Univerzita Komenského
Fakulta matematiky, fyziky a informatiky
Katedra aplikovanej informatiky

Spracovanie štruktúrovaných dát pomocou lineárnej RAAM

(Diplomová práca)

Michal Pokorný

Diplomový vedúci: Ing. Igor Farkaš, PhD.

Bratislava, 2006

Zadanie

Implementujte model neurónovej siete *linear RAAM* schopnej reprezentovať štruktúrované dáta. Model otestujte na vhodných dátach a analyzujte jeho dynamické vlastnosti a schopnosť zovšeobecnenia. Vyšetrite vplyv kódovania vstupov na reprezentačné vlastnosti modelu. Implementujte model tak, aby bol použiteľný aj pod OS Linux.

Čestne prehlasujem, že som túto diplomovú prácu vypracoval samostatne s použitím citovaných zdrojov, pod starostlivým vedením môjho diplomového vedúceho.

. . . . . .

Michal Pokorný

Poďakovanie

V prvom rade chcem vyjadriť vďačnosť svojmu školiteľovi Ing. Igorovi Farkašovi, PhD. za všetky cenné rady a pripomienky k mojej práci.

Ďalej sa chcem poďakovať svojím rodičom, súrodencom, mojej priateľke a kamarátom za to, že mi boli oporou (nielen) počas písania tejto práce.

Abstrakt

Spracovanie štruktúrovaných dát je pre neurónové siete ťažkou úlohou. V nedávnej dobe predstavili Voegtlin & Dominey [2] modifikáciu modelu rekurzívnej autoasociatívnej pamäte (RAAM, Pollack [7]), keď použili lineárne neuróny a Ojovo pravidlo Hebbovského učenia. Model *linear RAAM* sa zdá byť v súčastnosti najsilnejší pre túto úlohu, vzhľadom na jeho obrovskú kapacitu a jednoduchšie trénovanie. V tejto práci najprv porovnávame vlastné výsledky s publikovanými v [2]. Potvrdzujeme, že lineárna RAAM má veľkú kapacitu, avšak konečný výsledok je citlivý na počiatočné podmienky. Navyše, iba málo z reprezentovateľných štruktúr je gramaticky správnych. V ďalšej časti skúmame vplyv reprezentácií symbolov na konečné vlastnosti siete. Najviac štruktúr vie sieť reprezentovať, ak použijeme reprezentácie vytvorené na základe charakteristických vlastností slov [13]. Naopak, pri použití reprezentácií popisujúcich kontext [11] je celková chyba rekonštrukcie najmenšia. Navyše, pri použití týchto typov reprezentácií dosahuje lineárna RAAM tretiu úroveň systematickosti podľa klasifikácie Niklassona & van Geldera [17].

Obsah

Zoznam obrázkov

Zoznam tabuliek

Kapitola 1

Úvod

Štruktúrované dáta sa prirodzene vyskytujú napríklad v jazyku (každá veta jazyka má aj svoju hĺbkovú štruktúru v podobe stromu). Klasické modely ľudskej kognície využívajú symbolický prístup. Pri manipulácií štruktúrovaných objektov (zoznamy, grafy, stromy) môžu efektívne pristupovať k jednotlivým prvkom (konštituentom) a podľa potreby ich meniť. Reprezentácie nových objektov vytvárajú zreťazením reprezentácií jednotlivých konštituentov. V roku 1988 Fodor a Pylyshyn [4] vzniesli kritiku na konekcionizmus. Tvrdili, že konekcionistické modely nedokážu reprezentovať a spracovať štruktúrované dáta bez toho, aby v konečnom dôsledku implementovali symbolické prístupy.

Jednou z odpovedí na kritiku bol model, ktorý v roku 1990 predstavil Pollack [7]. Rekurentná neurónová sieť trénovaná rekurzívne ako autoasociatívna pamäť metódou spätného šírenia chýb (error back-propagation) bola schopná naučiť sa vytvárať reprezentácie pevnej dĺžky pre potenciálne ľubovolne veľké n-árne stromy. Navyše umožňovala efektívny prístup k jednotlivým listom stromu. Model nazvaný Recursive Auto-Associative Memory (RAAM, rekurzívna autoasociatívna pamäť) znamenal pre zástancov konekcionizmu novú nádej. RAAM bola použitá v kombinácii s obyčajnou doprednou sieťou pri holistických[1] transformáciach viet prirodzeného jazyka z aktívneho do pasívneho tvaru [20, 21], alebo pri transformáciach logických výrazov [17]. Tiež boli vytvorené rôzne modifikácie RAAM. Napríklad Sperduti [23] upravil model tak, aby bol schopný reprezentovať grafové štruktúry.

[1]Holistická operácia je akýkoľvek výpočet, ktorý sa môže aplikovať na všetky konštituenty objektu súčastne, bez nutnosti lokalizovať, extrahovať či modifikovať jeho konštituenty (Hammerton, [19]).

Avšak pôvodný model RAAM nebol bez nedostatkov. Ukázalo sa, že je obtiažne sieť dobre natrénovať, lebo trénovanie je založené na metóde spätného šírenia chýb. Druhým, azda ešte väčším nedostatkom sú slabé schopnosti zovšeobecnenia. Sieť je schopná reprezentovať iba niekoľkonásobne viac štruktúr, ako jej bolo predložených počas trénovania.

Tieto nedostatky sa podarilo zmierniť v roku 2005 v modifikácii pôvodného modelu nazvanej lineárna RAAM. Voegtlin & Dominey [2] použili lineárne neuróny a učenie metódou spätného šírenia chýb nahradili Ojovým pravidlom Hebbovského učenia. V porovnaní s pôvodným modelom dosahuje lineárna RAAM o rád lepšie generalizačné schopnosti, nepodlieha tzv. efektu pretrénovania a za určitých podmienok je možné dopredu predpovedať, aké štruktúry bude sieť schopná reprezentovať [2]. Ďalej autori ukázali, že nové štruktúry zachovávajú poradie slov (prvkov) vzhľadom na ich výskyt v štruktúrach z trénovacej množiny.

Vo svojej práci Voegtlin & Dominey tiež naznačili, ako vplýva na generalizačné vlastnosti siete výber reprezentácií atomických prvkov (terminálov), z ktorých môžu byť štruktúry zložené. Pollack [7] v jednom zo svojich experimentov trénoval sieť na množine anglických viet pretransformovaných na ternárne stromy. V reprezentáciach jednotlivých slov použil dodatočnú informáciu o príslušnosti ku gramatickej kategórii (podstatné mená a pod.). Voegtlin & Dominey ukázali, že tým sa úloha pre samotnú sieť zjednodušila. Pri použití neutrálneho kódu, ktorý iba rozlišuje slová medzi sebou a nenesie v sebe dodatočnú informáciu, lineárna RAAM nebola schopná reprezentovať všetky zmysluplné vety.

Spracovanie štruktúrovaných dát je pre neurónové siete ťažkou úlohou. Zároveň je to aj veľká výzva, ak majú neurónové siete uspieť ako modely ľudskej kognície. Lineárna RAAM je v súčastnosti nový model, ktorý sa javí ako najsilnejší pre úlohy tohto typu.

V mojej diplomovej práci sa najskôr v kapitole 2 venujem podrobnému popisu lineárnej RAAM. V kapitole 3 potom uvádzam mnou namerané výsledky rovnakého experimentu, na ktorom Voegtlin & Dominey demonštrovali výhody nového modelu, a porovnávam ich s publikovanými závermi.

Lokalistické kódovanie pravdepodobne nezodpovedá spôsobu, akým sa reprezentujú informácie v mozgu. Preto má význam skúsiť použiť kódovanie, ktoré nejakým spôsobom obnáša dodatočné informácie, a skúmať jeho vplyv na zmysluplné zovšeobecňovanie. V kapitole 4 porovnávam na rozsiahlom

experimente vplyv kódovania terminálnych symbolov celkom pre tri spôsoby reprezentácií.

Niklasson & van Gelder [17] definovali päť úrovní systematickosti podľa toho, v akom vzťahu sú nové reprezentovateľné štruktúry a trénovacie vzory. V kapitole 5 ukážeme, že lineárna RAAM dosahuje tretiu úroveň systematickosti pri použití vhodných reprezentácií terminálov.

Pre účely experimentovania s lineárnou RAAM som vytvoril päť programov, ktoré zahŕňajú najčastejšie operácie spojené s trénovaním modelu a testovaním jeho vlastností. Programy sú napísané v jazyku C a sú navrhnuté tak, aby sa dali jednoducho používať z príkazového riadku. Na priloženom CD sa nachádzajú zdrojové kódy a podrobná dokumentácia v anglickom jazyku.

Kapitola 2

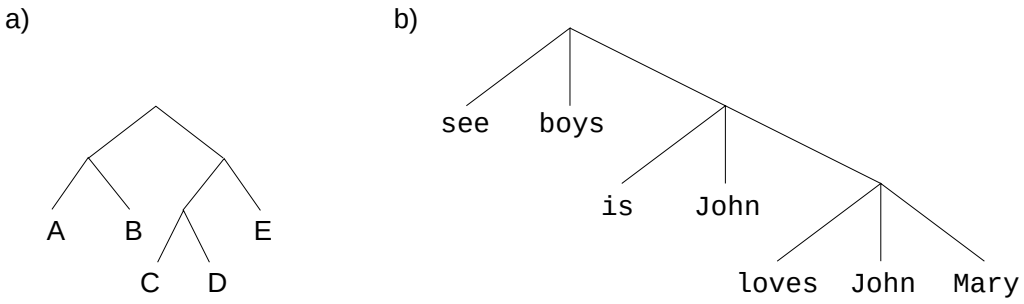
Lineárna RAAM

Model lineárnej rekurzívnej autoasociatívnej pamäte (*linear Recursive Auto-Associative Memory*, linear RAAM) predstavil nedávno vo svojej dizertačnej práci a následne v samostatnom článku Thomas Voegtlin[1] [1, 2]. Ide o modifikáciu klasického modelu RAAM, ktorý koncom 80. rokov minulého storočia uviedol Jordan B. Pollack [7, 8]. Oba typy modelov slúžia na reprezentovanie stromovitých štruktúr vektormi konštantnej dimenzie, pričom lineárna verzia má väčšiu kapacitu a zmierňuje problémy, ktoré sa vyskytli v súvislosti s trénovaním klasického modelu.

Od pôvodného modelu sa lineárna RAAM líši v troch bodoch. Po prvé, kým klasická RAAM sa skladá z nelineárnych neurónov so sigmoidálnou aktivačnou funkciou, lineárna RAAM – a odtiaľ pochádza aj jej názov – používa lineárne neuróny s identickou aktivačnou funkciou. Druhý rozdiel je v spôsobe, akým sú počas trénovania modifikované váhy spojení. Klasická RAAM je trénovaná metódou spätného šírenia chýb (*error backpropagation*). Váhy v lineárnej RAAM sú modifikované podľa Ojovho pravidla pre metódu hlavných komponentov (*Principal Components Analysis*, PCA). Od toho je odvodený starší názov PCA-RAAM. Posledný rozdiel medzi oboma modelmi je v tom, že lineárna RAAM používa tie isté váhy na kódovanie aj na dekódovanie štruktúr.

Text v tejto kapitole je organizovaný nasledovne. Najskôr zavedieme základné pojmy. Potom popíšeme stavbu siete. S tým súvisí aj spôsob, akým sieť vytvorí reprezentáciu štruktúry a naopak, ako z reprezentácie zrekonštruuje pôvodný objekt. Na záver popíšeme celý proces trénovania siete.

[1] spoluautorom článku je Peter F. Dominey



Obr. 2.1: Príklad binárnej a ternárnej štruktúry. Strom na ľavej strane má v listoch znaky abecedy bez hlbšieho významu. Terminálne symboly v listoch druhej štruktúry sú slová, ktoré majú z ľudského hľadiska daný význam. Avšak pre sieť nie je podstatné, aké terminály zvolíme, ale to, aké reprezentácie im priradíme.

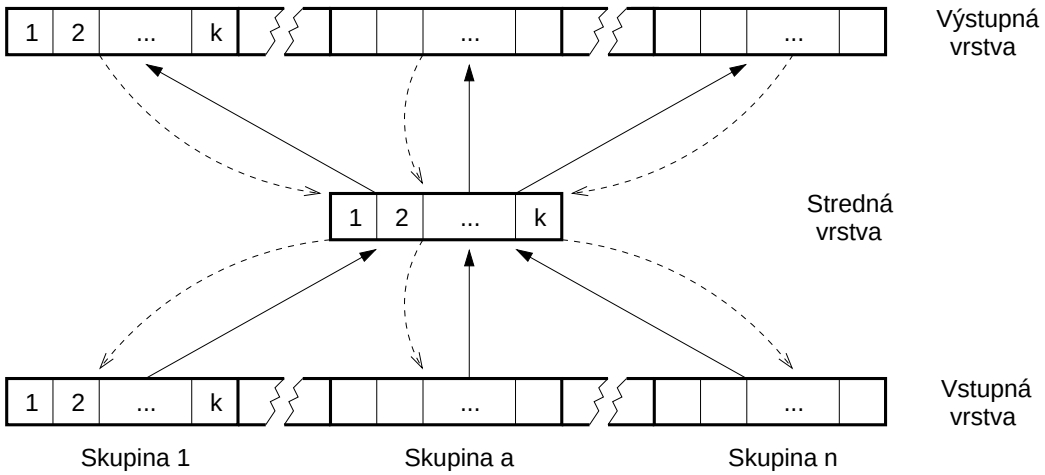
2.1 Základné pojmy

Lineárna RAAM je trojvrsvtvová neurónová sieť schopná naučiť sa reprezentovať n-árne stromovité štruktúry. Arita štruktúr určuje *aritu* samotného modelu. Podľa toho budeme hovoriť napríklad o binárnej alebo ternárnej RAAM. Hoci arita môže teoreticky byť ľubovolné prirodzené číslo väčšie ako jedna, v tejto práci sa budeme zaoberať výhradne binárnou a ternárnou lineárnou RAAM.

Štruktúry, ktoré je lineárna RAAM schopná naučiť sa reprezentovať, majú tvar stromu, ktorého listy nesú informáciu v podobe slova alebo nejakej značky. Tieto slová resp. značky budeme nazývať *terminálne symboly*, alebo skrátene *terminály*. Príklad binárnej a ternárnej štruktúry môžeme vidieť na obrázku 2.1.

Každému terminálnemu symbolu priradíme reprezentáciu v podobe vektora reálnych čísel pevne zvolenej dimenzie. Budeme hovoriť, že lineárna RAAM má *dimenziu* k, ak sú terminály reprezentované vektormi dimenzie k. Základný typ predstavuje lokalistické reprezentovanie symbolov, ktoré budeme v ďalšom texte označovať *neutrálny kód*. Pri tomto type kódovania priradíme terminálnemu symbolu vektor, ktorý má na jednom (pre každý symbol inom) mieste jednotku a na ostatných miestach nulu. Z toho vyplýva, že žiadne dva terminály nemajú rovnakú reprezentáciu. Zároveň, dimenzia reprezentácií je zdola ohraničená počtom terminálnych symbolov.

Okrem neutrálneho kódovania sa v tejto práci budeme zaoberať repre-



Obr. 2.2: Architektúra n-árnej lineárnej RAAM dimenzie k. Plné šípky znázorňujú úplné prepojenie doprednými váhovanými spojeniami. Prerušované šípky predstavujú spätné spojenia, ktoré s časovým oneskorením kopírujú aktivity neurónov niektorej zo skupín výstupnej vrstvy na strednú vrstvu, resp. zo strednej vrstvy na neuróny niektorej zo skupín vstupnej vrstvy.

zentáciami vytvorenými modelom WCD (word co-occurrence detector, [9, 10, 11]), ktoré popisujú kontext, v akom sa jednotlivé terminály vyskytujú, a reprezentáciami získanými z databázy WordNet [14] extrakciou charakteristických príznakov [13].

2.2 Architektúra modelu

n-árna lineárna RAAM dimenzie k má na vstupnej aj na výstupnej vrstve nk neurónov. Strednú vrstvu tvorí k neurónov. Preto sa v literatúre môžeme stretnúť s označením *nk–k–nk (lineárna) RAAM*. Ako sme už spomenuli, používame lineárne neuróny s identickou aktivačnou funkciou, t.j. také, ktoré posúvajú prichádzajúci signál na svoj výstup nezmenený.

Neuróny na susedných vrstvách sú úplne prepojené doprednými váhovanými spojeniami. Okrem dopredných existujú medzi vrstvami aj spätné, časovo oneskorené *rekurentné* spojenia, ktoré majú nemennú jednotkovú váhu a slúžia na kopírovanie obsahu medzi vrstvami. Neuróny na vstupnej a výstupnej vrstve logicky rozdeľme do n skupín tak, že v každej skupine bude k susediacich neurónoch. Rekurentné spojenia sú vedené tak, aby bolo možné

kopírovať obsah každej zo skupín výstupnej vrstvy na strednú vrstvu, resp. obsah strednej vrstvy na každú zo skupín vstupnej vrstvy. Situáciu lepšie dokumentuje obrázok 2.2.

Z hľadiska účelu môžeme sieť rozdeliť na dve časti – *kóder* a *dekóder*. Ako už napovedajú samotné názvy, kóder bude slúžiť na získavanie redukovaných reprezentácií komplexných štruktúr. Na druhej strane, pomocou dekódera budeme schopní zrekonštruovať pôvodný objekt z jeho redukovanej reprezentácie.

Spôsob, akým sa reprezentujú štruktúrované objekty v bežných počítačoch, vyžaduje pamäť úmernú veľkosti a zložitosti samotnej štruktúry. Modely neurónových sietí typu RAAM rekurzívne zobrazujú reprezentácie podštruktúr resp. terminálnych symbolov do reprezentácie celej štruktúry, využívajúc pritom obmedzené prostriedky v podobe konštantného počtu neurónov. Preto v tomto prípade hovoríme o *redukovaných reprezentáciach*.

2.2.1 Kódovanie štruktúr

Kóder je časť siete skladajúca sa zo vstupnej a strednej vrstvy a z príslušných spojení medzi nimi. Úlohou kódera je vygenerovať redukovanú reprezentáciu štruktúry na vstupe.

Spracovanie štruktúry prebieha rekurzívne zdola nahor. V každom kroku skopírujeme reprezentácie terminálov, resp. podštruktúr do príslušných skupín na vstupnej vrstve. Následne zobrazíme aktivity neurónov vstupnej vrstvy na strednú vrstvu, kde dostaneme redukovanú reprezentáciu štruktúry. V prvom kroku vieme vygenerovať reprezentáciu iba takej časti celej štruktúry, ktorá má ako svoje podštruktúry iba terminálne symboly, lebo ich reprezentácie sú dané na začiatku. V každom ďalšom kroku potom môžeme spracovávať tie časti celej štruktúry, ktoré sú zložené z terminálov alebo podštruktúr so známou reprezentáciou.

Celý proces generovania redukovanej reprezentácie štruktúry prevedieme na príklade binárneho stromu z obrázka 2.1a. V prvom kroku vieme získať reprezentácie podstromov `(A B)` a `(C D)`. Potom pomocou rekurentných spojení skopírujeme zo strednej vrstvy reprezentáciu podstromu `(C D)` na neuróny prvej skupiny vstupnej vrstvy. Do druhej skupiny načítame reprezentáciu znaku `E`. Po zobrazení dostaneme na strednej vrstve reprezentáciu podstromu `((C D) E)`. Posledný krok pozostáva zo skopírovania strednej vrstvy do druhej skupiny a načítania podstromu `(A B)` do prvej skupiny. Aktivity

neurónov strednej vrstvy potom budú zodopvedať redukovanej reprezentácii celého stromu. Keďže redukovanú reprezentáciu štruktúry dostaneme na strednej vrstve, budeme ju nazývať aj *kompresná vrstva*.

