Универзитет у Београду
Електротехнички факултет

Одређивање сличности између научних радова применом метода машинског учења

Мастер рад

Ментор:

Проф. др Жељко Ђуровић

Кандидат:
Марија Станојевић
2016/3132

Београд, септембар 2017.

САДРЖАЈ

1. Увод

Циљ овог рада је да се направи преглед методологија за обраду и разумевање текста и да се одреде семантички слични радови објављени на *arxiv.org* веб сајту. Семантична сличност представља сличност на основу значења текстова [1]. Овај проблем, иако лак за човека који познаје област у коме је рад објављен, представља један од тежих проблема у вештачкој интелигенцији. Број радова који се објављују у току године је све већи и људи више нису у стању да испрате све радове који су објављени чак ни у области њиховог истраживања, а још мање у другим областима. С друге стране, истраживања су данас у великој мери мултидисциплинарна или имају примену у различитим областима, зато је од велике важности да истраживач може са што мање труда да добије најновија сазнања која су везана за његову област, чак и у случају да рад није директно објављен у његовој области.

Данас се проблемима налажења сличности између текстова, груписања текстова по ауторима, по областима истраживања и садржају и одређивања кључних речи и теме текста баве многи водећи истраживачки центри и компаније. У сличне проблеме спадају и претраживање текстова, разумевање текста, превођење текста на други језик и одговарање на питања у вези са текстом. Помоћу машинског учења и машинске обраде природног језика највеће светске компаније и истраживачки центри покушавају да нађу што боља решења за горе наведене проблеме. Неке од познатијих примена су преводилац компаније *Google*, као и кућни асистенти компанија *Amazon, Apple* и *Google.*

Постоји неколико области машинског учења које се баве обрадом текста, а најпознатије су: процесуирање природног језика (*natural language processing – NLP*) и проналажење информација (*information retrieval - IR*). За решавање многих проблема се заједно са овим техникама користе и методе из области препознавања говора. У овом мастер раду фокус је на методологијама које уче језичке моделе у зависности од дистрибуција речи или дистрибуција секвенца речи. С друге стране, постоје бројни приступи који се баве разумевањем улога речи у реченици и конструкције реченица/текста на основу улога и редоследа речи у реченици. Детаље таквих модела овај рад неће дискутовати.

Иако ће се у овом раду превасходно радити са текстовима научних радова, методологијама које ће бити приказане могу се обрађивати и општи текстови са википедије, вести из новина, текстови као што је програмски код или кратки текстови са друштвених мрежа, као што је твитер. Сваки од ових типова текстова има своје специфичности, тако да не постоји јединствено решење за све проблеме и због тога ће у наредним поглављима бити представњено више различитих алгоритама. Такође, треба нагласити да су сви ови алгоритми тестирани и добро истражени на енглеском језику, док је њихова примена на другим језицима ограничена.

Српски језик је веома незгодан машинама за разумевање, тако да се неки алгоритми врло тешко могу применити на српски језик. Новији алгоритми који су у стању да добро раде и са српским језиком траже велику количину података и текстова како би могли да произведу одговарајуће моделе. Због тога чак и највеће компаније као што је *Google* имају проблеме са својим производима на језицима као што је српски. На пример, без обзира на јако добар

алгоритам, њихов преводилац за српски језик је лош у поређењу са преводиоцима на неке друге језике (нпр. шпански, немачки,...) за које постоји пуно више података.

При представљању алгоритама биће специфицирано који су услови за њихову примену, на које детаље треба обратити пажњу и како их имплементирати. На крају ће бити представњен метод за одређивање сличности између научних радова објављених на енглеском језику, на веб сајту *arxiv.org* у разним областима доступним на тој платформи. Ово је отворен проблем и ауторов допринос се састоји у томе да тестира различите моделе и одреди њихове перформансе за овај проблем. Модел представља комбинацију неких од представљених алгоритама уз модификације уведене од стране аутора овог рада.

Текст је подељен на целине према типовима обраде текста. У другом поглављу су описане методологије за претпроцесирање текста. Ово је први корак у дизајну машинског система који може да препознаје текст, међутим за различите алгоритме различити су начини за претпроцесирање. Ово поглавље садржи опис више техника као што су чишћење текстова од интерпункције, чишћење текстова од честих речи (*stop word removal*), одређивање основа речи (*stemming*), одређивање основа речи помоћу речника (*lemmatization*), издвајање речи, тј. токена (*tokenization*), груписање речи (*chunking*), синтаксно означавање речи (*tagging*) и парсирање (*parsing*).

