dôležitý. Umožní nám urýchliť výpočet energií mutácií reťazca, ak chceme naraz mutovať dva alebo viac symbolov.

Pozrime sa najskôr, ako to je, keď sú mutácie nezávislé. Opäť si vypočítajme čiastkové energie pôvodného reťazca a ich sumu. Potom si vypočítame sumačné rozdiely energií medzi pôvodným reťazcom a všetkými jeho mutáciami s jedným zmutovaným symbolom. To, ako sme si už ukázali, vieme v čase $O(n)$. Nech chceme v reťazci zmutovať symboly na *i*-tom a *j*-tom mieste. Ak platí $(i < j) \wedge (j - i > k)$ (teda mutácie sú nezávislé), nie je potrebné počítať energiu pomocou hľadania víťazov v sieti. Stačí využiť už vypočítané sumačné rozdiely energie pre zmutované reťazce s jednou mutáciou na *i*-tom alebo *j*-tom mieste.

Veta 6.2: Nech $s = a_1, a_2 ... a_n$ je pôvodný reťazec. Nech $p_1 ... p_n$ sú pravé čiastkové energie a nech $l_1 ... l_n$ sú ľavé čiastkové energie reťazca *s*. Pričom p_i je pravá čiastková energia *i*-teho symbolu reťazca *s* a l_i je ľavá čiastková energia *i*-teho symbolu reťazca *s*. Nech *k* je maximálna hĺbka kontextu siete. Nech reťazec $s' = b_1 ... b_m$ vznikol z reťazca *s* mutáciou na *h* miestach. Očíslujme si tieto mutácie 1..*h*. Nech všetky mutácie v reťazci s' sú nezávislé. Nech s'_j je reťazec s jedinou mutáciou oproti pôvodnému reťazcu a to tou, ktorá má číslo *j* v reťazci s'. Potom energiu reťazca s' vypočítame nasledovne:

$$e(s') = \sum_{j=1}^{h} diff(s, s'_j) + \sum_{j=1}^{n} \left(l_j \cdot f_l(mov(j)) + p_j \cdot f_p(mov(j)) \right), \text{ kde}$$

$f_p : \{1...m\} \to \{0,1\}$ je funkcia definovaná nasledovne:

$$f_p(j) = \begin{matrix} 0 \ ak \ \ j \in \bigcup_{i \ je \ miesto \ mutácie} \langle i-k, i+1 \rangle \\ 1 \ ak \ \ j \notin \bigcup_{i \ je \ miesto \ mutácie} \langle i-k, i+1 \rangle \end{matrix}$$

$f_l : \{1...m\} \to \{0,1\}$ je funkcia definovaná nasledovne:

$$f_l(j) = \begin{matrix} 0 \ ak \ \ j \in \bigcup_{i \ je \ miesto \ mutácie} \langle i-1, i+k \rangle \\ 1 \ ak \ \ j \notin \bigcup_{i \ je \ miesto \ mutácie} \langle i-1, i+k \rangle \end{matrix}$$

funkcia $mov : \{1...n\} \to \{1...n\}$ je funkciou posunu indexu symbolu

$mov(j) = j +$ (*počet vsunutí znaku pred j-ty znak v* s'

- (*počet vymazaní znaku pred j-tym znakom v* s')

Trochu zjednodušene povedané funkcia *mov* určuje, kde sa symbol s daným indexom v pôvodnom reťazci nachádza v zmutovanom reťazci. Zjednodušene povedané preto, že pri miestach mutácií to funguje trochu ináč, ale to je pre nás nepodstatné. Funkcia *f* aj tak vráti požadovaný výsledok.

Funkcia *f* zase určuje, či daný symbol je ovplyvnený mutáciou. Teda či sa nachádza vo vzdialenosti maximálnej hĺbky kontextu od miesta mutácie. Ak áno, jeho čiastková energia sa mutáciou zmenila a je už zaradený vo výpočte v prvej sume a preto funkcia *f* vráti nulu. Inak je potrebné započítať jeho čiastkovú energiu z pôvodného reťazca.

Časová zložitosť výpočtu energie spôsobom popísaným vyššie je síce $O(n)$, takže sa môže na prvý pohľad zdať, že sme zrýchlenie nedosiahli, ale treba si uvedomiť, že už máme predpočítané všetky potrebné hodnoty. Teda nemusíme vykonávať časovo najnáročnejšiu operáciu - hľadanie víťaza v sieti. Potrebujeme spraviť len $O(n)$ operácií sčítania, ktoré sú neporovnateľne rýchlejšie oproti hľadaniu víťaza v sieti. Tým sa výpočet urýchli niekoľkonásobne.