Keď označíme $z_j^{(a)}$ aktivitu j-teho neurónu a-tej skupiny vstupnej vrstvy, c_i aktivitu i-teho neurónu kompresnej vrstvy a $w_{ij}^{(a)}$ synaptickú váhu spojenia týchto dvoch neurónov, môžeme vyjadriť distribúciu aktivít v kóderi vzťahom

$$\begin{aligned} c_i &= \sum_{j=1}^{k} w_{ij}^{(1)} z_j^{(1)} + \sum_{j=1}^{k} w_{ij}^{(2)} z_j^{(2)} + \ldots + \sum_{j=1}^{k} w_{ij}^{(n)} z_j^{(n)} \\ &= \sum_{a=1}^{n} \sum_{j=1}^{k} w_{ij}^{(a)} z_j^{(a)} \end{aligned} \tag{2.1}$$

Nech $\mathbf{z}^{(a)}$ je k-rozmerný stĺpcový vektor reprezentujúci aktivity neurónov v a-tej skupine vstupnej vrstvy, $\mathbf{c}$ je vektor aktivít neurónov strednej vrstvy a $\mathbf{W}^{(a)} = \|w_{ij}^{(a)}\|_{k\times k}$ je matica váh dopredných spojení medzi a-tou skupinou vstupnej vrstvy a strednou vrstvou. Vzťah 2.1 zapíšeme v maticovej podobe nasledovne:

$$\mathbf{c} = \mathbf{W}^{(1)}\mathbf{z}^{(1)} + \mathbf{W}^{(2)}\mathbf{z}^{(2)} + \ldots + \mathbf{W}^{(n)}\mathbf{z}^{(n)} \tag{2.2}$$

Keď ďalej označíme $\mathbf{W} = [\mathbf{W}^{(1)}; \mathbf{W}^{(2)}; \ldots; \mathbf{W}^{(n)}]$, $\mathbf{W} \in \mathbb{R}^{k\times nk}$, maticu váh všetkých dopredných spojení kódera a $\mathbf{z} = [\mathbf{z}^{(1)^T}; \mathbf{z}^{(2)^T}; \ldots; \mathbf{z}^{(n)^T}]^T$, $\mathbf{z} \in \mathbb{R}^{nk\times 1}$, vektor aktivít vstupnej vrstvy, rovnica 2.1 získava podobu

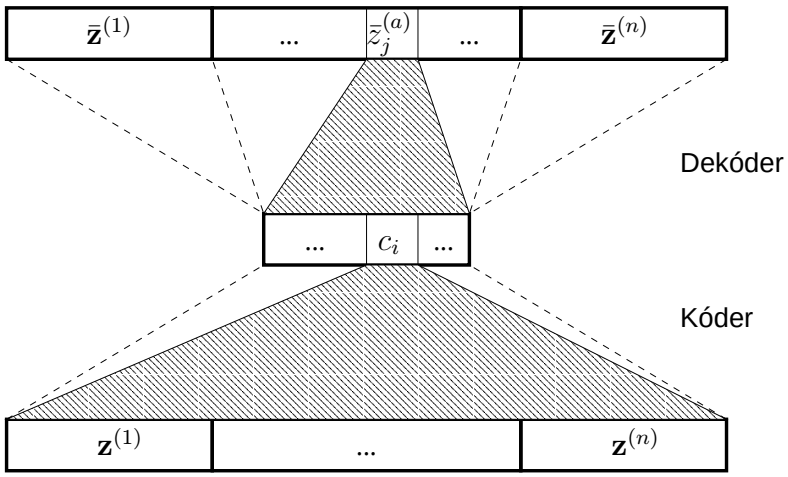
$$\mathbf{c} = \mathbf{W}\mathbf{z} \tag{2.3}$$

Predchádzajúce vzťahy graficky znázorňuje obrázok 2.3.

2.2.2 Proces dekódovania

Dekóder tvorí stredná a výstupná vrstva lineárnej RAAM spolu s príslušnými doprednými a rekurentnými spojeniami. Pripomeňme si, že úlohou dekódera je zrekonštruovať pôvodnú štruktúru z jej redukovanej reprezentácie.

Rekonštrukcia prebehne tiež rekurzívne, tentoraz však zhora nadol. Reprezentáciu celej štruktúry v podobe aktivít neurónov strednej vrstvy distribuujeme na výstupnú vrstvu. V jednotlivých skupinách dostaneme reprezentácie príslušných podstromov. Ak aktivity neurónov niektorej zo skupín nereprezentujú terminálny symbol, potom je nutné ďalšie dekódovanie. Rekurentnými spojeniami skopírujeme z príslušnej skupiny vzor aktivít na strednú



Obr. 2.3: Distribúcia aktivít neurónov v lineárnej RAAM. Vyšrafované oblasti znázorňujú váženú sumu aktivít neurónov na vstupnej, resp. na strednej vrstve, popísanú rovnicami 2.1, resp. 2.4.

vrstvu a pokračujeme v dekódovaní, až kým každá skupina neobsahuje reprezentáciu terminálneho symbolu.

Proces dekódovania ilustrujeme na ternárnej štruktúre z obrázka 2.1b. V prvom kroku skopírujeme na strednú vrstvu reprezentáciu celého stromu. Jej rozložením dostaneme na výstupnej vrstve v prvej skupine neurónov reprezentáciu terminálu `see` a v druhej skupine reprezentáciu terminálu `boys`. Vzorec aktivít tretej skupiny neurónov nepredstavuje reprezentáciu žiadneho z terminálnych symbolov. Preto obsah neurónov skopírujeme na strednú vrstvu a opäť rozložíme. V prvej a druhej skupine teraz dostaneme reprezentácie terminálov `is`, resp. `John`. Obsah tretej skupiny musíme ešte raz skopírovať na strednú vrstvu. Po rozložení je rekonštrukcia štruktúry dokončená, nakoľko všetky tri skupiny výtupnej vrstvy budú reprezentovať terminály, konkrétne `loves`, `John` a `Mary`.

Stromovité štruktúry vo všeobecnosti môžu mať na jednej úrovni hĺbky viacero vrcholov, ktoré nie sú listami. Napríklad obaja potomkovia koreňa stromu na obrázku 2.1a nie sú listy. Vtedy sa v priebehu kódovania dostávame do situácie, keď vygenerovanú reprezentáciu jednej podštruktúry musíme uskladniť v nejakom zásobníku a neskôr, až získame reprezentáciu zvyšných podštruktúr, zo zásobníka skopírovať na vstupnú vrstvu. Podobný problém riešime aj pri dekódovaní reprezentácie do štruktúry. Skôr než rekurzívne rozbalíme reprezentáciu niektorej podštruktúry, musíme si do zásobníka

uložiť reprezentácie ostatných podštruktúr, ktoré nie sú terminálmi.

Ostáva nám vyjadriť distribúciu signálu zo strednej vrstvy na výstupnú vrstvu. Podobne ako v predchádzajúcom odseku označme c_i aktivitu i-teho neurónu strednej vrstvy, $\bar{z}_j^{(a)}$ aktivitu j-teho neurónu a-tej skupiny výstupnej vrstvy a $w_{ij}^{(a)}$ váhu spojenia týchto dvoch neurónov. Dostávame

$$\bar{z}_j^{(a)} = \sum_{i=1}^{k} w_{ij}^{(a)} c_i \tag{2.4}$$

Pripomíname, že na dekódovanie sa používa tá istá matica váh. Môžeme sa na celý proces dívať aj tak, že kóder i dekóder sú totožné siete, akurát v dekóderi sa aktivity distribuujú smerom z kompresnej vrstvy, teda opačne ako v kóderi.

Uveďme ešte maticový tvar vzťahu 2.4. Rekonštrukciu reprezentácie a-tej podštruktúry vyjadríme vzťahom

$$\bar{\mathbf{z}}^{(a)} = \mathbf{W}^{(a)^T}\mathbf{c} \tag{2.5}$$

Zobrazenie aktivít kompresnej vrstvy na výstupnú vrstvu popisuje rovnica

$$\bar{\mathbf{z}} = \mathbf{W}^T\mathbf{c} \tag{2.6}$$

Avšak proces dekódovania má v sebe skrytý problém. Doteraz sme predpokladali, že zrekonštruované reprezentácie terminálov sú úplne totožné s ich skutočnými reprezentáciami. Nebrali sme do úvahy možnosť nepresnej rekonštrukcie. Presnejšie vyjadrené, ak na vstupe kódera bol vektor $\mathbf{z}$ s redukovanou podobou $\mathbf{c} = \mathbf{W}\mathbf{z}$, po rozbalení na výstupnú vrstvu dekódera sme očakávali, že bude platiť

$$\mathbf{z} = \mathbf{W}^T\mathbf{W}\mathbf{z} = \mathbf{W}^T\mathbf{c} = \bar{\mathbf{z}} \tag{2.7}$$

Vo všeobecnom prípade sieť nevie vytvoriť takú redukovanú reprezentáciu štruktúry, ktorú by potom vedela bez chyby rekurzívne rozložiť až po terminálne symboly. Inými slovami, nie je možné vždy dosiahnúť stav, kedy by 2.7 platilo pre všetky vstupné vzory $\mathbf{z}$ v rámci danej štruktúry[2]. Preto

[2] Pri tvorbe redukovaných reprezentácií totiž dochádza k redukcii dimenzie. Hodnosť matice $\mathbf{P} = \mathbf{W}^T\mathbf{W}$ je najviac k, kým dimenzia vstupných vektorov je až nk. Ak je teda na vstupe viac ako k lineárne nezávislých vektorov, ich obrazy v zobrazení definovanom maticou P budú určite lineárne závislé vektory a teda rôzne od pôvodných.

musíme pre dekóder zaviesť pravidlo, podľa ktorého bude možné určiť, či sa pozorovaný vzorec aktivít dá považovať za reprezentanta terminálneho symbolu, alebo je nutné ďalšie dekódovanie. Toto pravidlo má názov *terminálny test*.

2.2.3 Terminálny test

Terminálny test, ktorý používal Pollack [7], porovnával aktivity jednotlivých neurónov. V jeho experimentoch boli terminály reprezentované vektormi núl a jednotiek. Rekonštruovaný vektor bol považovaný za terminálny symbol, ak všetky hodnoty aktivít boli menšie ako τ, ak mal byť výsledok 0, resp. väčšie ako $1-\tau$, ak mal byť výsledok 1. Pollack pracoval s hodnotou $\tau = 0.2$.

Voegtlin & Dominey [2] používali meranie euklidovskej vzdialenosti. V článku sa priamo píše, citujem:

> „A decoded vector was considered to encode a terminal if and only if the Euclidean distance to the vector encoding this terminal was below a treshold, θ."
>
> Preklad: „Dekódovaný vektor bol považovaný za reprezentanta terminálu práve vtedy, keď euklidovská vzdialenosť od vektora reprezentujúceho tento terminál bola menšia ako prah θ."
>
> Voegtlin & Dominey, [2, časť *3.3 Terminal test*]

Parameter θ budeme v tejto práci označovať *prah rekonštrukcie* (*reconstruction threshold*). Ak zvolíme jeho hodnotu príliš veľkú, môže za istých okolností nastať situácia, že dekódovaný vektor je vzdialený o povolenú dĺžku od dvoch alebo viacerých reprezentácií terminálov. Kvôli tomuto prípadu považujem citovanú formuláciu za nejasnú. Z nej by totiž vyplývalo, že dekódovaný vektor reprezentuje všetky takéto terminály, čo je zjavne neprípustné.

Terminálny test porovnávajúci euklidovskú vzdialenosť s hodnotou parametra θ je výhodnejší z toho hľadiska, že je aplikovateľný aj na iné než binárne kódy terminálnych symbolov. Preto ho budem používať aj v tejto práci, avšak budem sa držať nasledovnej konvencie:

> Dekódovaný vektor reprezentuje terminál Ψ práve vtedy, keď jeho euklidovská vzdialenosť od reprezentácie terminálu Ψ je menšia ako zvolený prah rekonštrukcie θ a zároveň euklidovská vzdialenosť od reprezentácie každého iného terminálu je väčšia ako θ.

Akonáhle teda môže nastať viacznačnosť v dekódovaní, budem dekódovanie považovať za neúspešné.

V tejto chvíli nám treba povedať, čo budeme rozumieť pod úspešným dekódovaním. Aj keď pripustíme určité nepresnosti pri dekódovaní a zavedieme terminálny test, napriek tomu sa nám nemusí podariť rekonštruovať pôvodnú štruktúru. K tomu môže dôjsť v jednom z nasledujúcich prípadov:

- Už vyššie spomínaná *viacznačnosť*. Rekonštruovaný vektor môže byť pre zvolenú hodnotu prahu θ považovaný za reprezentanta viacerých terminálov. V takomto prípade prerušíme proces dekódovania.
- *Nerozpoznaný terminál*. Rekonštruovaná štruktúra má mať na určitej pozícii terminálny symbol, ale aktivity príslušnej skupiny neurónov výstupnej vrstvy sa príliš líšia od reprezentácie toho terminálu. Dekóder by teda mylne pokračoval v rekonštrukcii a mohol by sa dostať do nekonečného cyklu (*infinite loops*). Nekonečným cyklom predídeme, ak zhora ohraničíme hĺbku štruktúr.
- *„Rozpoznaný" neterminál* (*terminating non-terminal*). Na určitej pozícii dekódujeme terminál, avšak pôvodná štruktúra sa ďalej vetví.
- *Zle rozpoznaný terminál*. Na určitej pozícii dekódujeme terminál, avšak v pôvodnej štruktúre je iný symbol. Aj predošlú, aj túto chybu odhalíme porovnaním pôvodnej štruktúry s tou, ktorú sme zrekonštruovali z redukovanej reprezentácie.

Budeme hovoriť, že *dekódovanie štruktúry bolo úspešné*, ak nenastala ani jedna z vyššie spomenutých chýb, t.j. nedošlo k zacykleniu a rekonštruovaná verzia súhlasí s pôvodnou štruktúrou. O takejto štruktúre budeme hovoriť, že je sieťou *reprezentovateľná*.

2.3 Algoritmus trénovania siete

Keď sme v prechádzajúcich častiach opisovali spracovávanie komplexných objektov a ich následnú rekonštrukciu, predpokladali sme, že váhy spojení sú v konfigurácii, ktorá umožňuje vytvoriť takú reprezentáciu štruktúry, ktorú bude možné so zanedbaním malých chýb transformovať naspäť do pôvodnej štruktúry. Avšak pri inicializácii siete sú váhy dopredných spojení nastavené na náhodnú hodnotu. Počiatočná konfigurácia siete teda takmer určite

nespĺňa naše požiadavky. Riešenie samozrejme spočíva v trénovaní siete prostredníctvom modifikovania synaptických váh.

Sieť budeme trénovať ako autoasociátor, to znamená, že na výstupe dekódera očakávame ten istý vektor, ako bol na vstupnej vrstve. Nech je na vstupe kódera vzorec aktivít $\mathbf{z}$. Na výstupnej vrstve dostaneme pozmenený vektor $\bar{\mathbf{z}}$. Minimalizáciou kvadratickej chyby

$$E(\mathbf{W}) = \|\mathbf{z} - \bar{\mathbf{z}}\|^2 = \sum_{a=1}^{n} \sum_{j=1}^{k} \left(z_j^{(a)} - \bar{z}_j^{(a)} \right)^2 \tag{2.8}$$

dostaneme stochastické pravidlo modifikácie váh[3]

$$\begin{array}{l} \forall 1 \leq i, j \leq k \\ \forall 1 \leq a \leq n \end{array} \qquad \Delta w_{ij}^{(a)} \;=\; \eta \, c_i \left(z_j^{(a)} - \sum_{r=1}^{k} w_{rj}^{(a)} c_r \right) \tag{2.9}$$

kde parameter η predstavuje *rýchlosť učenia*. Toto pravidlo korešponduje s Ojovým pravidlom Hebbovského učenia (pozri napríklad [12]). O tomto pravidle je známe, že nájde podpriestor generovaný k hlavnými komponentami distribúcie vstupného vektora $\mathbf{z}$.

Metóda hlavných komponentov (*Principal Components Analysis*) umožňuje lineárne transformovať dáta z viacrozmerného vstupného priestoru do priestoru *príznakov* s nižšou dimenziou. Pozostáva z nájdenia vektorov, ktoré zodpovedajú smerom s najväčšou varianciou vstupných dát. Je tiež známe, že PCA je optimálny spôsob kódovania náhodných dát vzhľadom na chybu rekonštrukcie. Avšak model lineárnej RAAM vykonáva zložitejšiu operáciu, ako je PCA. Vstupný vektor $\mathbf{z}(t)$ totiž obsahuje redukovanú reprezentáciu $\mathbf{c}(t-1)$, ktorá je výsledkom predošlého vstupu $\mathbf{z}(t-1)$. Takže distribúcia vektora $\mathbf{z}$ nie je definovaná a priori, ale mení sa podľa toho, ako sa menia redukované reprezentácie navrhnuté sieťou. Tento typ učenia sa nazýva *moving target problem*.

Adaptácia siete na celej štruktúre prebieha rekurzívne zdola nahor, podobne ako pri získavaní redukovanej reprezentácie štruktúry. Najprv sieti predkladáme iba tie časti celej štruktúry, ktoré obsahujú iba terminálne symboly. Vždy si zapamätáme reprezentáciu predloženého podstromu a modifikujeme váhy podľa pravidla 2.9. V ďalších krokoch môžeme spracovať tie

[3]Poznamenajme, že váhy sa adaptujú všetky naraz v jednom kroku. Prakticky to môžeme simulovať tak, že najprv vypočítame všetky zmeny váh a potom samotné váhy zmeníme na základe vypočítaných hodnôt.

Vzor aktivít $\mathbf{z}$		Redukovaná reprezentácia		Výstup siete $\bar{\mathbf{z}}$
`(A B)`	$\longrightarrow$	$R_{AB}(t_1)$	$\longrightarrow$	`(`$\bar{\texttt{A}}$ `` ` ``$\bar{\texttt{B}}$`)`
`(C D)`	$\longrightarrow$	$R_{CD}(t_2)$	$\longrightarrow$	`(`$\bar{\texttt{C}}$ `` ` ``$\bar{\texttt{D}}$`)`
`(`$R_{CD}(t_2)$ `E)`	$\longrightarrow$	$R_{CDE}(t_3)$	$\longrightarrow$	`(`$\bar{R}_{CD}(t_2)$ `` ` ``$\bar{\texttt{E}}$`)`
`(`$R_{AB}(t_1)$ $R_{CDE}(t_3)$`)`	$\longrightarrow$	$R_{ABCDE}(t_4)$	$\longrightarrow$	`(`$\bar{R}_{AB}(t_1)$ $\bar{R}_{CDE}(t_3)$`)`

Tabuľka 2.1: Poradie, v akom predkladáme sieti vzory na trénovanie. V ľavom stĺpci v i-tom riadku je vzorec aktivít predkladaný sieti v čase t_i. V strede je označenie pre redukovanú reprezentáciu vygenerovanú na kompresnej vrstve. Rekonštruované vektory sú v pravom stĺpci. Príklad znázorňuje trénovanie siete na štruktúre z obrázka 2.1a.

časti celej štruktúry, ktoré obsahujú ako svoje prvky terminály alebo podštruktúry so získanou reprezentáciou. Tabuľka 2.1 dokumentuje načrtnutý postup na príklade stromu 2.1a.