Треће поглавље садржи методе за процесирање текста које су најчешће коришћене у обради текста. У овом поглављу описани су *TF-IDF (*term frequency–inverse document frequency*)* алгоритам, н-грами - методе које користе прозоре дужине *n* (*n-grams*), модели који се базирају на декомпозицији по сингуларнинм вредностима (*Singular value decomposition - SVD*) и новији алгоритми који се базирају на методама дубоког учења, као што су конволуционе мреже, рекурентне неуралне мреже и њихове подврсте. На крају, описане су и најкоришћеније методе за процесирање текстова који се одликују великом тачношћу и брзином: алгоритам претварања речи у вектор *word2vec (word to vector*) развијен у компанији *Google* и његова модификација *GloVe (Global Vectors for Word Representation)* развијен на универзитету *Stanford*.

Четврто поглавње садржи методе за кластеризацију текстова на основу њихових карактеристика добијених алгоритмима из претходног поглавља. Овде ће бити описана техника кластеризације методом *k* средњих вредности, хијерархијска кластеризација, спектрална кластеризација и новије методе базиране на дубоком учењу: кластеризација помоћу рекурентних неуралних мрежа и помоћу конволуционих неуралних мрежа.

Поред теоретске основе, описа услова и могућности за примену метода, у поглављима два до четири биће наведена имена неких библиотека или оквира (*framework*) у којима су ове технике већ имплементиране и спремне за коришћење. Код машинског учења, а поготово код дубоког учења препоручује се коришћење тих имплементација уколико је могуће, зато што су оне оптимизоване за што брже извршавање.

У петом поглављу биће описан алгоритам који је аутор користио за одређивање сличности између научних радова, почевши од начина на који су подаци прикупљени, па до евалуације резултата.

У закључку ће бити дата препорука за коришћење описаних метода и укратко објашњени резултати и закључци овог рада.

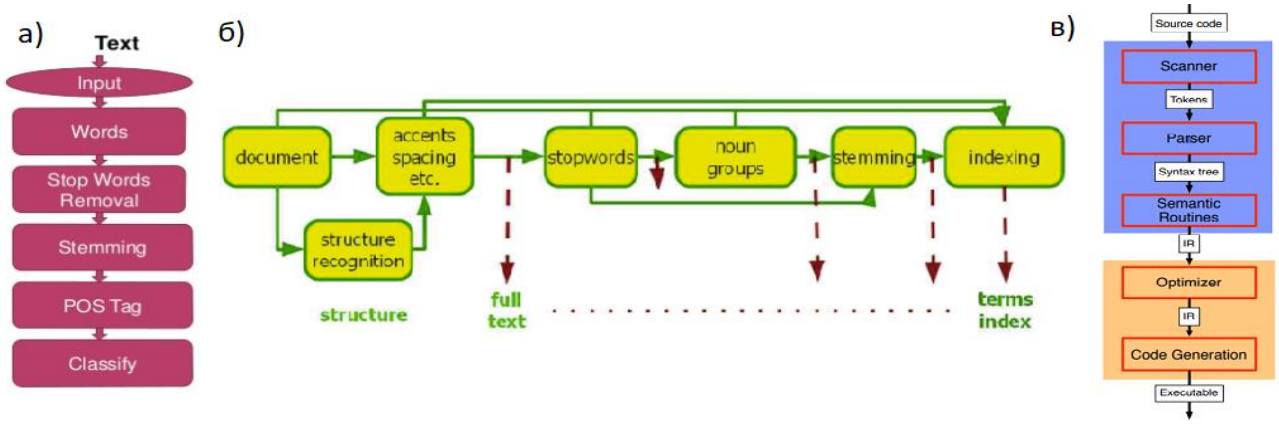
2. ПРЕТПРОЦЕСИРАЊЕ ТЕКСТА

У зависности од алгоритма за рад са текстом користе се и различите технике претпроцесирања текста. Постоје примери у којима се оригинални текст употребљава као улаз у алгоритам, али најчешће је потребно урадити неке од измена наведених у овом поглављу. У случају обраде кратких текстова, документи се могу надовезати један на други током претпроцесирања како би се могли применити алгоритми за рад са дугим текстовима. Треба имати у виду да текстови могу да садрже грешке настале током писања текста. То се може решити провером постојања речи у речнику за тај језик и у случају да не постоји налажењем најсличније речи. Овде се неће детаљније обрађивати овај случај, пошто се сматра да се грешке овог типа не налазе у научним радовима.