Ak mutácie nie sú nezávislé, teda ak zmutujeme symboly na *i*-tom a *j*-tom mieste a platí $0 < j - i = d \le k$, je situácia trochu komplikovanejšia. Opäť môžeme využiť už vypočítané čiastkové energie pre niektoré symboly, ale mutácie sa navzájom ovplyvňujú, takže treba niektoré energie prepočítať nanovo.

Definicia 6.3: Nech je daná neurónová sieť MSOM. Nech je daný reťazec *s* a jeho mutácia s'. Nech $m_1...m_n$ sú indexy, na ktorých je reťazec s' zmutovaný. Množinu mutácií s indexami $m_1...m_n$ nazývame závislé mutácie, ak $\forall i \in \{1...n\} \exists j \in \{1...n\} i \neq j \wedge \left|m_i - m_j\right| \leq k$.

Definicia 6.4: Nech je daná neurónová sieť MSOM. Nech je daný reťazec *s* a jeho mutácia s'. Nech $m_1...m_n$ sú indexy, na ktorých je reťazec s' zmutovaný a nech je množina mutácií s týmito indexami závislá. Nech $M_1...M_m$ sú všetky ostatné mutácie v reťazci s', t.j. $\{M_1...M_m\} \cap \{m_1...m_n\} = \emptyset \wedge \{M_1...M_m\} \cup \{m_1...m_n\} = \{\text{všetky mutácie v reťazci } s'\}$.
Množinu mutácií s indexami $m_1...m_n$ nazývame maximálnou množinou závislých mutácií, ak platí $\forall i \in \{1...n\} \forall j \in \{1...m\} \left|m_i - M_j\right| > k$. Skrátene budeme hovoriť o maximálne závislých mutáciách.

Definicia 6.5: Nech $s = a_1, a_2...a_n$ je pôvodný reťazec. Nech $p_1...p_n$ sú pravé čiastkové energie a nech $l_1...l_n$ sú ľavé čiastkové energie reťazca *s*, pričom p_i je pravá čiastková energia *i*-teho symbolu reťazca *s* a l_i je ľavá čiastková energia *i*-teho symbolu reťazca *s*. Nech *k* je maximálna hĺbka kontextu siete. Nech reťazec s' vznikol z reťazca *s* mutáciou niekoľkých symbolov a nech je množina týchto mutácií maximálne závislá. Nech *ml* je index najľavejšej mutácie a nech *mp* je index najpravejšej mutácie. Nech $p'_1...p'_n$ sú pravé čiastkové energie a nech $l'_1...l'_n$ sú ľavé čiastkové energie reťazca s'. Pričom p'_i je pravá čiastková energia *i*-teho symbolu reťazca s' a l'_i je ľavá čiastková energia *i*-teho symbolu reťazca s'. Potom energia množiny maximálne závislých mutácií je definovaná nasledovne:

$$e_{mut}(s') \sum_{j=ml-1}^{mp+k} l'_j + \sum_{j=mp+1}^{ml-k} p'_j$$

Sumačný rozdiel energií reťazcov *s* a s' je definovaný nasledovne:

$$diff(s,s') = \sum_{j=ml-1}^{mp+k} l'_j - \sum_{j=mov^{-1}(ml-1)}^{mov^{-1}(mp+k)} l_j + \sum_{j=mp+1}^{ml-k} p'_j - \sum_{j=mov^{-1}(mp+1)}^{mov^{-1}(ml-k)} p_j$$

Veta 6.3: Nech $s = a_1, a_2 ... a_n$ je pôvodný reťazec. Nech $e_{sum}(s)$ je suma čiastkových energií reťazca. Nech k je maximálna hĺbka kontextu siete. Nech reťazec s' vznikol z reťazca s zmutovaním niekoľkých symbolov a nech tieto mutácie sú závislé. Potom energiu reťazca možno vypočítať nasledovne:

$$e(s') = \frac{e_{sum}(s) + diff(s, s')}{|s'|}, \text{ kde } |s'| \text{ je dĺžka reťazca } s'$$

Platnosť predchádzajúcej vety vyplýva priamo z definície maximálnej hĺbky kontextu a závislých mutácií, ako aj z diskusie pred ňou.