Jedna epocha, alebo tiež *iterácia* učenia pozostáva z prezentácie štruktúr z trénovacej množiny v náhodnom poradí podľa vyššie uvedeného postupu. Celá trénovacia procedúra teda pozostáva z týchto krokov

1. zvolíme hodnotu parametra η a počet epoch trénovania N
2. inicializujeme váhy spojení na náhodné hodnoty z malého intervalu
3. N-krát prezentujeme sieti v náhodnom poradí všetky štruktúry z trénovacej množiny

Týmto sme dokončili opis lineárnej RAAM. Vplyv jednotlivých parametrov na vlastnosti modelu analyzujeme v ďalších kapitolách na rozličných experimentoch.

Kapitola 3

Pollackov experiment

V tejto kapitole zopakujeme experiment, na ktorom pôvodne Pollack [7] demonštroval schopnosť modelu RAAM reprezentovať štruktúrované objekty, a na ktorom neskôr Voegtlin & Dominey [2] (ďalej V&D) porovnali kapacitu lineárnej RAAM s pôvodným modelom. Našim cieľom bude v základoch overiť publikované výsledky dosiahnuté použitím lineárnej RAAM.

Budeme trénovať lineárnu RAAM na korpuse siedmich binárnych stromov, ktoré jednoduchým spôsobom vyjadrujú stavbu anglických viet. Potom ostestujeme, koľko nových stromov dokáže natrénovaná sieť reprezentovať. Ukážeme, že nami dosiahnuté výstupy približne zodpovedajú publikovaným výsledkom a pokúsime sa vysvetliť všetky odlišnosti.

3.1 Trénovacia množina

Uvažujme anglickú vetu „The little boy ran up the street“. Ak jednotlivé slová vo vete nahradíme ich gramatickými kategóriami, dostaneme postupnosť „`d a n v p d n`“, kde jednotlivé písmená sú skrátené zápisy pre člen (*determiner*), prídavné meno (*adjective*), podstatné meno (*noun*), sloveso (*verb*) a predložku (*preposition*).

Bezkontextová gramatika v tabuľke 3.1 predstavuje spôsob, akým môžeme vytvoriť podobné schématické zápisy viet. Napríklad, predošlú schému dosiahneme odvodením

$$\texttt{S} \rightarrow \texttt{NP VP} \rightarrow \texttt{d AP v PP} \rightarrow \texttt{d a n v p NP} \rightarrow \texttt{d a n v p d n} \qquad (3.1)$$

Strom odvodenia z uvedenej gramatiky jednoduchým spôsobom opisuje stavbu viet. Získame ho miernou úpravou pôvodných pravidiel tak, že okolo pravej

Význam skupiny pravidiel	Pravidlá odvodenia
vetná fráza (*Sentence*)	`S` → `NP VP`, `S` → `NP v`
fráza pods. mena (*Noun Phrase*)	`NP` → `d AP`, `NP` → `d n`, `NP` → `NP PP`
slovesná fráza (*Verb Phrase*)	`VP` → `v NP`, `VP` → `v PP`
predložková fráza (*Preposition Phrase*)	`PP` → `p NP`
fráza príd. mena (*Adjective Phrase*)	`AP` → `a AP`, `AP` → `a n`

zdroj: Pollack [7]

Tabuľka 3.1: Bezkontextová gramatika, z ktorej bola vygenerovaná trénovacia množina. Terminálne symboly sú označené malými písmenami predstavujúcimi skratky pre gramatické kategórie vetných členov. Strom odvodenia je vždy binárny.

strany každého pravidla pridáme zátvorky, napr. `NP` → `(d AP)`. K odvodeniu 3.1 prislúcha strom `((d(an))(v(p(dn))))`. Trénovacia množina pre tento experiment pozostáva zo siedmich takýchto stromov. Celá je uvedená v tabuľke 3.2.

3.2 Procedúra generalizácie

Na vyčíslenie počtu všetkých štruktúr, ktoré je natrénovaná sieť schopná reprezentovať, navrhol Pollack [7, odsek 4.1] procedúru, ktorá postupne generuje stromy poskladané z používaných terminálnych symbolov a predkladá ich sieti. Podotýkame, že výstupom tejto procedúry je zoznam všetkých reprezentovateľných štruktúr, teda nielen tých, ktoré sú gramaticky správne.

Úplné prehľadávanie množiny všetkých binárnych stromov nie je vhodný spôsob generovania kandidátov na reprezentovateľné objekty, nakoľko je táto množina nekonečná a nie je zrejmé, kedy možno prehľadávanie ukončiť. Pollack preto zaviedol predpoklad, že na to, aby bol nejaký strom reprezentovateľný, musia byť reprezentovateľné obidva jeho podstromy.

Zavedením tohoto predpokladu je už samotné riešenie priamočiaré. Budeme si udržiavať a iteratívne zväčšovať množinu všetkých reprezentovateľných štruktúr. Tá na začiatku procedúry obsahuje iba terminálne symboly, ktoré môžeme chápať ako špeciálny prípad reprezentovateľných štruktúr. V každej iterácii skombinujeme dva stromy z množiny, čím dostaneme nového kandidáta na reprezentovateľný objekt. V prípade úspešného dekódovania za-

Prvky trénovacej množiny	Podštruktúry v rámci množiny
`(d(a(a(an))))`	`(an)`
`((dn)(p(dn)))`	`(a(an))`
`(v(dn))`	`(a(a(an)))`
`(p(d(an)))`	`(dn)`
`((dn)v)`	`(p(dn))`
`((dn)(v(d(an))))`	`(d(an))`
`((d(an))(v(p(dn))))`	`(v(d(an)))`
	`(v(p(dn)))`

Tabuľka 3.2: Trénovacia množina, ktorú pôvodne použil Pollack [7] na trénovanie RAAM. V ľavom stĺpci je 7 stromov vygenerovaných z gramatiky uvedenej v tabuľke 3.1, vpravo je zoznam podštruktúr, ktoré sú obsiahnuté v rámci týchto stromov.

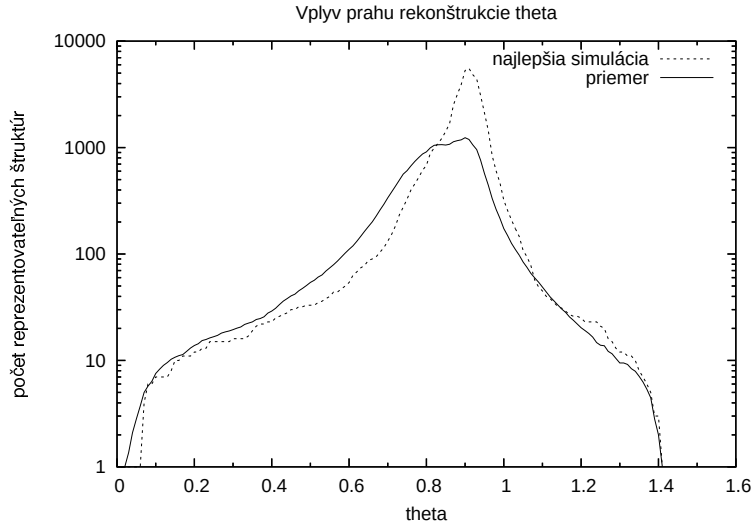
radíme tento objekt do množiny. Procedúra skončí, keď je množina stabilná, t.j. kombináciou jej prvkov nedostaneme nové reprezentovateľné štruktúry. Dodajme, že túto procedúru nie je zložité zovšeobecniť pre ľubovoľné n-árne stromy.

Avšak ani vyššie uvedený predpoklad nám nezaručí, že procedúra skončí. Procedúra nikdy neskončí, ak je sieť schopná reprezentovať nekonečné množstvo štruktúr. Ale to je zatiaľ aj pri použití lineárnej RAAM nereálna predstava.

3.3 Výsledky simulácií

Trénovali sme 20–10–20 lineárnu RAAM na korpuse siedmich uzátvorkovaných výrazov z tabuľky 3.2. Celkovo bolo spustených 11 simulácií s váhami náhodne inicializovanými rovnomerne z intervalu $[-0.5; 0.5]$. Sieť bola zakaždým trénovaná počas 10^5 iterácií, rýchlosť učenia bola $\eta = 0.001$.

Po ubehnutí 10000 iterácií bolo trénovanie na chvíľu prerušené, aby sme pomocou vyššie opísanej generalizačnej procedúry vyčíslili počet štruktúr, ktoré je za toho stavu sieť schopná reprezentovať. Výsledné počty sú znázornené v grafe 3.1 v závislosti od hodnoty prahu rekonštrukcie θ. Uvedené sú jednak priemerné hodnoty zo všetkých simulácií a tiež hodnoty zo simulácie, v ktorej sieť vedela reprezentovať najviac štruktúr. Túto simuláciu budeme ďalej v texte označovať ako *najlepšia*.

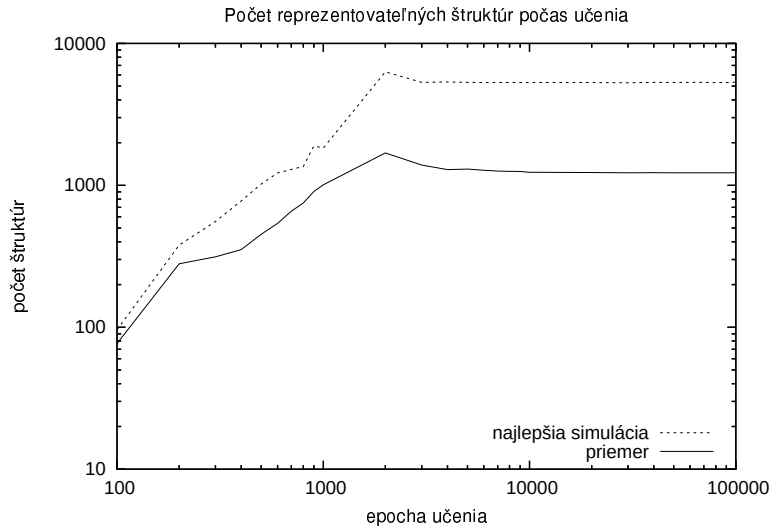


Obr. 3.1: Vplyv voľby prahu rekonštrukcie na generalizačné schopnosti modelu. Spojitá krivka zobrazuje priemerný počet štruktúr, ktoré bola sieť schopná reprezentovať po 10000 iteráciach učenia, v závislosti od hodnoty parametra θ. Priemerné hodnoty sú uvádzané z výsledkov 11 simulácií. Prerušovanou čiarou sú zachytené výsledky *najlepšej* simulácie, čo sa týka počtu reprezentovateľných štruktúr.

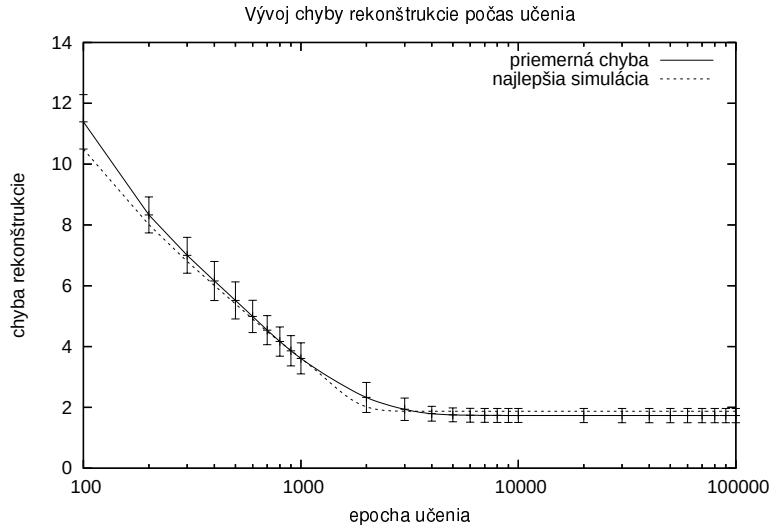
Podobne ako v prípade práce [2], aj teraz sú najväčšie hodnoty namerané približne pre $\theta = 0.9$. Za daných podmienok sieť v priemere reprezentovala 1237 štruktúr a v najlepšom prípade dokonca až 5557 štruktúr, čo je takmer trikrát viac, ako uvádzajú V&D [2].

Ďalej sme priebežne merali vývoj počtu reprezentovateľných štruktúr v závislosti od počtu epoch a celkovú chybu rekonštrukcie štruktúr z trénovacej množiny. Výsledky sú znázornené v grafoch 3.2, resp. 3.3. Uvádzané počty reprezentovateľných štruktúr sú dosiahnuté s optimálnym prahom rekonštrukcie $\theta = 0.9$.

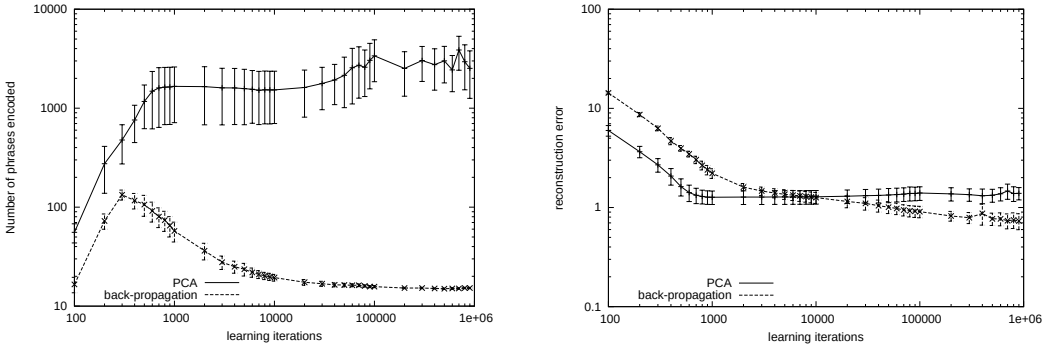
Opäť sme dosiahli istú zhodu s publikovanými výsledkami. Počet reprezentovateľných štruktúr stúpa s počtom epoch učenia, až sa od určitého okamihu ustáli na hodnote okolo 1250 štruktúr. Podobne aj celková chyba rekonštrukcie postupne klesá, až sa ustáli na hodnote $E(\mathbf{W}) \approx 1.73$. Navyše, k ustáleniu oboch kriviek dochádza približne v rovnakom čase, zhruba po 4000 iteráciach.



Obr. 3.2: Počet reprezentovateľných štruktúr v závislosti od počtu trénovacích epoch. Plnou čiarou sú vyznačené priemerné hodnoty z 11 simulácií, prerušovaná krivka znázorňuje výsledky pre najlepšiu simuláciu.



Obr. 3.3: Vývoj chyby rekonštrukcie počas trénovania. V tom čase, ako sa ustálila celková chyba, ustálil sa aj počet reprezentovateľných stromov.



Obr. 3.4: Výsledky publikované v [2]. Grafu vľavo zodpovedá graf 3.2, grafu vpravo zasa 3.3. *Zdroj:* Voegtlin & Dominey [2]

3.3.1 Vplyv rýchlosti učenia

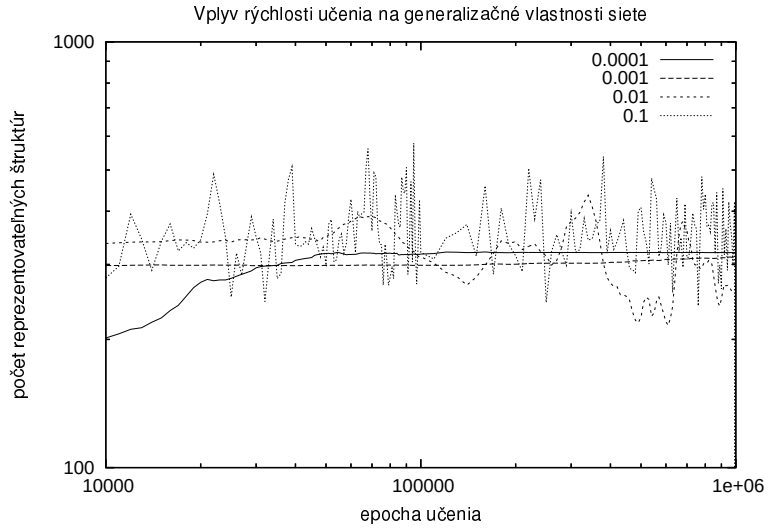
V našich simuláciach sme používali desaťnásobne menšiu rýchlosť učenia ako v [2]. Intuitívne je potom potrebných viac iterácií, kým sa váhy adaptujú. Preto sme vplyv parametra θ (pozri graf 3.1) skúmali až po 10000 iteráciach, a nie po 3000, ako v [2]. Teraz sa pokúsime obhájiť tento krok.

Ako sme už povedali, v grafoch 3.2 a 3.3 sa krivka ustáli približne po 4000 iteráciach. Toto ustálenie je už trvalé v tom zmysle, že hodnoty váh sa menia už iba veľmi pomaly. Avšak priebeh príslušných dvoch grafov v [2][1] je trochu iný. Počet reprezentovateľných štruktúr sa ustáli po 700 iteráciach, aby približne po 20000 iteráciach opäť začal stúpať. Asi po 10^5 iteráciach tento počet začne výrazne kolísať, výkyvy sú rádovo 1000 štruktúr.

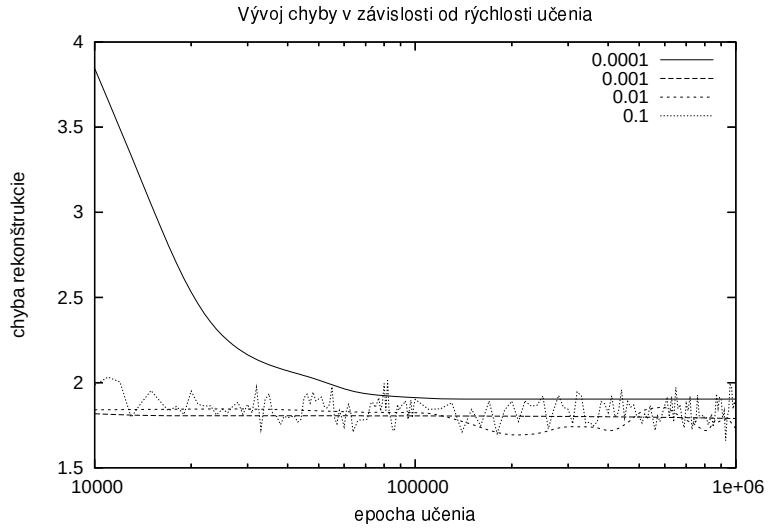
Podobne je to aj s chybou rekonštrukcie, i keď tam výkyvy nie sú na prvý pohľad také výrazné kvôli použitej mierke v grafe. Teraz ukážeme, že tieto výchylky súvisia s hodnotou rýchlosti učenia η. Zopakujeme experiment pre štyri rôzne rýchlosti učenia: 0.0001, 0.001, 0.01 a 0.1.

Pre každú z týchto hodnôt sme spustili 11 simulácií. Váhy boli zakaždým inicializované náhodne z intervalu $[-0.5; 0.5]$. Grafy 3.5 resp. 3.6 zobrazujú pre jednotlivé hodnoty parametra η priemerný počet reprezentovateľných štruktúr, resp. priemernú chybu rekonštrukcie v závislosti od epochy učenia. Hodnota prahu rekonštrukcie bola tentoraz iba $\theta = 0.7$. Tým sa síce počet reprezentovateľných štruktúr zníži, ale iba pomerne k hodnotám, ktoré by sme dostali s $\theta = 0.9$. Zobrazené sú iba hodnoty po prvých 10000 iteráciach, aby boli zreteľné aj menšie zmeny chyby rekonštrukcie.

[1] Pre porovnanie ich uvádzame na obrázku 3.4.



Obr. 3.5: Vývoj počtu reprezentovateľných štruktúr v závislosti od rýchlosti učenia. Každá krivka zobrazuje priemerné hodnoty z 11 simulácií pre danú rýchlosť učenia η. Počet reprezentovateľných štruktúr je uvedený pri použití prahu rekonštrukcie $\theta = 0.7$.



Obr. 3.6: Vývoj chyby rekonštrukcie v závislosti od rýchlosti učenia. Každá krivka zobrazuje priemerné hodnoty z 11 simulácií pre danú hodnotu η.

Z grafu 3.6 vidno, že pre malé hodnoty (10^{-4} a 10^{-3}) sa celková chyba po určitom čase ustáli. Rozdiel je iba v tom, ako rýchlo k tomu dôjde. Pri vyšších rýchlostiach začne krivka oscilovať, pričom pri rýchlosti 0.1 je táto oscilácia veľmi výrazná.

Podobné možno vidieť aj v grafe 3.5. Pre malé hodnoty η sa počet reprezentovateľných štruktúr mení len veľmi málo. Pre veľké hodnoty sa počet štruktúr stále dramaticky mení. Navyše, rast resp. pokles počtu štruktúr nezodpovedá poklesu resp. rastu chyby rekonštrukcie. Inými slovami, menšia chyba neznamená automaticky väčší počet reprezentovateľných štruktúr.

Na základe týchto pozorovaní považujem za vhodnejšie trénovať sieť s nie príliš veľkou rýchlosťou. Tak sa po istom čase sieť dostane do stavu, kedy sa už váhy nebudú veľmi meniť a s nimi sa nebudú príliš meniť ani vlastnosti siete. Podotýkame, že vhodnú rýchlosť η treba hladať pre každý experiment zvlášť, t.j. neexistuje nejaká univerzálna hodnota, s použitím ktorej by sa sieť zakaždým dostala do stabilného stavu[2]. V kapitole 4 tiež ukážeme, ako súvisí rýchlosť učenia s počiatočnými hodnotami váh.

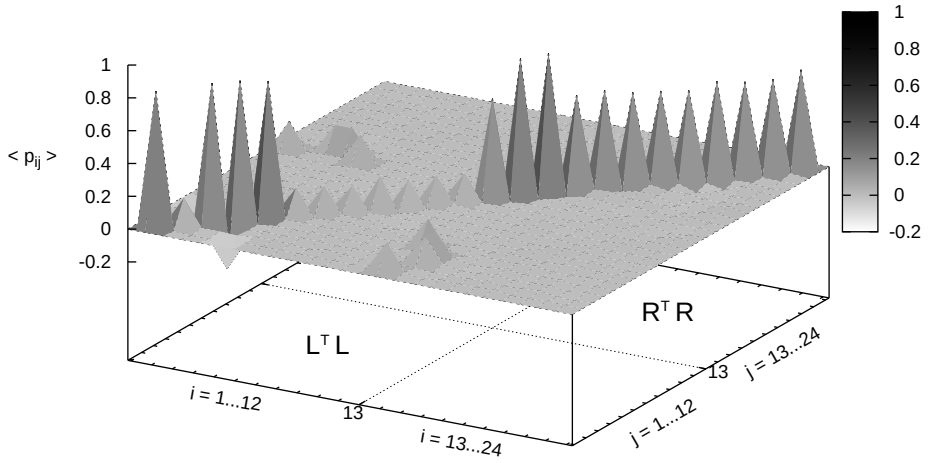
3.3.2 Rôznorodosť výsledkov

Hoci sieť vedela v priemere reprezentovať rádovo toľko štruktúr, ako uvádzajú V&D [2], výsledky jednotlivých simulácií sa od seba veľmi líšili. Kým v najlepšej simulácii sme dosiahli až 5557 reprezentovateľných štruktúr, v *najhoršej* simulácii to bolo len 395.

Otázka teraz znie, prečo sa tak výsledky jednotlivých simulácií od seba odlišujú. Pri hľadaní odpovede sme najskôr zisťovali, ako vyzerajú váhy po skončení trénovacieho procesu. Presnejšie, spustili sme 100 simulácií s náhodne inicializovanými váhami a chceli sme vedieť, či výsledné matice váh, chápané ako body vysokorozmerného priestoru, netvoria skupiny (*clusters*).

Na Pollackovom korpuse (tabuľka 3.2) sme trénovali lineárnu 24–12–24 RAAM. V&D [2] ukázali, že $k = 12$ je minimálna hodnota dimenzie vektorov reprezentácií, pre ktorú je natrénovaná sieť schopná reprezentovať celú trénovaciu množinu 3.2 pri ľubovoľne malom prahu θ. Rýchlosť učenia bola $\eta = 0.001$, trénovanie sme zakaždým ukončili po 10^5 epochách. Avšak zistili sme, že výsledné matice váh sa na seba nepodobali.

[2]Samozrejme, takmer určite by sme vždy uspeli, keby sme zvolili povedzme $\eta = 10^{-10}$, ale tým by sa trénovanie neúnosne predĺžilo.



Obr. 3.7: Priemerná matica $< \mathbf{P} >=< \mathbf{W}^T\mathbf{W} >$ transformácie vstupných vektorov na ich rekonštruované obrazy. Hodnoty jej prvkov $< p_{ij} >$ pre $1 \leq i, j \leq 24$ sú znázornené v zvislom rozmere.

Potom sme pre každú simuláciu vypočítali maticu $\mathbf{P} = \mathbf{W}^T\mathbf{W}$ transformácie vstupných vektorov na ich rekonštruované obrazy. Teraz bola podobnosť evidentná. V grafe 3.7 uvádzame priemerné hodnoty $< p_{ij} >$ zo 100 simulácií.

Ako sme povedali v kapitole 2 v poznámke 2, hodnosť matice $\mathbf{P} \in \mathbb{R}^{24\times 24}$ môže byť nanajvýš rovná 12. Teda určite $\mathbf{P} \neq \mathbf{I}_{24}$. Avšak z grafu 3.7 vidno, že priemerná matica $< \mathbf{P} >$ má do určitej miery charakter jednotkovej matice.

Zvláštne sa môže zdať, že niektoré prvky na diagonále $< p_{ii} >$ majú malé hodnoty v porovnaní s ostatnými. Vysvetlenie môže byť nasledovné. Podobne ako v [2][3] označme maticu váh $\mathbf{W} = [\mathbf{L}; \mathbf{R}]$. Potom platí

$$\mathbf{P} \;=\; \mathbf{W}^T\mathbf{W} = \begin{bmatrix} \mathbf{L}^T \\ \mathbf{R}^T \end{bmatrix} [\, \mathbf{L}; \mathbf{R} \,] = \begin{bmatrix} \mathbf{L}^T\mathbf{L} & \mathbf{L}^T\mathbf{R} \\ \mathbf{R}^T\mathbf{L} & \mathbf{R}^T\mathbf{R} \end{bmatrix} \tag{3.2}$$

Nech je na vstupe vektor $\mathbf{z}$, v riadkovej podobe zapísaný $\mathbf{z} = [\mathbf{l}; \mathbf{r}]$. Potom

[3]Podľa označenia zavedeného v kapitole 2 je $\mathbf{L} = \mathbf{W}^{(1)}$ a $\mathbf{R} = \mathbf{W}^{(2)}$.

pre rekonštrukciu $\bar{\mathbf{z}}$ platí:

$$\bar{\mathbf{z}} \;=\; \mathbf{P}\mathbf{z} = \begin{bmatrix} \mathbf{L}^T\mathbf{L} & \mathbf{L}^T\mathbf{R} \\ \mathbf{R}^T\mathbf{L} & \mathbf{R}^T\mathbf{R} \end{bmatrix} \begin{bmatrix} \mathbf{l} \\ \mathbf{r} \end{bmatrix} = \begin{bmatrix} \mathbf{L}^T\mathbf{L}\mathbf{l} + \mathbf{L}^T\mathbf{R}\mathbf{l} \\ \mathbf{R}^T\mathbf{L}\mathbf{r} + \mathbf{R}^T\mathbf{R}\mathbf{r} \end{bmatrix} \tag{3.3}$$

Z grafu 3.7 môžeme približne odhadnúť $< \mathbf{L}^T\mathbf{R} > \doteq \mathbf{0}$ a tiež $< \mathbf{R}^T\mathbf{L} > \doteq \mathbf{0}$. Preto pre priemerný prípad platí

$$< \mathbf{P} > \mathbf{z} \doteq \begin{bmatrix} < \mathbf{L}^T\mathbf{L} > \mathbf{l} \\ < \mathbf{R}^T\mathbf{R} > \mathbf{r} \end{bmatrix} \tag{3.4}$$

Matica $< \mathbf{R}^T\mathbf{R} >$ má charakter jednotkovej matice, teda približne platí $< \mathbf{R}^T\mathbf{R} > \mathbf{r} = \mathbf{r}$, teda vektor dekódovaný na pravej pozícii sa približne rovná vzoru na vstupe. Avšak matica $< \mathbf{L}^T\mathbf{L} >$ je tvaru $diag\,(1, \varepsilon, 1, 1, 1, \varepsilon, \ldots, \varepsilon)$, kde $\varepsilon \doteq 0.15$. Keď pozorne preskúmame trénovaciu množinu – Pollakov korpus, zbadáme, že na ľavých pozíciach, t.j. ako vektor $\mathbf{l}$, sa nevyskytuje terminál `a`, v našom prípade reprezentovaný vektorom $(0, 1, 0, \ldots, 0)$. Preto sa dá povedať, že nie je „potrebné“, aby bola hodnota $< p_{2,2} >$ rovná 1. Ďalej si môžeme všimnúť, že iba dve štruktúry sa vyskytujú ako ľavé podstromy: `(dn)` a `(d(an))`. Ak predpokladáme, že 4 z prvých 5 prvkov na diagonále ($p_{1,1}$ a $p_{3,3}$ až $p_{5,5}$) potrebujeme na správne dekódovanie terminálnych symbolov, zvyšné prvky matice $\mathbf{L}^T\mathbf{L}$ môžu byť ľubovoľne využité na správne dekódovanie vyššie spomenutých dvoch štruktúr. Teda nie je nutné, aby sa aj $p_{6,6}$ až $p_{12,12}$ približne rovnali 1.

Vráťme sa teraz naspäť k pôvodným jedenástim simuláciam z časti 3.3. Už sme spomínali, že sme pozorovali veľké rozdiely v kapacite natrénovanej siete. Keď sme však prezreli všetky štruktúry reprezentovateľné sieťou z najlepšej simulácie, zistili sme, že iba 31 z celkového počtu 5557 je vygenerovateľných z gramatiky 3.1 a z týchto 31 je iba 16 takých, ktoré sa nevyskytli v rámci trénovacej množiny. Navyše, sieť z najhoršej simulácie vedela reprezentovať 27 gramatických štruktúr, čo je iba o 4 menej ako v najlepšom prípade. Všetkých 31 štruktúr je uvedených v tabuľke 3.3.

Pollack [7] uviedol, že natrénovaná klasická 20–10–20 RAAM vedela reprezentovať 31 nových štruktúr, z ktorých bolo 19 gramaticky správnych. Takže z hľadiska počtu nových, gramaticky správnych reprezentovateľných štruktúr dosahuje lineárna RAAM rovnakú mieru zovšeobecnenia ako klasický model.

Typ	(nová)	Štruktúra
NP		`(dn)`
AP		`(an)`
NP		`(d(an))`
VP		`(v(dn))`
PP		`(p(dn))`
AP		`(a(an))`
S		`((dn)v)`
NP	$\longrightarrow$	`(d(a(an)))`
VP		`(v(d(an)))`
VP		`(v(p(dn)))`
PP		`(p(d(an)))`
AP		`(a(a(an)))`
S	$\Longrightarrow$	`((d(an))v)`
NP		`(d(a(a(an))))`
VP	$\longrightarrow$	`(v(d(a(an))))`
VP	$\longrightarrow$	`(v(p(d(an))))`
PP	$\longrightarrow$	`(p(d(a(an))))`
AP	$\longrightarrow$	`(a(a(a(an))))`
S	$\Longrightarrow$	`((dn)(v(dn)))`
NP		`((dn)(p(dn)))`
S		`((dn)(v(d(an))))`
S	$\longrightarrow$	`((dn)(v(p(dn))))`
NP	$\longrightarrow$	`((dn)(p(d(an))))`
S	$\longrightarrow$	`((d(an))(v(dn)))`
NP	$\Longrightarrow$	`((d(an))(p(dn)))`
S	$\longrightarrow$	`((dn)(v(d(a(an)))))`
S	$\longrightarrow$	`((dn)(v(p(d(an)))))`
S	$\longrightarrow$	`((d(an))(v(d(an))))`
S		`((d(an))(v(p(dn))))`
NP	$\Longrightarrow$	`((d(an))(p(d(an))))`
S	$\longrightarrow$	`((d(an))(v(p(d(an)))))`

Tabuľka 3.3: Gramatické frázy, ktoré natrénovaná sieť z *najlepšej* simulácie vedela reprezentovať pri prahu rekonštrukcie $\theta = 0.9$. Vľavo je typ gramatickej frázy (porovnaj s tabuľkou 3.1), vpravo je príslušná štruktúra. Jednoduchá šípka označuje štruktúry, ktoré sa nevyskytli v rámci trénovacej množiny. Dvojitá šípka označuje štruktúry, ktoré neboli reprezentovateľné sieťou z *najhoršej* simulácie.

3.4 Zhrnutie

V reprodukovanom experimente sa nám podarilo dosiahnuť približne rovnaké výsledky, ako uvádzajú Voegtlin & Dominey [2]. Počet štruktúr, ktoré bola lineárna RAAM schopná po natrénovaní reprezentovať, bol v priemere o dva rády väčší ako veľkosť trénovacej množiny.

Oproti [2] sme používali menšiu rýchlosť učenia, čím sme dosiahli stabilizáciu váh po určitom počte iterácií. Akonáhle sa váhy prestanú výrazne meniť, prestanú sa meniť aj vlastnosti siete, t.j. ustáli sa chyba rekonštrukcie a počet reprezentovateľných štruktúr. V čase, keď dôjde k ustáleniu váh, možno trénovanie ukončiť.

Výsledné počty reprezentovateľných štruktúr sa v jednotlivých simuláciach veľmi líšili. Z výraznej štandardnej odchýlky v grafe 3.4a možno usúdiť, že s niečim podobným sa stretli aj V&D. Zistili sme však, že spomedzi všetkých reprezentovateľných štruktúr je iba málo vygenerovateľných z gramatiky 3.1. Z hľadiska počtu gramaticky správnych reprezentovateľných štruktúr sa dá povedať, že lineárna RAAM má rovnakú kapacitu ako klasická RAAM.

Ako sme pozorovali, pre rôzne inicializácie váh sú rôzne aj výsledné matice $\mathbf{W}$. Procedúra generalizácie, pomocou ktorej sme merali kapacitu siete, nerozlišuje medzi gramatickými a negramatickými štruktúrami. Tak sa môže stať, že raz sa sieti podarí reprezentovať viac negramatických štruktúr a inokedy zas menej. Preto stojí za zváženie, či je procedúra generalizácie vhodným prostriedkom na testovanie kapacity siete. Relevantnejšie, a tiež na počiatočné podmienky menej citlivé výsledky by sme zrejme dosiahli testovaním počtu gramaticky správnych reprezentovateľných štruktúr.

Kapitola 4

Vplyv kódovania terminálov

Na priebeh trénovania a na vlastnosti natrénovaného modelu lineárnej RAAM vplýva viacero parametrov. V predchádzajúcej kapitole sme na jednoduchých simuláciach ukázali, že s príliš veľkou rýchlosťou učenia η sa váhy nedokážu dostať do rovnovážneho stavu. Ďalej z našich, tak ako aj z výsledkov publikovaných v práci [2] vyplýva, že príliš prísna alebo príliš voľná voľba prahu rekonštrukcie θ znižuje kapacitu modelu.

Schopnosti modelu ovplyvňuje aj spôsob reprezentovania terminálov. Poukázali na to Voegtlin & Dominey [2, časť 6] pri rekonštrukcii iného Pollackovho experimentu. Lineárnu RAAM trénovali na štrnástich ternárnych štruktúrach, ktoré vznikli transformáciou viet prirodzeného jazyka. Celkovo sa v rámci štruktúr trénovacej množiny nachádzalo 23 terminálnych symbolov. Pri trénovaní siete použili najprv neutrálny kód a v ďalšej simulácii potom Pollackom vytvorené reprezentácie obsahujúce informáciu o príslušnosti k danej kategórii. Ukázalo sa, že v druhom prípade dokázala sieť reprezentovať viac významovo korektných fráz, ako pri použití neutrálneho kódu.

Neutrálne kódovanie iba odlišuje od seba jednotlivé symboly. Euklidovská vzdialenosť ľubovoľných dvoch reprezentácií je rovná $\sqrt{2}$. Tento typ reprezentácií teda sieti neposkytuje žiadnu dodatočnú informáciu o vlastnostiach a význame jednotlivých symbolov. Pri druhom type kódovania mali všetky slová patriace do jednej triedy nastavený určitý bit a medzi sebou sa líšili hodnotami na iných bitoch. Napríklad všetky „veci“ mali nastavený prvý bit a medzi sebou sa líšili hodnotami na pozíciach 2 až 4. Navyše, každé slovo, ktoré nebolo „vec“, malo hotnoty na prvých štyroch pozíciach nulové. Takto boli reprezentácie slov patriacich do jednej skupiny topologicky bližšie ako vektory slov z rôznych kategórií. Hadley [16] tvrdí, že sémanticky

konzistentné zovšeobecňovanie môže vyplývať najmä z tejto vlastnosti.

Keďže neutrálne kódovanie pravdepodobne nezodpovedá spôsobu, akým sa reprezentujú informácie v mozgu, má význam skúsiť použiť kódovanie, ktoré nejakým spôsobom obnáša dodatočné informácie, a skúmať jeho vplyv na zmysluplné zovšeobecňovanie.

V tejto kapitole opíšeme experiment, na základe ktorého sme porovnávali celkom tri rôzne spôsoby reprezentovania symbolov. Najprv povieme, ako sme pretransformovali umelo vygenerované vety do ternárnych štruktúr, čím sme dostali trénovaciu množinu. Potom charakterizujeme použité typy reprezentácií terminálov. Na záver opíšeme samotný experiment a zhodnotíme dosiahnuté výsledky.

4.1 Príprava trénovacej množiny

Príprava trénovacej množiny pre náš experiment pozostávala z dvoch krokov. Najprv sme pomocou jednoduchého generátora jazyka (*simple language generator*, SLG [15]) vygenerovali veľké množstvo anglických viet. Niektoré z nich sme potom transformovali do ternárnych štruktúr podobným spôsobom ako v úvode spomenutom pokuse. Tieto štruktúry potom boli použité na trénovanie.

4.1.1 Vety vytvorené pomocou SLG

Ak každému pravidlu odvodenia bezkontextovej gramatiky priradíme pravdepodobnosť, s akou má byť vybrané z pomedzi ostatných použiteľných pravidiel, dostaneme tzv. *stochastickú bezkontextovú gramatiku*. Tento typ *generatívnej* gramatiky umožňuje kontrolovať distribúciu vyprodukovaných viet a aj preto je častou voľbou pre generovanie viet pripomínajúcich prirodzený jazyk. Avšak v prirodzenom jazyku sa vyskytujú sémantické obmedzenia a závislosti, ktoré vylučujú niektoré kombinácie slov, resp. iné kombinácie preferujú pred ostatnými. Už na vygenerovanie pomerne jednoduchých množín viet môže byť priveľmi náročné opísať takéto vzťahy priamo pravidlami gramatiky. Formát vstupu pre SLG [15] umožňuje používateľovi špecifikovať základnú syntax jazyka v podobe bezkontextových pravidiel a definovať dodatočné obmedzenia v prirodzenej podobe. SLG vstup najprv transformuje do podoby stochastickej bezkontextovej gramatiky a tú potom použije na generovanie viet.