Zložitosť pre výpočet energie zmutovaného reťazca s len závislými mutáciami je $O(n)$. Treba však povedať, že ak je počet týchto mutácií konečný. Tak je zložitosť len $O(k)$ teda konštantná.

Vetu 6.2. môžeme teraz rozšíriť o reťazce so závislými mutáciami.

Veta 6.4: Nech $s = a_1, a_2 ... a_n$ je pôvodný reťazec. Nech $p_1 ... p_n$ sú pravé čiastkové energie a nech $l_1 .. l_n$ sú ľavé čiastkové energie reťazca *s*. Pričom p_i je pravá čiastková energia *i*-teho symbolu reťazca *s* a l_i je ľavá čiastková energia *i*-teho symbolu reťazca *s*. Nech *k* je maximálna hĺbka kontextu siete. Nech reťazec $s' = b_1 ... b_m$ vznikol z reťazca *s* mutáciami na viacerých miestach. Nech je celkom v tomto reťazci maximálne *h* závislých množín mutácií. Očíslujme si ich 1..*h*. Nech s'_j je reťazec s tými mutáciami oproti pôvodnému reťazcu, ktoré sú v množine závislých mutácií s číslom *j*. Potom energiu reťazca s' vypočítame nasledovne:

$$e(s') = \sum_{j=1}^{h} diff(s, s'_j) + \sum_{j=1}^{n} \left(l_j \cdot f_l(mov(j)) + p_j \cdot f_p(mov(j)) \right), \text{ kde}$$

funkcie *f* a *mov* sú definované rovnako ako vo vete 6.2

Časová zložitosť výpočtu energie spôsobom popísaným vyššie je síce opäť $O(n)$, takže sa môže na prvý pohľad zdať, že sme zrýchlenie nedosiahli, ale opäť si treba uvedomiť, že už máme predpočítané všetky potrebné hodnoty. Teda nemusíme vykonávať časovo najnáročnejšiu operáciu - hľadanie víťaza v sieti. Potrebujeme spraviť len $O(n)$

operácií sčítania, ktoré sú neporovnateľne rýchlejšie oproti hľadaniu víťaza v sieti. Tým sa výpočet urýchli niekoľkonásobne.

Teda, ak sme ochotní pripustiť len konštantnú maximálnu veľkosť množiny závislých mutácií, vieme energie všetkých mutácií reťazca vypočítať v čase $O(n)$, čo sa týka hľadania víťazov v sieti. Navyše potrebujeme aj $O(n^2)$ operácií sčítania, ale ako sme si povedali, tie sú veľmi rýchle. Napriek tomu je možné i počet operácií sčítania zredukovať na $O(n)$.

Počet operácií sčítania je možné zredukovať tak, že nebudeme počítať energiu pre každú mutáciu. Každý symbol v pôvodnom reťazci má čiastkovú energiu, ktorú sme si vypočítali ešte na začiatku. Vypočítame si aj energie všetkých množín maximálne závislých mutácií. Potom z týchto energií vieme nájsť energiu mutácie s najvyššou energiou.

Veta 6.5: Nech *mut* je ľubovoľná množina maximálne závislých mutácií. Nech *g*(*mut*) je energia tejto množiny, *p*(*mut*) a *l*(*mut*) sú indexy najpravejšej a najľavejšej mutácie z tejto množiny vzhľadom k pôvodnému reťazcu. Nech *z* je ľubovoľný symbol z pôvodného reťazca. Nech $l(z) = p(z)$ je index tohto symbolu v reťazci *s*. Potom energia mutácie reťazca *s*, ktorá má najnižšiu energiu, je daná nasledovne:

$$e(s') = \max \sum_{i=1}^{m} g(mut_i),$$

kde mut_i je množina maximálne závislých mutácií alebo niektorý zo symbolov z reťazca *s* a platí $l(mut_1) = 1 \wedge \forall i \in \{1..m\} p(mut_i) = l(mut_{i+1}) - 1 \wedge p(mut_m) = m$

Samotný reťazec, ktorý je mutáciou s najvyššou energiou potom jednoducho zostavíme, ak si pri hľadaní maxima budeme pamätať, ktoré časti reťazca (symboly alebo závislé mutácie) boli pri počítaní energie použité. Teda je zrejmé, že nepotrebujeme počítať energiu každej mutácie reťazca, aby sme zistili, ktorá z nich má najvyššiu energiu. Vyššie uvedené maximum je možné vypočítať v čase $O(n^2)$. Ako sme si však povedali, je rozumné v záujme zrýchlenia pripustiť len konštantnú maximálnu veľkosť množiny závislých mutácií. V takom prípade sa dá vyššie uvedené maximum vypočítať v čase $O(n)$.