Podstatné mená		Slovesá		Zámená		Vlastné
singulár	plurál	základ	3. os.	singulár	plurál	mená
boy	boys	bark	barks	who	who_pl	John
bread		bite	bites			Kate
cat	cats	chase	chases			Mary
dog	dogs	eat	eats			Steve
fish		feed	feeds			
girl	girls	hate	hates			
ham		hear	hears			
man	men	like	likes			
meat		run	runs			
woman	women	see	sees			
		sing	sings			
		swim	swims			
		talk	talks			
		walk	walks			

Tabuľka 4.1: Slová vyskytujúce sa vo vygenerovaných vetách.

Pre nasledujúce simulácie sme pomocou SLG vytvorili 10000 anglických oznamovacích viet zložených zo slovníka 50 slov. Slovník pozostával z podstatných mien v jednotnom a množnom čísle, zo slovies v neurčitku a v tretej osobe jednotného čísla, z vlastných mien a zo zámien `who` a `who_pl`. Všetky slová sú vymenované v tabuľke 4.1.

Vygenerované vety mali nasledujúci tvar:

PODMET (rozvitý) PRÍSUDOK PREDMET (rozvitý)

Predmetová časť nebola zastúpená v každej vete. Niektoré slovesá to totiž nevyžadovali (napr. `eat`, jesť) a naopak, niektoré predmetovú časť bez použitia predložiek neumožňovali (napríklad `bark`, štekať).

Jednoduché vety mali teda dve alebo tri slová. Zložené vety mali podmetovú a/alebo predmetovú časť rozvitú vedľajšou vetou prívlastkovou začínajúcou zámenom `who` (ktorý, -á, -ého, -ú), resp. `who_pl` (ktorí, -é, -ých), podľa toho, či bolo rozvíjané podstatné meno v jednotnom alebo množnom čísle. Vedlajšia veta takisto mohla obsahovať podstatné meno vo funkcii predmetu, ten mohol byť rovnako rozvitý, atď. Avšak pravdepodobnosť generovania ďalšej vedľajšej vety klesala geometrickým radom, čím bolo znemožnené

a) Steve walks
b) women see boys
c) dogs who_pl see girl bark
d) boy feeds cat who John sees
e) dog who boy sees sees cat who chases cats
f) boy who girl likes walks dogs who_pl see John who hates cat
g) cat who sees boys who_pl like girl who runs walks

Tabuľka 4.2: Príklady vygenerovaných viet. a) Jednoduchá veta bez predmetu, b) s predmetom. Zložená veta s rozvitým c) podmetom, d) predmetom, e) s podmetom aj predmetom. f) Veta s dvojnásobne rozvitým predmetom, g) veta s trojnásobne rozvitým podmetom.

generovanie príliš dlhých viet. Tabuľka 4.2 uvádza príklady získaných viet.

4.1.2 Transformácie viet do štruktúr

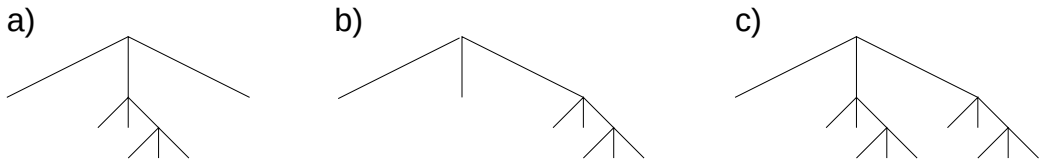
Z vygenerovaného množstva viet sme vybrali 100 tak, aby bolo zastúpené každé slovo aspoň raz. Vybrané vety sme pretransformovali do ternárnych štruktúr nasledovným spôsobom. Podobne ako Pollack v [7, odsek 3.2.2], snažili sme sa prepísať vetné členy tak, aby bolo rekurzívne zachované poradie (ACTION AGENT OBJECT), pričom tieto kategórie zodpovedajú postupne prísudku, podmetu a predmetu.

Transformácia jednoduchej vety je priamočiara. Stačí zameniť poradie podmetu a prísudku a pridať zátvorky. Malý problém je s vetami, v ktorých sa nenachádza predmet. Vyriešime to pridaním nového terminálneho symbolu `NULL`. Tento terminál budeme vždy reprezentovať nulovým vektorom. Ak teda vo vete chýba predmetová časť, tretia skupina vstupných neurónov nebude vykazovať žiadnu aktivitu.

Štruktúry prislúchajúce zloženým vetám budú mať nasledujúci tvar:

$$(\text{ PRÍSUDOK } \; R_{podmet} \quad R_{predmet} \;)$$

kde R_{podmet} znamená buď samotný podmet, ak nie je rozvitý, alebo štruktúru, ktorú dostaneme spracovaním podmetovej časti vety. Obdobné platí aj pre $R_{predmet}$, s tým rozdielom, že ak vo vete predmet chýba, $R_{predmet}$ bude znamenať špeciálny symbol `NULL`. Prísudok v hlavnej vete nebol nikdy rozvíjaný, preto ho stačí umiestniť na prvú pozíciu.



Obr. 4.1: Charakter štruktúr, ktoré vznikli transformáciou zložených viet. Stromy vznikli transformáciou vety a) s rozvitým podmetom, b) predmetom, c) podmetom aj predmetom.

Tým pádom nám teraz stačí povedať, ako pretransformujeme rozvitý podmet alebo predmet do požadovanej štruktúry. V oboch prípadoch ide o rovnaký algoritmus. Podobne ako Pollack [7, odsek 3.2.2] zavedieme dva nové terminálne symboly `is` a `are`. Ako príklad rozvitého podmetu/premetu použijeme časť vety „John who sees Mary“. Priamočiary preklad do slovenčiny je „Janko, ktorý vidí Marienku“. Avšak preklad „Janko vidiaci Marienku“ lepšie ukazuje, že Jankovi je takto priradená vlastnosť „vidí Marienku“. Práve takýto preklad nám pomôže pochopiť transformáciu tejto časti vety do ternárnej štruktúry prevedenú nasledovne:

(is John (sees John Mary))

Zvyšok opisu tvoria už iba detaily. Ak rozvíjame podstatné meno v množnom čísle, použijeme namiesto `is` slovo `are`. Ak je predmet vo vedľajšej vete rozvitý, transformujeme ho rekurzívne. Medzi vygenerovanými vetami sa nachádzajú ešte spojenia typu „John who Mary sees“, v preklade „Janko videný Marienkou“. V tomto prípade vystupuje `Mary` vo vedľajšej vete vo funkcii podmetu. Preto nás neprekvapí zodpovedajúca štruktúra

(is John (sees Mary John))

Nakoniec pripomíname, že v takto vytvorených ternárnych stromoch sa nebudú nachádzať symboly `who` a `who_pl`.

Stromy, ktoré vzniknú vyššie opísaným spôsobom, nemajú príliš rôznorodý tvar. Ľavý podstrom má zakaždým podobu listu. Stredný a pravý podstrom zodpovedajú rekurzívnej transformácii rozvitého podmetu, resp. predmetu a môžu byť ďalej rozbaľované iba cez pravých synov, nakoľko vo vedľajšej vete môže byť rozvitý iba predmet. Situáciu dokumentuje obrázok 4.1.

Detailne sme opísali postup, akým sme prekladali pôvodné vety do ternárnych stromov. Uveďme ešte praktické ukážky. Štruktúry zodpovedajúce

a) **(** walks Steve NULL **)**
b) **(** see women boys **)**
c) **(** bark (are dogs (see dogs girl)) NULL **)**
d) **(** feeds boy (is cat (sees John cat)) **)**
e) **(** sees (is dog (sees boy dog)) (is cat (chases cat cats)) **)**
f) **(** walks (is boy (likes girl boy)) ...
... (are dogs (see dogs (is John (hates John cat)))) **)**
g) **(** walks (is cat (sees cat (are boys (like boys ...
... (is girl (runs girl NULL)))))) NULL **)**

Tabuľka 4.3: Vety z tabuľky 4.2 preložené do podoby ternárnych stromov. Keď bol v pôvodnej vete n-násobne rozvitý podmet alebo predmet, hĺbka štruktúry bude $2n + 1$.

vetám z tabuľky 4.2 sú uvedené v tabuľke 4.3. Dlhšie z nich sú rozdelené do dvoch riadkov.

4.2 Reprezentácie terminálov

Cieľom experimentu opísaného v tejto kapitole je preskúmať vplyv rozličných spôsobov reprezentovania terminálnych symbolov. Celkovo budeme porovnávať tri typy kódovania. Prvý, najjednoduchší spôsob reprezentácií je neutrálny kód, ktorý iba od seba odlišuje jednotlivé symboly. Druhý typ predstavujú reprezentácie vytvorené modelom WCD (word co-occurrence detector, [9, 10, 11]), ktoré charakterizujú kontext, v akom sa jednotlivé slová vyskytujú. Tretí, posledný typ reprezentácií je získaný z databázy WordNet [14] extrakciou charakteristických príznakov pomocou procedúr, ktoré vytvoril Harm [13].

Skôr, než opíšeme jednotlivé spôsoby reprezentovania terminálov, zamyslíme sa nad optimálnou hodnotou prahu rekonštrukcie θ (optimálnou z hľadiska počtu reprezentovateľných štruktúr). Z predošlej kapitoly vieme, že ak je hodnota θ príliš malá, kapacita siete bude tiež malá. Na základe nasledujúcich úvah budeme môcť približne určiť optimálny prah rekonštrukcie iba na základe zvolených reprezentácií terminálov.

4.2.1 Optimálna hodnota prahu rekonštrukcie

Terminálne symboly sú reprezentované reálnymi vektormi pevne zvolenej dimenzie k. Takto sa môžeme na reprezentácie dívať ako na body k-rozmerného euklidovského priestoru. Pri dekódovaní redukovanej reprezentácie štruktúry je dekódovaný vektor porovnávaný s jednotlivými vektormi reprezentujúcimi terminály. Ak je dekódovaný vektor vzdialený od reprezentácie terminálu o menej ako θ (prah rekonštrukcie), je to ekvivalentné tvrdeniu, že dekódovaný vektor sa nachádza vnútri k-rozmernej sféry so stredom v bode, ktorý zodpovedá reprezentácii terminálu, a s polomerom θ. Nejednoznačnosť pri dekódovaní nastáva práve vtedy, keď dekódovaný vektor patrí do oblasti prieniku viacerých k-rozmerných sfér. A nakoniec, dekódovaný vektor je dekóderom považovaný za reprezentáciu štruktúry, ak leží mimo oblastí týchto sfér.

Nech d_{min} označuje euklidovskú vzdialenosť najbližších dvoch terminálov. Zrejme ak zvolíme $\theta = d_{min}/2$, k nejednoznačnému dekódovaniu určite nedôjde, lebo prienik sfér prislúchajúcich terminálnym reprezentáciam bude prázdny. Avšak to neznamená, že s touto hodnotou prahu rekonštrukcie bude sieť vedieť reprezentovať maximálny možný počet štruktúr. Napríklad, v Pollackovom experimente (pozri kapitolu 3) bolo $d_{min}/2 \approx 0.7$, kým graf 3.1 potvrdzuje, že táto hodnota θ nie je *optimálna* z hľadiska počtu reprezentovateľných štruktúr.

Objem časti k-rozmernej sféry, ktorá nepatrí do prieniku s inými sférami, označme *čistý objem*. Čím je pre daný terminál čistý objem príslušnej k-rozmernej sféry väčší, tým väčšia je šanca, že tento terminál bude správne rozpoznaný. Avšak to nie je jediná podmienka, ktorú treba zohľadniť pri odhadovaní optimálnej hodnoty prahu rekonštrukcie. Totiž pravidlo modifikácie váh 2.9 zabezpečuje, že rekonštruované reprezentácie jednotlivých terminálov sú z hľadiska euklidovskej vzdialenosti blízko pôvodných vektorov. Preto určitá oblasť okolo reprezentácie každého terminálu nesmie patriť do prieniku viacerých sfér.

Zo skúsenosti odhadujeme, že optimálna hodnota prahu rekonštrukcie θ_{opt} bude z intervalu $(d_{min}/2,\ d_{min})$. Pre dané reprezentácie terminálov a pre danú trénovaciu množinu môžeme θ_{opt} presnejšie určiť experimentálne po skončení trénovania (pozri graf 3.1 a tiež graf 4.6).

4.2.2 Neutrálny kód

Každému terminálu je priradený vektor, ktorý má na jednom, pre každý symbol zvláštnom mieste nastavený bit na 1 a na ostatných miestach je nula. Vety z trénovacej množiny pre tento experiment obsahujú celkom 50 slov. To znamená, že prvých 50 miest vo vektoroch reprezentácií je rezervovaných na odlíšenie jednotlivých terminálov, kým zvyšné miesta (s nulovou hodnotou) poskytujú sieti väčšiu voľnosť. Špeciálny terminál `NULL` budeme reprezentovat nulovým vektorom.

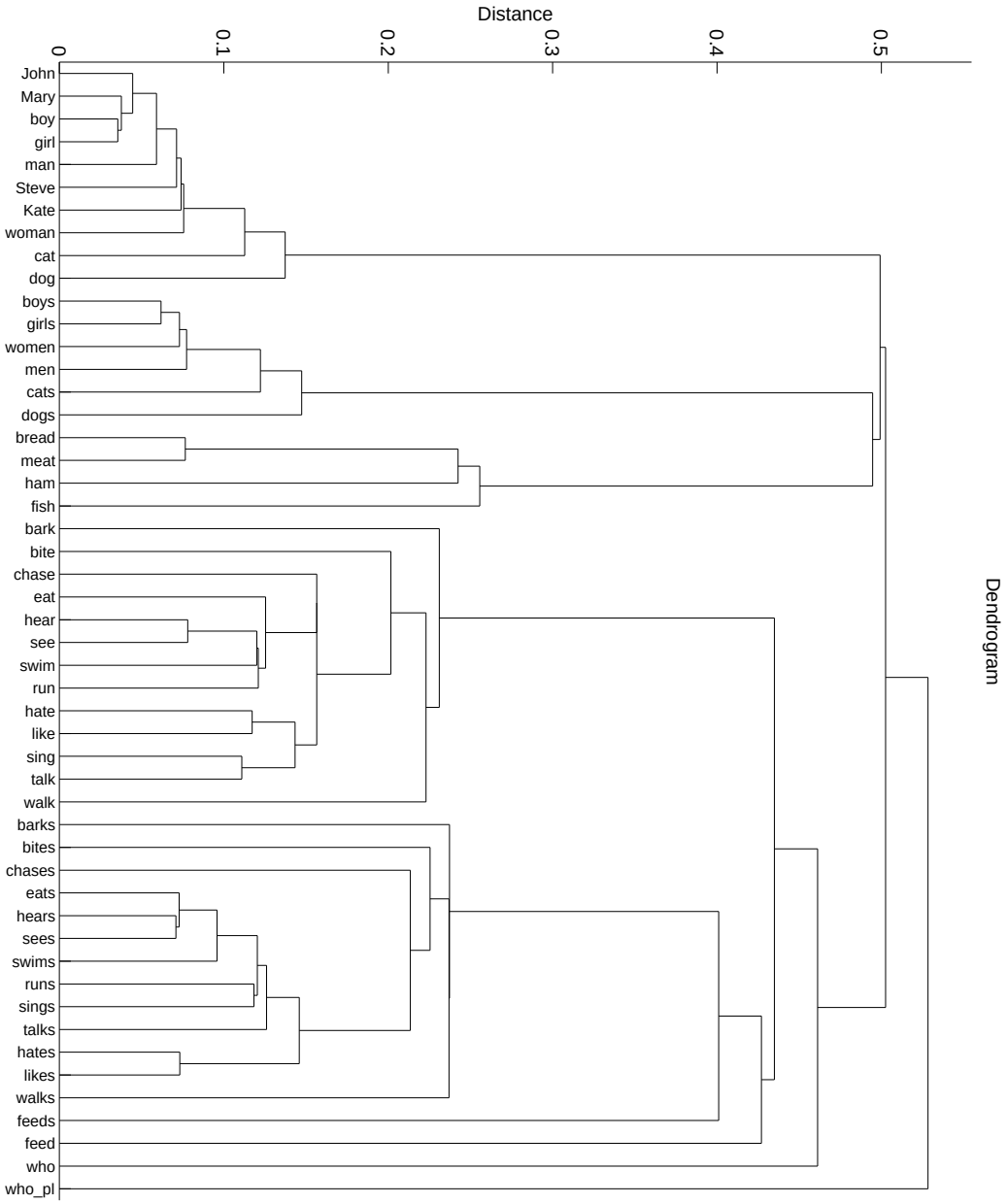
Vzdialenosť každých dvoch z 50 terminálov je $\sqrt{2}$, avšak vzdialenosť každého slova od terminálu `NULL` je 1. Preto optimálna hodnota prahu rekonštrukcie pre tento typ kódovania bude $\theta_{opt} > 0.5$.

4.2.3 Reprezentácie vygenerované modelom WCD

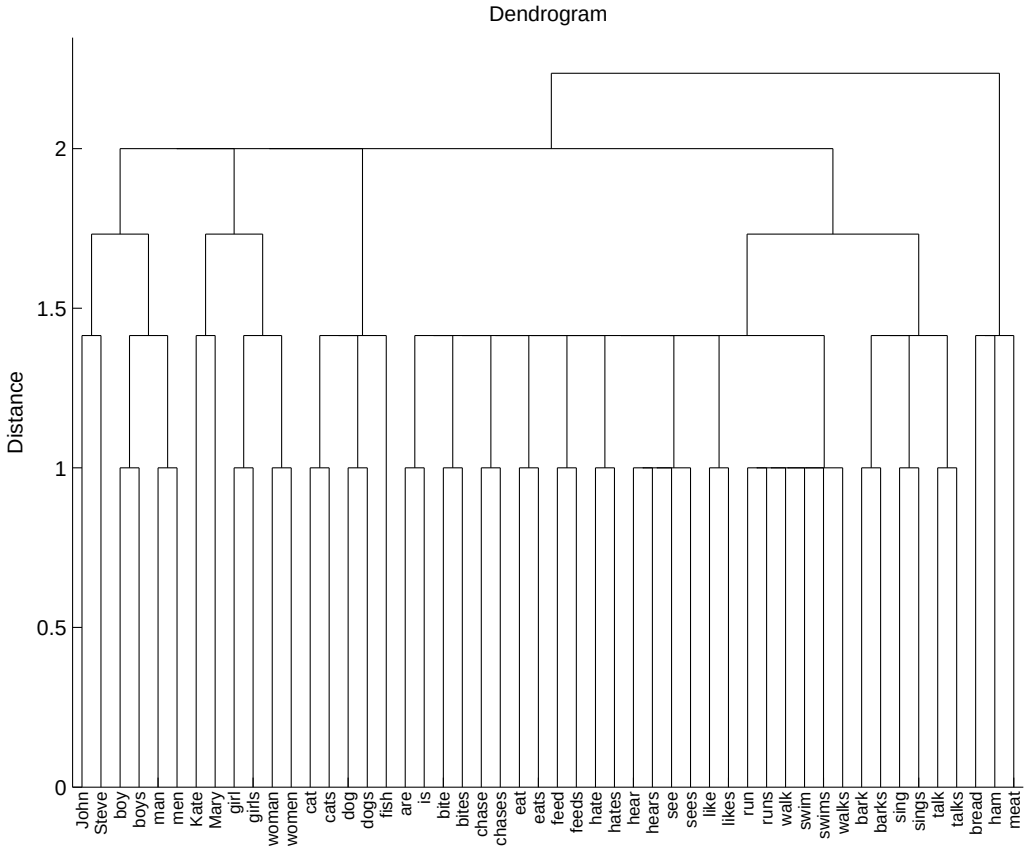
Človek približne vie pochopiť význam (nového) slova iba podľa jeho pozície v texte, lebo zo skúsenosti vie, že v danom kontexte sa môžu vyskytovať iba slová s určitým významom. Model rekurentnej neurónovej siete *Word co-occurrence detector* [9, 10, 11] je schopný vytvoriť podobné reprezentácie pre slová, ktoré sa v texte vyskytujú v podobnom kontexte. Sieti je predložená postupnosť slov (za sebou zoradené vety) na spracovanie. Váhy po adaptovaní reprezentujú pravdepodobnosti vzájomného výskytu každých dvoch slov zo slovníka. Nech N je počet slov v slovníku. Matica váh po adaptovaní siete predstavuje tabuľku pravdepodobností vzájomných výskytov slov. Reprezentáciu i-teho slova tvorí vektor dimenzie $2N$, ktorý vznikne zložením i-teho riadku a i-teho stĺpca tejto tabuľky. Jedna časť reprezentácie predstavuje pravý kontext daného slova a druhá časť reprezentácie ľavý kontext.

Na obrázku 4.2 je zobrazený dendrogram *WCD reprezentácií* 50 slov. Vstup pre WCD model tvorilo 10000 viet vygenerovaných pomocou SLG. Pre symboly `is` a `are` použijeme vygenerované WCD reprezentácie slov `who` resp. `who_pl`.

Z dendrogramu vidno, že minimálna vzdialenosť dvoch slov je $d_{min} < 0.05$. Pre lepšie trénovanie sme všetky vektory naškálovali na maximálnu dĺžku tak, aby na žiadnej pozícii vo vektoroch reprezentácií nebola hodnota väčšia ako 1. Keďže najväčšia hodnota na pozíciach bola približne 0.5, naškálované vektory mali približne dvojnásobnú dĺžku. Preto odhadujeme, že pre optimálnu hodnotu prahu rekonštrukcie pre WCD reprezentácie bude platiť $\theta_{opt} \in (0.05, 0.1)$.



Obr. 4.2: Dendrogram pre WCD reprezentácie. Veľmi výrazné skupiny tvoria životné podstatné mená v jednotnom a v množnom čísle, neživotné podstatné mená (*bread*, *meat*, *ham*, *fish*), slovesá v neurčitku a slovesá v tretej osobe singuláru. Sloveso *feed* netvorí skupinu so žiadnymi inými slovesami.



Obr. 4.3: Dendrogram pre WordNet reprezentácie. Vľavo je vidieť výrazné skupiny osôb mužského pohlavia, osôb ženského pohlavia a zvierat. Veľkú skupinu tvoria slovesá, v rámci ktorých si môžeme všimnúť kategórie slovies súvisiacich s vnímaním (*hear*, *see*), pohybom (*run*, *walk*, *swim*) a rečou (*bark*, *sing*, *talk*). Poslednú skupinu tvoria jedlá.

4.2.4 Reprezentácie odvodené z databázy WordNet

WordNet reprezentácie boli odvodené pomocou procedúr (Harm, [13]), ktoré vygenerujú pre každé slovo zo slovníka na vstupe množinu príznakov, pomocou ktorých sa dajú jednotlivé slová od seba odlíšiť. Procedúry pritom využívajú najmä údaje z databázy WordNet [14], v ktorej sú anglické slová klasifikované na základe synoným, antoným a hierarchických vzťahov.

Keď sme na vstup Harmových procedúr dali slovník všetkých 50 slov, na výstupe sa nachádzali aj reprezentácie slov, ktoré sa na vstupe nevyskytovali. Čo bolo horšie, pre niektoré slová (napr. vlastné mená) reprezentácie

chýbali. Tento problém sme vyriešili nasledovne. Na vstup procedúr sme dali iba podstatné mená v jednotnom čísle bez vlastných mien a slovesá v neurčitku. Výstupom boli príznaky pre všetky slová na vstupe plus pre nejaké slová navyše. Tieto prebytočné údaje sme odstránili, čím nám zostali iba charakteristiky slov na vstupe.

Následne bolo potrebné vyrobiť príznaky pre chýbajúce slová, ktoré sme nedali na vstup. Príznaky pre sloveso v tretej osobe jednotného čísla sme odvodili od vygenerovaných príznakov pre príslušné sloveso v neurčitku pridaním vlastnosti `third_person_morph`. Príznaky pre podstatné meno v množnom čísle sme odvodili od vygenerovaných príznakov pre príslušné podstatné meno v jednotnom čísle pridaním vlastnosti `plural_morph`.

Doterajšie ručné úpravy boli prevedené rovnako, ako by ich vygenerovali skripty. V prípade vlastných mien sme sa však nemohli oprieť o žiaden príklad, preto sme sa rozhodli nasledovne. Príznaky pre vlastné mená sme odvodili od vygenerovaných príznakov pre slová `man`, resp. `woman`. Pridali sme spoločnú vlastnosť `has_name` a originálny príznak odlišujúci každé meno od ostatných.

Celkovo bolo pre 50 slov použitých 41 rôznych príznakov. Pre každé slovo bolo použitých 1 až 6 príznakov (samozrejme, okrem slova `NULL`, ktoré opäť budeme reprezentovať nulovým vektorom). Dendrogram pre tento typ reprezentácií je na obrázku 4.3. Minimálna euklidovská vzdialenosť reprezentácií dvoch slov je v tomto prípade $d_{min} = 1$, preto optimálna hodnota prahu rekonštrukcie bude $\theta_{opt} > 0.5$.

4.3 Simulácie

Dimenzia vektorov vygenerovaných modelom WCD je rovná dvojnásobku veľkosti použitého slovníka, čo je v našom prípade 100. Na neutrálny kód a na WordNet reprezentácie stačia 50, resp. 41 rozmerné vektory. Teda dimenzia siete musí byť aspoň 100. My použijeme práve túto dimenziu, $k = 100$. Takto pri použití WCD reprezentácií nedávame sieti žiadnu voľnosť v podobe prázdnych bitov.

Trénovacia množina pozostáva tiež zo 100 štruktúr, ktoré sme dostali transformáciou vygenerovaných viet. Spolu s podštruktúrami sa v rámci trénovacej množiny nachádzalo celkom 325 rôznych objektov.

Pre trénovanie potrebujeme ešte stanoviť rýchlosť učenia a počiatočné hodnoty váh. Keď sme váhy inicializovali tak, ako pri rekonštruovanom Pol-

lackovom experimente (kapitola 3), čiže náhodne z intervalu $[-0.5; 0.5]$, mali sme veľký problém úspešne trénovať sieť. Za daných okolností bolo totiž nutné zvoliť η rádovo 10^{-6} až 10^{-7}. Pre väčšie rýchlosti jednotlivé váhy divergovali do $\pm\infty$.

Na druhej strane, pre tak malú rýchlosť učenia bolo potrebných 200000 iterácií učenia, aby sieť pri neutrálnom kódovaní vedela reprezentovať 20% štruktúr z trénovacej množiny. Aj po toľkých iteráciach celková chyba rekonštrukcie stále klesala, takže trénovanie nebolo u konca. Navyše, vzhľadom na veľkú trénovaciu množinu a veľký počet váh (30000) trvala simulácia veľmi dlho (na počítači s 2.4 GHz procesorom niekoľko dní).

V nasledujúcej časti analyzujeme tieto problémy a určíme vhodnú rýchlosti učenia a interval, z ktorého budú inicializované váhy. Potom zhodnotíme výsledky simulácií s týmito hodnotami parametrov.

4.3.1 Rýchlosť učenia a inicializácia váh

Všimnime si pravidlo pre modifikáciu váh 2.9

$$\Delta w_{ij}^{(a)} = \eta\, c_i \left(z_j^{(a)} - \sum_{r=1}^{k} w_{rj}^{(a)} c_r \right)$$

Vidíme, že zmena váhy závisí od rýchlosti učenia η, od aktivity c_i neurónu na strednej vrstve, od aktivity $z_j^{(a)}$ neurónu na vstupnej vrstve a nakoniec od váhovanej sumy aktivít neurónov strednej vrstvy. Navyše c_i je len váhovaný súčet aktivít na vstupnej vrstve, takže zmena váhy závisí iba od rýchlosti učenia, od vstupu a od váh siete. Teraz spravíme odhad, ako sa zmenia váhy prvý raz po inicializácii.

Na začiatku procesu učenia sa môžu byť sieti predložené iba podštruktúry zložené výlučne z terminálnych symbolov. Nech $\mathbf{z}$ je vektor aktivít, ktorý sa ako prvý objaví na vstupe. Jeho distribúcia závisí iba od štruktúr v trénovacej množine a od reprezentácií terminálov. Zachovávajúc zavedené označenia z kapitoly 2, súčet zložiek vektora $\mathbf{z}$ je

$$s(\mathbf{z}) = \sum_{a=1}^{n} \sum_{j=1}^{k} z_j^{(a)} \tag{4.1}$$

Predpokladáme, že váhy boli inicializované náhodne z intervalu $[-\omega; \omega]$. Aktivita c_i neurónu na strednej vrstve je daná váhovaným súčtom aktivít na vstupnej vrstve. Môžeme teda urobiť nasledovný odhad

$$-\omega s(\mathbf{z}) \leq c_i \leq \omega s(\mathbf{z}) \tag{4.2}$$

Nech je dimenzia siete rovná k. Potom platí:

$$-\omega^2 s(\mathbf{z}) \leq w_{rj}^{(a)} c_r \leq \omega^2 s(\mathbf{z}) \tag{4.3}$$

$$-k\omega^2 s(\mathbf{z}) \leq \sum_{r=1}^{k} w_{rj}^{(a)} c_r \leq k\omega^2 s(\mathbf{z}) \tag{4.4}$$

Vďaka symetrii vzťahu 4.4 platí rovnaký odhad aj pre $(-\sum_{r=1}^{k} w_{rj}^{(a)} c_r)$. Ak je hodnota k dostatočne veľká, môžeme vo výraze $\left(z_j^{(a)} - \sum_{r=1}^{k} w_{rj}^{(a)} c_r\right)$ zanedbať člen $z_j^{(a)} \in [0;1]$. Dostávame konečný odhad

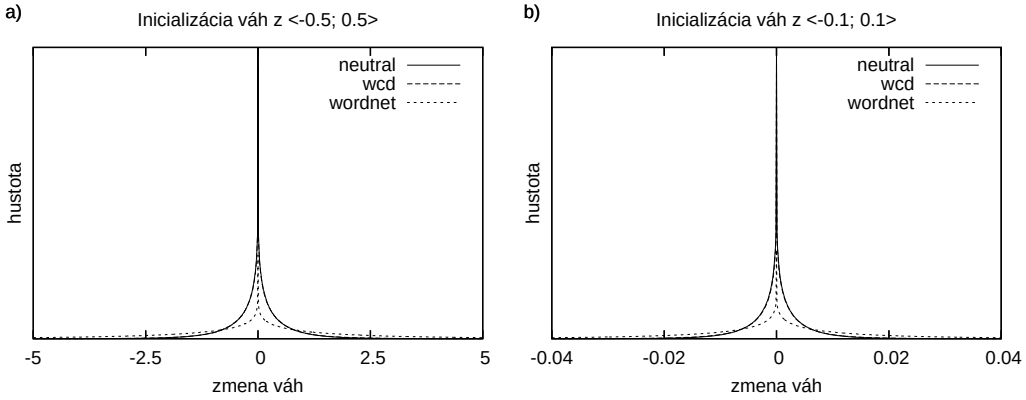
$$-\alpha\eta\omega^3 \leq \Delta w_{ij}^{(a)} \leq \alpha\eta\omega^3 \tag{4.5}$$

kde $\alpha = k \cdot (s(\mathbf{z}))^2$ závisí od dimenzie siete, od trénovacej množiny a od reprezentácií terminálov, čo v našom prípade máme pevne dané. Preto zmena váh na začiatku trénovania závisí iba lineárne od rýchlosti učenia a kubicky od hraničných hodnôt váh.

Podotýkame, že odhad 4.5 nezávisí od a, i, ani od j. Inými slovami, na zmeny váh sa dá dívať ako na výsledky náhodnej premennej X s nulovou strednou hodnotou. Samozrejme, X nebude mať rovnomerné rozdelenie. Funkciu hustoty náhodnej premennej X však môžeme získať experimentálne. Z trénovacej množiny získanej preložením 100 viet sme vybrali všetkých 229 podštruktúr hĺbky 1. Príslušné vektory aktivít sú práve tie, ktoré sa môžu objaviť na vstupe ako prvé. Pre každú z týchto podštruktúr sme vypočítali zmeny každej z 30000 váh siete (ternárna lineárna RAAM dimenzie $k = 100$). Váhy boli pre porovnanie inicializované najprv z intervalu $[-0.5; 0.5]$ a potom z intervalu $[-0.1; 0.1]$. Grafy na obrázku 4.4 zobrazujú funkciu hustoty premennej X pre oba prípady. Rýchlosť učenia sme zvolili $\eta = 1$, aby boli rozdiely medzi oboma prípadmi zreteľné.

Teraz máme odhad potvrdený aj experimentálne. Vidíme, že keď zmenšíme 5-násobne veľkosť intervalu, z ktorého sú inicializované hodnoty váh, zmena váh sa zmenší 125-násobne. Rovnako tak ale môžeme 125-násobne zväčšiť rýchlosť učenia pri zachovaní veľkosti zmien váh[1].

[1]Nesmieme však zabúdať, že rýchlosť učenia nesmie byť ani príliš veľká, ináč by sa váhy neustálili. Pozri tiež časť 3.3.1.



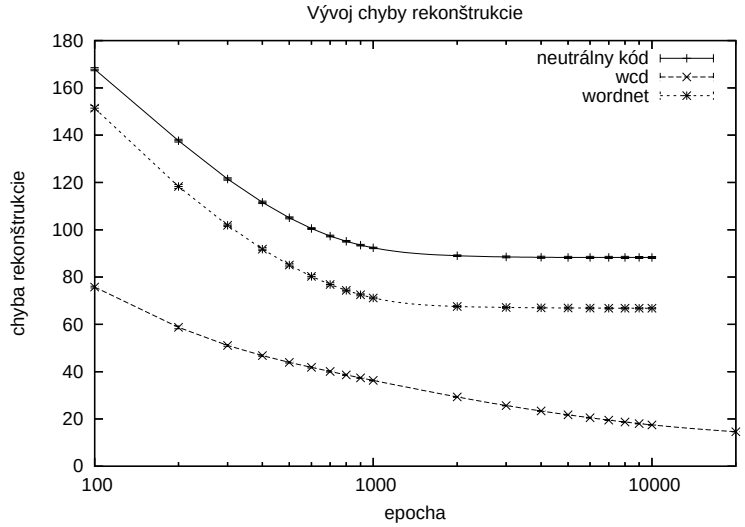
Obr. 4.4: Počiatočné zmeny váh pre trénovaciu množinu, ktorú tvorí 100 viet pretransformovaných do ternárnych štruktúr. Rýchlosť učenia sme zvolili $\eta = 1$, aby boli zreteľné rozdiely medzi oboma prípadmi. Najväčšiu disperziu maju zmeny váh pri použití WordNet reprezentácií.

V nasledujúcom experimente, v ktorom budeme porovnávať vplyv kódovania terminálov, budeme váhy inicializovať z intervalu $[-0.1; 0.1]$. Rýchlosť učenia bude rovná $\eta = 0.001$. Teda počiatočné zmeny váh sa budú na začiatku pohybovať v rozmedzí $\pm 10^{-5}$ (porovnaj s grafom 4.4b).

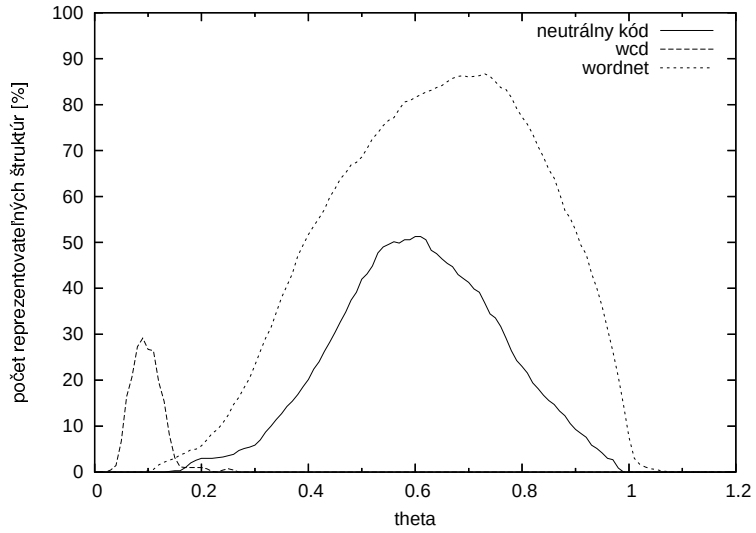
4.3.2 Výsledky simulácií

Pre každý typ kódovania bolo spustených 7 simulácií s náhodne inicializovanými váhami rovnomerne z intervalu $[-0.1; 0.1]$. Rýchlosť učenia bola vždy $\eta = 0.001$. V prípadoch, keď bolo použité neutrálne kódovanie a WordNet reprezentácie, celková chyba rekonštrukcie pre trénovaciu množinu prestala výrazne klesať po približne 2000 epochách učenia. Tieto simulácie sme nechali bežať 10000 iterácií. Pri použití WCD reprezentácií sme trénovanie zastavili po 20000 epochách učenia. Ešte aj vtedy chyba rekonštrukcie klesala. Avšak výsledky naznačovali, že trénovanie sa pomaly blíži ku koncu a že v krátkom čase by sa váhy ustálili. Priemerná hodnota celkovej chyby rekonštrukcie štruktúr trénovacej množiny je zobrazená v grafe 4.5.

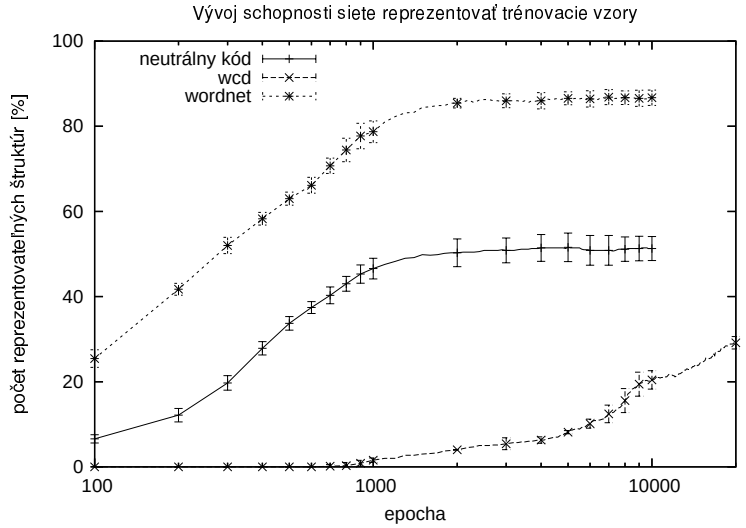
Po ukončení trénovania sme hľadali optimálne hodnoty prahu rekonštrukcie θ pre jednotlivé typy kódovania terminálov. V grafe 4.6 je zobrazený priemerný počet reprezentovateľných štruktúr z trénovacej množiny v závislosti od hodnoty θ. Optimálna hodnota pre neutrálny kód je 0.6, pre WCD reprezentácie 0.09 a pre WordNet reprezentácie 0.73.