Využitie maximálnej hĺbky kontextu a nezávislých mutácií sa ukazuje ako kľúčové pre rýchle opravovanie dlhších reťazcov. Bez neho by bolo potrebné dlhý reťazec najskôr rozdeliť na malé úseky tak ako to robil Kočalka v [2] a počítať energie po takýchto malých úsekoch, čo by nebolo až také efektívne, ale hlavne vypočítané energie by boli menej presné.

V nasledujúcom je stručný výpis kódu zrýchleného algoritmu popísaného v tejto kapitole:

```
procedure mutate(muts, first, last, count)
begin
   muts := mutateChar(muts, last);
   mute[first-k, last+k] := max(mute[first-k, last+k], e(muts[first-k,last-k]));
   if (count > 0)
      for k := last+1 step 1 until last+k do
         mutateChar(muts, first, k, count-1);
end;
```

Procedúra `mutate` volá rekurzívne sama seba a mutuje postupne reťazec `muts` tak, aby mutácie, ktoré v ňom spraví, boli závislé. V reťazci zmutuje maximálne `count` symbolov. Pamätá si index prvej a poslednej mutácie, ktorú spravila. V poli `mute` si pamätá energie všetkých množín závislých mutácií, ktoré vytvorí. Konštanta `k` je maximálna hĺbka kontextu siete.

```
procedure mutationEnergies
begin
     // pre potreby počítania maxima najskôr všade priradíme nekonečno
     for i := 1 step 1 until length(s) do
        for j := i+2*k+1 step 1 until (MAX_MUT+1)*(k-1)+2 do mute[i,j] := -∞

     // vypočítame energie pre závislé mutácie
     for i := k+1 step 1 until length(s)-k-1 do mutate(s, i, i, MAX_MUT-1);
end;
```

Procedúra `mutationEnergies` vypočíta energie všetkých prípustných množín závislých mutácií a z týchto energií si zapamätá tie najvyššie v danom rozsahu. V množine závislých mutácií môže byť maximálne `MAX_MUT` mutácií.

```
procedure findMaxEnergy
begin
win[1,0] := 0;
    for i := 1 step 1 until length(s) do
        win[1,i] := win[1,i-1] + e(s[1]);
        for j := i-(MAX_MUT+1)*(k-1)+2 step 1 until i-2*k-1 do
        begin
            if (j>0) then win[1,i] := max(win[1,i], win[1,j-1]+mute[j,i]);
        end;
end;
```

Procedúra `findWinner` počíta energiu tej mutácie pôvodného reťazca, ktorá má najvyššiu energiu. Teda energiu reťazca, ktorý ma byť korektný. V podstate táto procedúra vyskladá reťazec z malých kúskov, ktoré sú buď znakmi z pôvodného reťazca alebo úsekmi pôvodného reťazca s niekoľkými zmutovanými znakmi, ktoré sú maximálne závislé. Teda robí presne to, čo hovorí veta 6.5. Ak `MAX_MUT` je maximálny počet závislých mutácií, potom množina závislých mutácií má dĺžku maximálne `(MAX_MUT+1)*(k-1)+2` a minimálne `2*k-1`. Nie je ťažké túto procedúru doplniť tak, aby si dokázala spätne vytvoriť aj samotný korektný reťazec.

6.5 Záver

Metódy uvedené vyššie nie je možné skombinovať každú s každou. Ak chceme využiť hashovanie neurónov nemá už význam využívať organizáciu mapy. Z týchto metód je najlepšie skombinovať vyhodenie neaktívnych neurónov, hashovanie neurónov a využite maximálnej hĺbky kontextu a nezávislých mutácií. Kombináciou týchto metód sa mi podarilo dosiahnuť výrazné zníženie časovej náročnosti. Korekcia textu s maximálne jedným zmutovaným symbolom v podstate prestala byť problémom. Pri mojich pokusoch trvala korekcia reťazca dĺžky 100 asi 5.5 sekundy, čo je výrazné zrýchlenie oproti pôvodným vyše 7 hodinám. Pri maximálne dvoch závislých mutáciách trvala korekcia 527 sekúnd. Pri viac povolených závislých mutáciách je už doba korekcie pridlhá a nie je mysliteľné uvažovať viac mutácií naraz.