Obr. 4.5: Vývoj chyby rekonštrukcie počas trénovania.



Obr. 4.6: Optimálna hodnota parametra θ. Pre neutrálny kód je to 0.6, pre WCD reprezentácie 0.09 a pre WordNet reprezentácie 0.73.



Obr. 4.7: Percento úspešne dekódovaných štruktúr v závislosti od počtu epoch. Znázornené sú jednak priemerné hodnoty zo siedmich simulácií a tiež štandardné odchýlky.

Takto získané optimálne hodnoty prahu rekonštrukcie θ pre jednotlivé typy kódovania sme použili na zistenie počtu reprezentovateľných štruktúr z trénovacej množiny. V grafe 4.7 je znázornený vývoj tohoto počtu v závislosti od počtu epoch. Uvádzame priemerné hodnoty zo siedmich simulácií a štandardné odchýlky.

4.4 Zhrnutie

Podarilo sa nám trénovať 300–100–300 lineárnu RAAM na trénovacej množine obsahujúcej spolu s podštruktúrami až 325 rôznych objektov. Navrhli sme postup, ako sa dá zvoliť vhodná rýchlosť učenia η a počiatočné hodnoty váh tak, aby sa váhy sieti v rozumnom čase ustálili. Tiež sme načrtli spôsob, ako sa dá približne určiť optimálny prah rekonštrukcie θ_{opt} iba na základe zvolených reprezentácií terminálov.

Cieľom experimentu bolo porovnať vplyv troch typov reprezentácií terminálnych symbolov na reprezentačné schopnosti siete. Najviac štruktúr vedela sieť reprezentovať, keď sme použili WordNet reprezentácie. Na druhej strane, pri použití WCD reprezentácií celková chyba rekonštrukcie dosiahla najmenšie hodnoty, čo zodpovedá presnejšiemu dekódovaniu.

V našom prípade malá kapacita siete pri použití WCD reprezentácií je dôsledok toho, že reprezentácie niektorých slov sa na seba príliš podobali, t.j. ich euklidovská vzdialenosť bola veľmi malá. Tak bolo nutné použiť malú hodnotu prahu rekonštrukcie θ, čo malo za následok slabé reprezentačné schopnosti siete.

Skúsili sme preto všetky vektory zväčšiť o konštantný násobok, čím by sme zároveň dosiahli zväčšenie minimálnej vzdialenosti medzi dvoma slovami. Pritom sme si stanovili podmienku, že hodnoty na jednotlivých pozíciach nesmú byť väčšie ako 1. Avšak v rámci viet vygenerovaných pomocou SLG sa napríklad v 41% prípadoch vyskytovalo vpravo od slova `women` zámeno `who_pl`, takže hodnota na príslušnej pozícii vo vektore WCD reprezentácie slova `women` bola 0.41. Podobná situácia bola vo viacerých prípadoch. Takto bolo možné naškálovať vektory iba približne na dvojnásobok, pri zachovaní stanovenej podmienky o maximálnej hodnote 1 na jednotlivých pozíciach.

Jedno riešenie tohoto problému môže spočívať v návrhu takej gramatiky pre SLG, že vo vygenerovaných vetách sa budú dvojice slov vyskytovať rovnomernejšie, t.j. výsledné pravdepodobnosti vzájomných výskytov dvoch slov budú rovnomerne rozložené medzi viacero dvojíc. Zrejme bude potrebné zvoliť zložitejšiu štruktúru viet, ako to bolo v našom prípade (časť 4.1.1). Druhá možnosť je naškálovať vektory vygenerované WCD modelom o dostatočný násobok, hoci aktivity neurónov pri takýchto reprezentáciach budú väčšie ako 1, t.j. mimo bežne zaužívaných hodnôt. Tretia možnosť spočíva v zväčšení dimenzie reprezentácií pridaním niekoľkých bitov s nulovými hodnotami. Tým poskytneme sieti voľnosť, čo sa môže prejaviť v presnejších rekonštrukciách štruktúr.

Kapitola 5

Úrovne systematickosti

V roku 1988 Fodor & Pylyshyn [4] predniesli výzvu pre konekcionistické modely, aby objasnili systematickosť pozorovanú v ľudskej kognícii bez toho, že by de facto implementovali klasickú architektúru. Následne bolo uvedených viacero konekcionistických modelov, o ktorých autori tvrdili, že do určitej miery sú schopné systematickosť dosiahnuť. Avšak nebolo možné rozhodnúť, či je to naozaj pravda. Ako uvádzajú Niklasson & van Gelder [17], problém bol v tom, že Fodor & Pylyshyn neuviedli žiadnu presnú definíciu systematickosti a hoci tvrdili, že systematickosť je v súvislosti s klasickými modelmi tradičný pojem, neodkazovali sa na žiadne predchádzajúce práce. Nanajvýš systematickosť opisovali iba v náznakoch a analógiach.

Hadley [16] bol prvý, kto sa podrobne venoval otázke, čo v skutočnosti systematickosť je a čo musia konekcionistické modely predviesť, aby o nich bolo možné povedať, že systematickosť dosiahli. Zistil, že problém systematickosti musí byť pre konekcionistické modely formulovaný ako problém učenia v nasledovnom zmysle. Keď sieť (model) nadobudne v dôsledku trénovania schopnosť reprezentovať určitú skupinu objektov (trénovaciu množinu), v akom vzťahu sú k tejto skupine ostatné objekty, ktoré je takto adaptovaná sieť automaticky schopná reprezentovať?

Hadley definoval celkom tri úrovne systematickosti: slabú (*weak*), kvázi- (*quasi-*) a silnú (*strong*). Niklasson & van Gelder [17] navrhli inú, podľa nich jednoduchšiu a zároveň obsiahlejšiu klasifikáciu úrovní systematickosti, pričom tvrdili, že systematickosť môže byť definovaná mnohými spôsobmi.

V tejto kapitole opíšeme tri experimenty, na ktorých pre jednotlivé typy kódovania terminálov preskúmame, či lineárna RAAM dosahuje prvú, druhú, resp. tretiu úroveň systematickosti podľa Niklassona & van Geldera [17] (ďalej

skrátene N&vG). Všetky tri experimenty sú veľmi podobné tomu z predošlej kapitoly, navyše navzájom sa od seba líšia iba použitou trénovacou a testovacou množinou. Ostatné parametre budú pre všetky tri experimenty rovnaké. N&vG definovali celkom päť úrovní systematickosti. Zmienime sa preto aj o tom, či je lineárna RAAM schopná dosiahnuť štvrtú a piatu úroveň. V závere potom vyhodnotíme získané výsledky.

5.1 Príprava rozšírenej bázy štruktúr

Vety vygenerované pomocou SLG (pozri časť 4.1.1) využijeme ešte raz. Trénovacia množina v experimente z predošlej kapitoly pozostávala z ternárnych štruktúr získaných tranformovaním 100 viet. Nasledujúce experimenty vyžadujú trénovať model na inej množine viet, na akej potom bude otestovaný. Preto sme pretransformovali ďalších 200 z viet vygenerovaných SLG generátorom. Takto sme dostali *rozšírenú bázu* obsahujúcu 300 ternárnych štruktúr, z ktorej budeme čerpať pri príprave trénovacích a testovacích množín pre nasledujúce simulácie.

Pri dodatočnom výbere 200 viet sme postupovali nasledovne. Všetky vety vygenerované generátorom SLG sme rozdelili podľa toho, koľko vedľajších viet obsahuju. Napríklad, keď bol podmet rozvitý dvojnásobne a predmet trojnásobne, veta v sebe zahrňovala päť vedľajších viet. Z celkového množstva 10000 vygenerovaných viet sme odstránili viacnásobné výskyty, čím nám ostalo 5598 rôznych viet. Z nich sme ešte vylúčili 38 viet, ktoré mali viac ako tri vedľajšie vety. Do rozšírenej bázy sme potom vybrali také množstvá jednoduchých viet a viet s jednou, dvoma resp. troma vedľajšími vetami, aby ich bolo spolu 300 a aby pomery jednotlivých počtov zodpovedali rozdeleniu všetkých viet. Konkrétne počty sú uvedené v tabuľke 5.1.

5.2 Parametre spoločné pre experimenty

Trénovaciu množinu bude v nasledujúcich experimentoch tvoriť vždy 150 štruktúr vybraných z rozšírenej bázy. Avšak výber sa bude riadiť pravidlami, ktoré budú pre každú úroveň systematickosti iné. Testovacia množina nebude vždy obsahovať rovnaký počet štruktúr. Výber štruktúr do trénovacej a testovacej množiny presne opíšeme pre každý experiment zvlášť.

Budeme trénovať ternárnu lineárnu RAAM dimenzie $k = 150$. Termi-

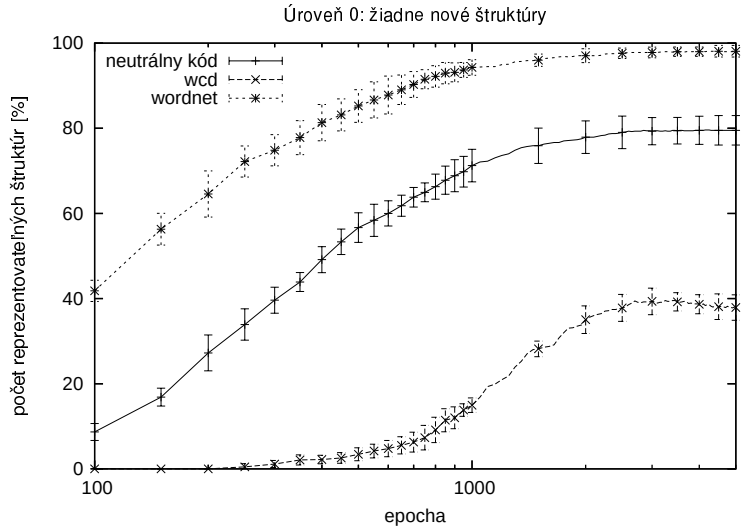
# vedľajších viet vo vete	# vygenerovaných viet toho druhu	# vybraných viet toho druhu
0	734	40
1	3173	171
2	1406	76
3	247	13
4 a viac	38	0
Spolu:	5598	300

Tabuľka 5.1: Počty viet vybraných do *rozšírenej bázy*. Vety do rozšírenej bázy sme vyberali tak, aby pomery počtov vybraných jednoduchých viet, viet s jednou, dvoma resp. troma vedľajšími vetami zodpovedali rozdeleniu 5598 vygenerovaných viet.

nálne symboly boli v predošlej časti reprezentované vektormi dimenzie 100. Pre nasledujúce experimenty pridáme 50 miest s nulovými hodnotami. Tak poskytneme sieti väčšiu voľnosť, čím sa zvýši jej kapacita.

Schopnosť siete dosiahnuť určité jednotlivé úrovne systematickosti budeme testovať opäť pre tri spôsoby reprezentovania terminálov. V prípade neutrálneho kódu a WordNet reprezentácií inicializujeme váhy náhodne z intervalu $[-0.1; 0.1]$, rýchlosť učenia zvolíme $\eta = 0.001$. V prípade WCD reprezentácií sa ukázala najvhodnejšou nasledujúca voľba: váhy inicializujeme z intervalu $[-0.25; 0.25]$ a rýchlosť učenia bude $\eta = 0.005$. Proces učenia ukončíme vždy po 5000 epochách.

Predmetom skúmania je schopnosť siete reprezentovať štruktúry, ktoré sú určitým spôsobom nové oproti trénovacej množine. Na vyčíslenie počtu reprezentovateľných štruktúr potrebujeme určiť prah rekonštrukcie. V nasledujúcich experimentoch budeme používať nasledovné hodnoty. Pre neutrálny kód bude $\theta_{neutral} = 0.55$, pre WCD reprezentácie bude $\theta_{WCD} = 0.1$ a nakoniec, pre WordNet reprezentácie zvolíme $\theta_{WordNet} = 0.7$. Tieto hodnoty približne zodpovedajú optimálnym hodnotám pre jednotlivé typy reprezentácií (pozri časť 4.2.1).



Obr. 5.1: Počet štruktúr trénovacej množiny reprezentovateľných sieťou. V grafe sú uvedené priemerné hodnoty zo siedmich simulácií a štandardné odchýlky. Trénovaciu množinu tvorilo v každej simulácii iných 150 štruktúr z rozšírenej bázy.

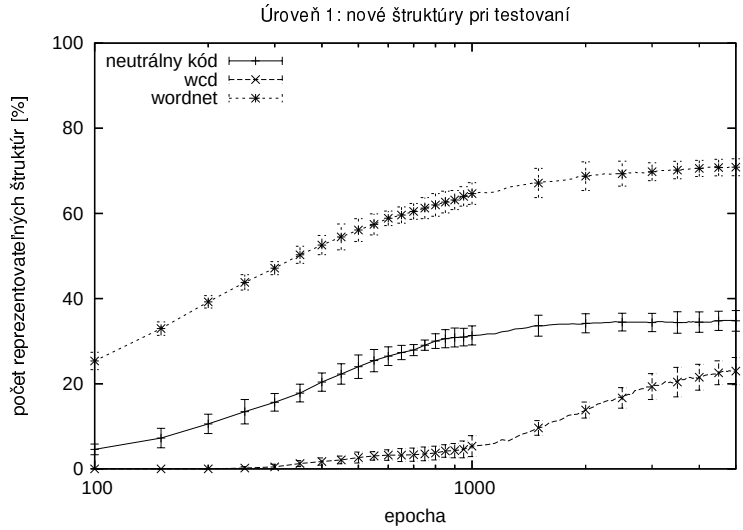
5.3 Experimenty

5.3.1 Úroveň 1: nové štruktúry

Definícia 1 (N&vG) *Konekcionistický model dosahuje prvú úroveň systematickosti, ak je schopný reprezentovať štruktúry, ktoré sa neobjavili v rámci trénovacej množiny, ale obsahujú symboly na takých pozíciach, na ktorých sa vyskytovali aj v štruktúrach trénovacej množiny.*

Do trénovacej množiny náhodne vyberieme 150 štruktúr z rozšírenej bázy štruktúr. Testovaciu množinu potom bude tvoriť zvyšných 150 štruktúr. Celkovo sme spustili 7 simulácií, pričom trénovaciu množinu tvorilo vždy iných 150 štruktúr z rozšírenej bázy.

Najprv sme zistili, koľko štruktúr z trénovacej množiny je sieť schopná reprezentovať. Výsledné počtu sú uvedené v grafe 5.1. N&vG definovali nultú úroveň systematickosti ako schopnosť modelu reprezentovať iba také vety, ktoré sa vyskytovali v rámci trénovacej množiny. Lineárna RAAM, samozrejme dosahuje nultú úroveň systematickosti, nakoľko je schopná reprezentovať štruktúrované objeky.



Obr. 5.2: Počet štruktúr testovacej množiny reprezentovateľných sieťou. V grafe sú uvedené priemerné hodnoty zo siedmich simulácií a štandardné odchýlky. Testovaciu množinu tvorilo v každej simulácii tých 150 štruktúr z rozšírenej bázy, ktoré sme nevybrali do trénovacej množiny.

Zväčšením dimenzie reprezentácií sme dosiahli, že pri použití WordNet reprezentácií vedela naučená sieť reprezentovať takmer všetkých 150 štruktúr trénovacej množiny. V prípade neutrálneho kódu to bolo približne 80% a v prípade WCD reprezentácií približne 40% štruktúr trénovacej množiny.

Potom sme sieti predložili štruktúry testovacej množiny. V grafe 5.2 je znázornené, koľko štruktúr vedela sieť reprezentovať pre jednotlivé spôsoby kódovania terminálov. Uvádzame priemerné hodnoty zo siedmich simulácií, ktoré boli spustené vždy s inou trénovacou množinou, a štandardné odchýlky.

V prípade neutrálneho kódovania vedela sieť reprezentovať približne 40% nových štruktúr v rámci testovacej množiny. Avšak v rámci trénovacej množiny to bolo až 80% (pozri graf 5.1). Ak predpokladáme, že kapacita siete je dostatočná na reprezentovanie 120 štruktúr (čo je 80% zo 150), potom môžeme povedať, že sieť reprezentovala iba polovičný počet nových štruktúr, ktoré mohla reprezentovať. Keď spravíme rovnaké porovnanie pre WCD a WordNet reprezentácie, zistíme, že počty nových reprezentovateľných štruktúr dosahujú približne 75% kapacitných možností pre tieto typy kódovania.

Všimnime si tiež, že štandardné odchýľky sú relatívne malé oproti priemerným hodnotám. Preto v nasledujúcich dvoch experimentoch spustíme

menej simulácií a štandardné odchýlky už nebudeme uvádzť.

5.3.2 Úroveň 2: symboly na nových pozíciach

Pod druhú úroveň systematickosti spadá príklad uvádzaný Fodorom & Pylyshynom [4], citujem:

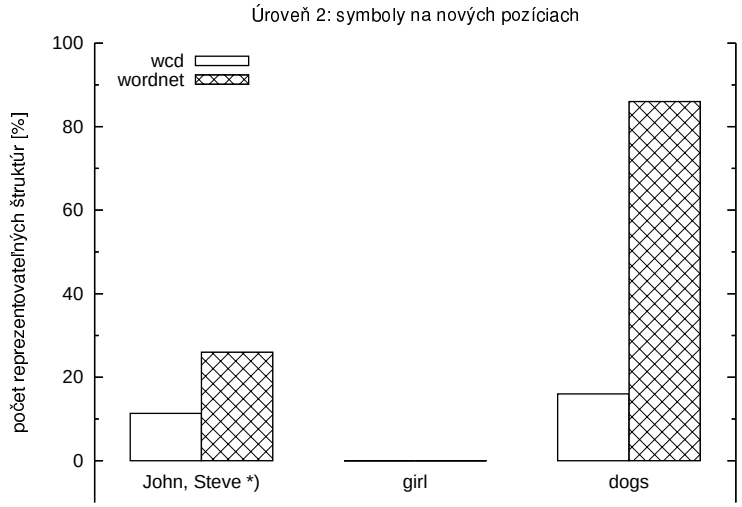
> „What does it mean to say that thought is systematic? Well, just as you don't find people who can understand the sentence 'John loves the girl' but not the sentence 'the girl loves John', so too you don't find people who can think the thought that John loves the girl but can't think the thought that the girl loves John."
>
> Fodor & Pylyshyn, [4]

Definícia 2 (N&vG) *Konekcionistický model dosahuje druhú úroveň systematickosti, ak je schopný reprezentovať štruktúry obsahujúce aspoň jeden symbol na pozícii, na ktorej sa neobjavil v štruktúrach trénovacej množiny.*

Pre tento experiment potrebujeme pripraviť trénovaciu množinu tak, aby obsahovala iba také štruktúry, v ktorých sa dané slovo nevyskytuje na všetkých možných pozíciach. Testovaciu množinu potom budú tvoriť štruktúry, v ktorých sa dané slovo nachádza na ostatných pozíciach.

V našom prípade je ľavá pozícia v rámci štruktúr určená výhradne pre slovesá. Podstatné mená sa však môžu vyskytovať tak vo funkcii podmetu (teda na strednej pozícii), ako aj predmetu (na pravej pozícii). Schopnosť lineárnej RAAM dosiahnuť druhú úroveň systematickosti sme skúmali celkom pre tri typy podstatných mien. V prvom prípade boli z rozšírenej bázy vylúčené štruktúry obsahujúce na pravej pozícii (teda vo funkcii predmetu) vlastné mená `John` a `Steve`. V druhom prípade sme vylúčili štruktúry obsahujúce na pravej pozícii podstatné meno jednotného čísla `girl`. V treťom prípade to bolo podstatné meno v množnom čísle `dogs`. Zo štruktúr, ktoré sme v jednotlivých prípadoch z rozšírenej bázy nevylúčili, sme potom vybrali 150 na trénovanie.