7 Programovanie

Programoval som v jazyku Java 5.0_01 v prostredí Eclipse 3.0.2. Pri programovaní som využil objektovo orientovaný prístup. Jeden neurón je reprezentovaný objektom typu *Neuron* a neurón v sieti MSOM, ktorý má navyše aj kontextové váhy je reprezentovaný objektom typu *NeuronMSOM*, ktorý je potomkom *Neuron*. Sieť MSOM je reprezentovaná objektom typu *MSOM* a obsahuje niekoľko *NeuronMSOM*.

Trieda *HashedNet* reprezentuje hashovanú MSOM tak, ako je to popísané v kapitole 6.3. Trénovanie siete je možné pustiť zavolaním triedy *MSOMLauncher*. Spustením triedy *Correction* je možné spustiť výpočet energií mutácií. Samotným výstupom nie je len reťazec, ktorý sieť považuje za korektný, keďže ako sme si povedali, sieť neopravuje text spoľahlivo, ale sú ním všetky prípustné mutácie spolu s ich energiami. V classe *NetTrainer* som sa snažil uplatniť postup pridávania neurónov popísaný v kapitole 5, ale ako je v tejto kapitole napísané, nie veľmi úspešne.

Podrobnú dokumentáciu ako aj zdrojový kód je možné nájsť v elektronickej prílohe.

8 Záver

V práci sa mi podarilo natrénovať neurónovú sieť MSOM na korpuse anglického jazyka. Predstavil som spôsob ako opravovať text pomocou takto natrénovanej siete. Ukázal som, že korekcia textu pomocou siete MSOM je príliš časovo náročná. Navrhol som však metódy na zrýchlenie tohto výpočtu, ktoré do veľkej miery zreálnili možnosti opravy textu pomocou MSOM. Opravovanie veľkého množstva pri sebe sa nachádzajúcich chýb je aj po časovej redukcii stále veľmi náročné, ale opravovanie aj veľkého množstva chýb, ktoré sa nachádzajú ďaleko od seba, je možné uskutočniť vďaka navrhnutým metódam v rozumnom čase. Korekcia textu bez využitia navrhnutých zrýchlení siete by trvala čas $O(n^m)$, ak m by mal byť maximálny počet chýb v reťazci dĺžky n. Ako je uvedené v kapitole 6.4, ak chceme opravovať reťazce, v ktorých sa nachádza len konštantný maximálny počet závislých mutácií, môžeme túto korekciu urobiť v čase $O(n)$. Pritom celkový počet chýb môže byť aj $O(n)$, ak je maximálny počet závislých mutácií konštantný. Takáto korekcia postačuje na drvivú väčšinu bežne sa vyskytujúceho poškodeného textu. Nepodarilo sa mi však zlepšiť presnosť siete a preto je oprava pomocou MSOM veľmi nespoľahlivá. Napriek tomu verím, že ak by sa natrénovala niekoľkonásobne väčšia neurónová sieť, bola by pomocou mnou navrhnutých postupov korekcia textu možná. Trénovanie takejto veľkej siete je ale veľmi časovo náročné a mohlo by trvať aj niekoľko mesiacov.

9 Použitá literatúra

[1] P. Kapalla, Dopĺňanie diakritiky do slovenského textu s využitím neurónovej siete, Diplomová práca, FEI STU, 1997.

[2] P. Kočalka, Fraktálna reprezentácia ľavých a pravých kontextov prirodzeného jazyka, aplikácia na automatickú korekciu textov, Diplomová práca, FEI STU, 1998.

[3] T. Kohonen, Self-Organizing Maps. Springer-Verlag, Berlin, 3rd edition, 2001.

[4] T. Kohonen, P. Somervuo, Self-organizing maps of symbol strings, *Neurocomputing* 21 (1998) 19-30.

[5] M.Strickert, B.Hammer, Merge SOM for temporal data,. *Neurocomputing* 64:39-72.

[6] T. Voegtlin, Recursive self-organizing maps. *Neural Networks,* 15(8{9):979-992, 2002.

[7] W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5: 115-133, 1943.

[8] Rumelhart, McClelland, PDP, 1986.