Testovaciu množinu vo všetkých troch prípadoch tvorili štruktúry, ktoré v sebe niekde obsahovali príslušné podstatné meno vo funkcii predmetu, t.j. na pravej pozícii. Keďže takých sa v rámci rozšírenej bázy nachádzalo príliš málo na získanie relevantných výsledkov, pretransformovali sme ďalších 150 viet, po 50 pre každú testovaciu množinu.



Obr. 5.3: Dosiahnuté výsledky pre druhú úroveň systematickosti. Zobrazené sú priemerné počty reprezentovateľných štruktúr z troch simulácií pre každý prípad. *) V prípade slov `John` a `Steve` sme použili prah $\theta = 0.07$ pre WCD, resp. $\theta = 0.9$ pre WordNet reprezentácie.

Pri použití neutrálneho kódovania symbolov sieť nebola schopná reprezentovať ani jednu novú štruktúru. Pre zvyšné dva typy reprezentácií boli výsledky o niečo priaznivejšie. V grafe 5.3 sú uvedené priemerné počty reprezentovateľných štruktúr testovacej množiny. Priemerné hodnoty sú uvedené z výsledkov troch simulácií pre každý prípad.

Z výsledkov je zrejmé, že pri použití WCD, resp. WordNet reprezentácií bola do určitej miery dosiahnutá druhá úroveň systematickosti. Nebudeme sa opäť vracať k rozdielnym počtom v dôsledku použitia jedného, či druhého typu kódovania terminálov. Zamerajme sa skôr na rozdiely pre jednotlivé prípady slov. Najviac testovacích štruktúr vedela sieť reprezentovať v prípade slova `dogs`. V prípade slov `John` a `Steve` sme museli zvoliť iné prahy rekonštrukcie na dosiahnutie aspoň takých hodnôt, ako sú uvedené v grafe 5.3. Nakoniec, v prípade slova `girl` sieť nevedela reprezentovať žiadne štruktúry.

Príčinu týchto rozdielov možno objaviť v testovacích množinách. Testovacia množina pre druhý prípad obsahuje slovo `girl` iba na najspodnejších úrovniach v rámci štruktúr, teda v hĺbke aspoň 3. Je očakávateľné, že chyba pri rekonštrukcii nového slova bude nepresnejšia ako chyba rekonštrukcie ostatných slov, ktoré sa v trénovacej množine vyskytovali na rovnakých pozí-

ciach ako v testovacej množine. Keď sa nové slovo nachádza vo väčšej hĺbke, chyba sa prenáša aj na redukované reprezentácie príslušných nadštruktúr. Pritom sa pravdepodobne s každou úrovňou zväčšuje jej dopad na úspešnosť rekonštrukcie príslušnej nadštruktúry. Preto sme v prípade slova `girl` nedostali žiadne reprezentovateľné štruktúry. Túto hypotézu potvrdujú aj reprezentovateľné štruktúry pre prípad slov `John` a `Steve`: dané slová sa v nich nachádzali vždy v hĺbke 1.

5.3.3 Úroveň 3: nové symboly

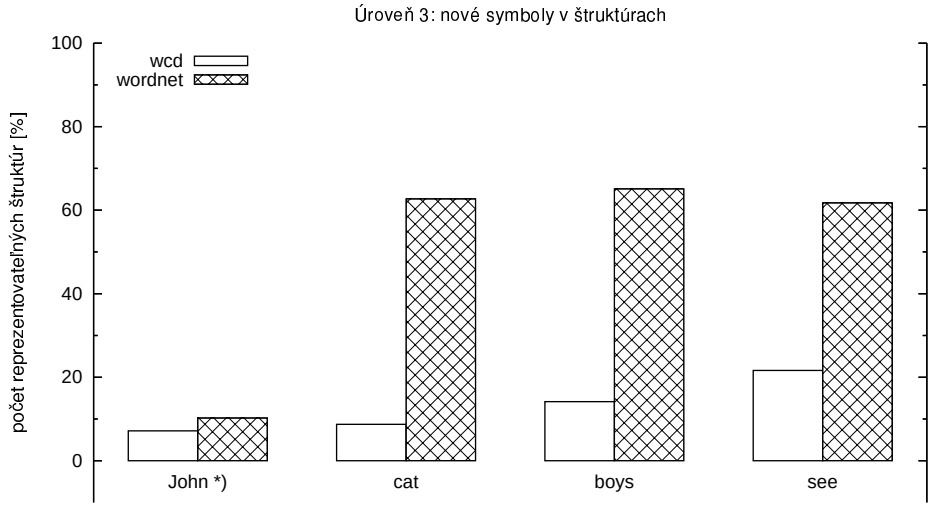
Ľudia majú schopnosť pochopiť vetu obsahujúcu také slovo, ktorého význam nepoznajú. Zrejme na to stačí skúsenosť, že v danom kontexte sa zvyknú nachádzať iba slová s určitým významom. Definícia tretej úrovne systematickosti zahŕňa tieto pozorovania.

Definícia 3 (N&vG) *Konekcionistický model dosahuje tretiu úroveň systematickosti, ak je schopný reprezentovať štruktúry obsahujúce aspoň jeden symbol, ktorý sa neobjavil v štruktúrach trénovacej množiny.*

Pre tento experiment bude príprava trénovacej a testovacej množiny o niečo jednoduchšia. Z rozšírenej bázy vylúčime najprv tie štruktúry, ktoré obsahujú zvolené slovo. Zo zostávajúcich následne vyberieme 150 štruktúr do trénovacej množiny. Príslušnú testovaciu množinu budú tvoriť všetky v prvom kroku vylúčené štruktúry, t.j. tie, v ktorých sa nachádza zvolené slovo. Testovacia množina teda bude mať pre každý prípad inú veľkosť. Podotýkame, že tentoraz je možné vytvoriť trénovaciu a testovaciu množinu spĺňajúcu podmienku definície aj pre slovesá.

Celkovo sme pripravili štyri sady množín. V prvom prípade sa v rámci trénovacej množiny nevyskytlo vlastné meno `John`. Testovaciu množinu tvorilo potom 49 štruktúr, pričom slovo `John` sa vyskytlo v každej z nich. V druhom prípade sme z trénovania vylúčili podstatné meno v jednotnom čísle `cat`, trénovacia množina obsahovala 63 štruktúr. Tretí prípad sa týkal podstatného mena v množnom čísle `boys` (53 štruktúr na testovanie) a posledný, štvrtý prípad zasa slovesa `see` (81 štruktúr).

Počty reprezentovateľných štruktúr, vyjadrené percentuálne vzhľadom na veľkosť testovacej množiny, sú znázornené v grafe 5.4. Uvedené sú priemené hodnoty z dvoch simulácií pre každé uvažované slovo a pre WCD a WordNet reprezentácie.



Obr. 5.4: Dosiahnuté výsledky pre tretiu úroveň systematickosti. Zobrazené sú priemerné počty reprezentovateľných štruktúr z dvoch simulácií pre každý prípad. *) V prípade slova `John` sme opäť použili prah $\theta = 0.07$ pre WCD, resp. $\theta = 0.9$ pre WordNet reprezentácie.

Aby sieť bola schopná reprezentovať štruktúry obsahujúce slovo, ktoré sa nevyskytlo v rámci trénovacej množiny, musí k danému novému slovu v štruktúrach trénovacej množiny existovať iné, „veľmi podobné“ slovo, pričom miera podobnosti slov sa v tomto prípade určuje na základe euklidovskej vzdialenosti ich reprezentácií. V tomto zmysle sú si slová pri neutrálnom kódovaní navzájom rovnako (ne)podobné, teda niet divu, že sieť nevedela reprezentovať ani jednu štruktúru z testovacích množín.

Pri WordNet reprezentáciach sa dvojice slov `cat` a `cats`, resp. `see` a `sees` líšia iba v jedinom príznaku. Euklidovská vzdialenosť ich reprezentácií je teda rovná 1. Podobné *príbuzné* dvojice slov možno vyčítať aj z dendrogramu pre WCD reprezentácie (obr. 4.2). Napríklad euklidovská vzdialenosť slovies `see` a `hear` je (po naškálovaní vektorov na dvojnásobok) približne 0.15.

Všimnime si ešte, ako veľmi sa líšia výsledné počty pre prvý prípad (slovo `John`) od ostatných prípadov, keď bolo použité WordNet kódovanie. Najbližšie k reprezentácii vlastného mena `John` je reprezentácia vlastného mena `Steve` (vzdialnosť je $\sqrt{2}$). Druhá najbližšia je potom reprezentácia slova `man`, resp. slova `boy` (vzdialnosť je $\sqrt{3}$). Keďže `Steve` sa nachádzal iba v 10 zo 150 štruktúr trénovacej množiny, testované slovo `John` v podstate nemalo v rámci

trénovacích vzorov svoje príbuzné slovo (na rozdiel od ostatných prípadov).

5.3.4 Ďalšie úrovne systematickosti

Niklasson & van Gelder definovali ešte ďalšie dve úrovne systematickosti.

Definícia 4 (N&vG) *Konekcionistický model dosahuje štvrtú úroveň systematickosti, ak je schopný reprezentovať štruktúry, ktoré majú väčšiu zložitosti (hĺbku), ako mali štruktúry trénovacej množiny.*

Definícia 5 (N&vG) *Konekcionistický model dosahuje piatu úroveň systematickosti, ak testovacie štruktúry obsahujú aspoň jeden nový symbol na novej úrovni zložitosti.*

Do rozšírenej bázy štruktúr (časť 5.1) sme schválne nezaradili vety, ktoré mali viac ako tri vedľajšie vety. Na jednej strane ich bolo tak málo, že by sme do rozšírenej bázy mohli vybrať iba dve z nich, aby sme dodržali príslušné pomery počtov viet. Druhý dôvod bol ten, že teraz ich môžeme využiť na test, či lineárna RAAM dosahuje štvrtú úroveň systematickosti podľa N&vG.

V prvom experimente tejto kapitoly sme vybrali do trénovacej množiny polovicu viet z rozšírenej bázy. Takto vybranú trénovaciu množinu môžeme použiť aj teraz, nakoľko definícia štvrtej úrovne systematickosti nekladie obmedzenia na výskyt symbolov v rámci trénovacej a testovacej množiny. Alebo, ešte lepšie, môžeme využiť natrénovanú sieť z prvého experimentu na otestovanie štvrtej úrovne.

Ukázalo sa, že lineárna RAAM nie je schopná dosiahnuť štvrtú úroveň systematickosti pre žiaden z troch používaných typov reprezentácií slov. V predchádzajúcich experimentoch sme zasa zistili, že sieť má problém reprezentovať štruktúry s novým symbolom vo väčšej hĺbke. Preto už zrejme nemá význam skúmať, či je lineárna RAAM schopná dosiahnuť aj piatu úroveň systematickosti.

5.4 Zhrnutie

V tejto kapitole sme skúmali, akú úroveň systematickosti podľa klasifikácie Niklassona & van Geldera [17] je lineárna RAAM schopná dosiahnuť pri použití niektorého z troch typov kódovania.

Ukázalo sa, že pri použití neutrálneho kódovania symbolov je model lineárnej RAAM schopný dosiahnuť v určitej miere iba prvú úroveň systematickosti. K rovnakým záverom dospeli aj Voegtlin & Dominey [2, časť 6.5]: ternárna lineárna RAAM sa nenaučila reprezentovať štruktúry typu `( ate meat Mary )` („mäso zjedlo Marienku“), pretože slovo `meat` sa v rámci trénovacej množiny nevyskytlo na strednej pozícii (vo funkcii podmetu).

V simuláciach, keď boli symboly reprezentované WCD, resp. WordNet reprezentáciami, lineárna RAAM bola schopná za určitých okolností dosiahnuť aj druhú a tretiu úroveň systematickosti. Ukázalo sa, že miera tejto schopnosti závisí od dvoch vecí:

- Po prvé, v dôsledku očakávanej nepresnej rekonštrukcie slova, ktoré je pri testovaní určitým spôsobom nové[1], sa chyba rekonštrukcie viac prejaví, ak sa nové slovo v rámci štruktúry vyskytuje vo väčšej hĺbke.
- Po druhé, ak sa v trénovacích vzoroch nachádza slovo s podobnou reprezentáciou[2], ako je reprezentácia nového slova, potom je takto natrénovaná sieť schopná reprezentovať aj štruktúry obsahujúce nové slovo, lebo z pohľadu siete vlastne toto nové slovo „nie je až také nové“.

[1] Pripomíname, že v prípade druhej úrovni systematickosti sa niektoré slovo nachádza pri testovaní na novej pozícii oproti trénovacím vzorom. V prípade tretej úrovne je niektoré slovo pri testovaní úplne nové.

[2] Podobnosť chápeme v zmysle euklidovskej vzdialenosti.

Kapitola 6

Záver

Voegtlin & Dominey [2] nedávno predstavili silnú modifikáciu klasického modelu rekurzívnej autoasociatívnej pamäte (RAAM, Pollack [7]), keď použili lineárne neuróny a Ojovo pravidlo Hebbovského učenia. V reprodukovanom experimente podľa [2] sme potvrdili, že lineárna RAAM je schopná reprezentovať o jeden až dva rády viac štruktúr, ako jej bolo prezentovaných počas trénovania. Vďaka menšej rýchlosti učenia sme dosiahli stabilizáciu váh po určitom počte epoch, čím je stanovený koniec procesu učenia. Už V&D naznačili, že výsledná kapacita siete veľmi závisí od počiatočných podmienok, t.j. od inicializácie váh. Rovnaký jav sme pozorovali aj v naších výsledkoch. Zdôvodňujeme to tým, že pre rôzne inicializácie váh sú rôzne aj výsledné matice váh. Ďalej sme zistili, že hoci je kapacita modelu obrovská, z celkového množstva reprezentovateľných štruktúr je iba málo gramaticky správnych. Presnejšie, z tohoto hľadiska dosahuje lineárna RAAM rovnaké výsledky ako pôvodný Pollackov model. Pokladáme za dôležité overiť toto pozorovanie na rozsiahlejších vstupných dátach, ako je Pollackov korpus, ktorý vrátane podštruktúr obsahuje iba 15 stromov.

V ďalších častiach tejto práce sme skúmali vplyv kódovania vstupov (terminálov) na výsledné vlastnosti siete. Celkom sme porovnávali tri typy reprezentácií: neutrálne (lokalistické) kódovanie, ďalej reprezentácie vytvorené modelom *word co-occurence detector*, ktoré vyjadrujú kontext výskytu slov v texte, a nakoniec reprezentácie získané z databázy WordNet [14] extrakciou charakteristických príznakov [13]. Najviac štruktúr vedela sieť reprezentovať, keď sme použili WordNet reprezentácie. Na druhej strane, pri použití WCD reprezentácií celková chyba rekonštrukcie dosiahla najmenšie hodnoty, čo zodpovedá presnejšiemu dekódovaniu.

Vplyv reprezentovania terminálnych symbolov sme skúmali ešte hlbšie. Zistili sme, že podľa klasifikácie Niklassona & van Geldera [17] lineárna RAAM dosahuje tretiu úroveň systematickosti za predpokladu, že použijeme WCD alebo WordNet reprezentácie. To znamená, že po natrénovaní je sieť schopná reprezentovať štruktúry obsahujúce symboly, ktoré sa nevyskytli v rámci štruktúr trénovacej množiny. Miera tejto schopnosti závisí jednak od hĺbky, v akej sa nachádza nový symbol v testovacích štruktúrach, a tiež od existencie symbolu s podobnou reprezentáciou v štruktúrach trénovacej množiny.

Literatúra

[1] Voegtlin, T. (2002). Neural networks and self-reference. *PhD Thesis, Universite Lyon 2*

[2] Voegtlin, T., & Dominey, P.F. (2005). Linear recursive distributed representations. *Neural Networks, 18*, 878-895

[3] Voegtlin, T. (2005). Recursive principal components analysis. *Neural Networks, 18(8)*, 1040-50

[4] Fodor, J.A., & Pylyshyn, Z.W. (1988). Connectionism and cognitive architecture: A critical analysis. *Cognition, 28*, 3-71

[5] Rybár J., Kvasnička V. & Farkaš, I. (2005). Jazyk a kognícia. *Kalligram, Bratislava*

[6] Aydede, M. (2004). The Language of Thought Hypothesis. *The Stanford Encyclopedia of Philosophy (Fall 2004 Edition), E. N. Zalta (ed.),* `plato.stanford.edu/archives/fall2004/entries/language-thought`

[7] Pollack, J. B. (1990). Recursive distributed representations. *Artificial Intelligence 46, 1*, 77-105

[8] Pollack, J. B. (1989). Implications of recursive distributed representations. In *Advances in Neural Information Processing Systems I, San Mateo: Morgan Kaufmann*, 527-536.

[9] Farkas, I., & Li, P. (2001). A self-organizing neural network model of the acquisition of word meaning. In *E. Altmann, A. Cleeremans, C. Schunn, & W. Gray (Eds.), Proceedings of the 4th International Conference on Cognitive Modeling, Fairfax, VA*, pp. 67-72

[10] Farkas, I., & Li, P. (2002). Modeling the development of lexicon with a growing self-organizing map. In H. J. Caulfield, et al. (Ed.), *Proceedings of the sixth joint conference on information sciencesx*, pp. 67-72. Durham, NC: JCIS/Association for Intelligent MAchinery, Inc.

[11] Li, P., Farkas, I., & MacWhinney, B. (2004). Early lexical acquisition in a self-organizing neural network. *Neural Networks, 17(8-9)*, 1345-1362

[12] Haykin S. (1999). Neural Networks: A Comprehensive Foundation (2nd ed.). *Prentice Hall, NJ*

[13] Harm, M. (2002). Building large scale distributed semantic feature set with WordNet. *Technical report PDP-CNS-02-1, Carnegie Mellon University*

[14] Miller, G. A. (1990). WordNet: An on-line lexical database. *International Journal of Lexicography, 3*, 235-312

[15] Rohde, D. (1999). Simple language generator: Encoding complex languages with simple grammars. *Technical report CMU-CS-99-123, Pittsburg, PA: Carnegie Mellon University*

[16] Hadley, R. F. (1994). Systematicity in connectionist language learning. *Mind and language, 9(3)*, 247-272

[17] Niklasson, L. F., & van Gelder, T. (1994). On being systematically connectionist. *Mind and Language, 9*, 288-302

[18] Boden, M., & Niklasson, L. (2000). Semantic systematicity and context in connectionist networks. *Connection Science, 12(2)*, pp. 1 - 31

[19] Hammerton, J. A. (1998). Holistic computation: Reconstructing a muddled concept. *Connection Science, 10(1)*, 3-19

[20] Chalmers, D. J. (1990). Syntactic transformations on distributed representations. *Connection Science, 2*, 53-62

[21] Christman, L. (1991). Learning recursive distributed representations for holistic computation. *Connection Science, 3*, 345-366

[22] Baldi, P., & Hornik, K. (1988). Neural networks and principal component analysis: Learning from examples without local minima *Neural Networks, 2(1)*, 53-58

[23] Sperduti, A. (1994). Labeling RAAM. *Connection Science, 6(4)*, 429-459

[24] MathWorld, the web's most extensive matematics resource. `http://mathworld.wolfram.com/`