Битно је такође обратити пажњу на велика и мала слова и како њихово разликовање утиче на каснији алгоритам. При обради природног језика треба разликовати речи које почињу великим од оних које почињу малим словом ако оне имају различито значење (на пример: Вишња као име и вишња као воће). Међутим, не треба разликовати речи које почињу великим словом зато што су на почетку реченице. Нажалост, уколико је реч Вишња на почетку реченице, онда је у току претпроцесирања немогуће знати које значење та реч узима. Такође, треба напоменути да је најтачнији начин да се у фази претпроцесирања одреди које су то речи које имају различита значења кад почињу великим и малим словом да се креира списак таквих случајева за одговарајући језик. Технике дубоког учења овакве проблеме решавају закључивањем из контекста током процесирања текста и за њих се обично не ради велико претпроцесирање текста.

Примери дати у претходном пасусу показују зашто је рад са текстом тежак проблем. Специјални случајеви, вишеструка значења речи, различити типови текстова (од програмског кода до жаргонских текстова у порукама и на друштвеним мрежама), различити језици и променљивост природног језика су карактеристике о којима се мора водити рачуна током конструкције модела. Из тог разлога, постоји велики број техника за рад са текстом и претпроцесирање, а у наредним потпоглављима су наведене најкоришћеније методе.

На слици 2.1 дат је приказ три примера дијаграма обраде текста у којима се користе неки од наведених метода за претпроцесирање како би се показале различите могућности комбиновања техника за препроцесирање за различите примере обраде текста.



Слика 2.1. Приказ дијаграма: а) за обраду, таговање и класификацију текста [2]; б) за анализу текстуалних вести [3]; в) за креирање преводиоца за програмске језике [4].

2.1. Чишћење текста од интерпункције

Чишћење речи од интерпункције се најчешће обавља регуларним изразом који је креиран да избрише све знакове који нису од интереса и не помажу током касније обраде. Друга могућност је да се користи *replace* метода доступна у већини програмских језика, при чему се интерпукцијски знакови замењују празним стрингом.

Иако је имплементација ове методе једноставна, одлука да ли је треба применити треба да се донесе пажљиво. На пример, уколико се ради обрада програмског језика, чишћење од интерпункције је непожељно јер нам интерпункција помаже у разумевању. С друге стране, ако од *html* странице треба добити само текст и уз то одстранити структурне лабеле, то се може постићи овом методом. У том контексту се појам интерпункције проширује тако да укључи структурне лабеле.

Да ли и како треба обавити чишћење од интерпункције не зависи само од типа текста, већ је првобитно условљено даљом обрадом. Код природног текста, на пример, у случају да се касније издвајају и упоређују реченице, треба задржати знакове интерпункције, а у случају да су потребне само речи треба извршити чишћење.

Под техником чишћења текста се сматра и промена великих слова у мала, које се примењује само у одговарајућим случајевима као што је описано у другом параграфу поглавља Процесирање текста. Уколико је битно разликовати мала од великих слова и постоје речи које имају различита значења у различитим случајевима, потребно је имати речник таквих речи. Такви речници се најчешће стварају заједнички и доступни су свима за коришћење и унапређивање. Пожељно је укључити лингвисте и познаваоце језика у процес формирања речника.

2.2. Чишћење текста од честих речи (*stop word removal*)

Одређене речи су веома честе у језику, а за многе методе обраде текста не доносе никакву информацију јер немају специфично значење. Такви примери на српском језику су речи: или, су, ја, ти, онако, овако, тако,... На енглеском језику то су речи као на пример: *the, a, an, in, or*,… Речи које се уклањају се деле на три групе: одредбене речи (*the, a, an, another*), везнике (*for, but, or, yet, so*) и предлоге (*in, under, towards*) [5].

Избор речи које ће се уклонити у овом кораку зависи од касније примене и од језика. Најчешће се речи бирају тако да је њихова учесталост у језику велика, а њихова улога у разумевању и решавању проблема небитна. Ове речи се уклањају ради смањења количине текста који се складишти и обрађује, а самим тим се утиче на убрзање. Додатно, смањује се разуђеност значајног текста.

Поред основне методе за чишћење текста од честих речи, постоје и друге технике као што су:

1) З-метода заснована на Зиповом правилу (*Z-method based on Zipf's Law*),
2) метода заједничке информације (*The Mutual Information Method - MI*) и
3) случано узимање узорка засновано на терминима (*Term Based Random Sampling - TBRS*).

Пошто број речи које треба уклонити на овај начин није мали, постоје листе тих речи. На пример, *NLTK (Natural Language Toolkit)* у програмском језику *python* садржи листу честих речи у 16 различитих језика [6]. Многи алати и библиотеке имају уграђене листе ових речи, али је препоручљиво да се за специфичне примене обрати пажња које ће се речи избацити.

Према резултатима наведеним у табели 5.1 књиге [7], избацивање ових речи смањује количину текста за 30 до 50 посто у зависности од броја избачених речи. Мерење је обављено за енглески језик.

Број речи који се уклања био је велики (200 до 300 речи), међутим с временом се тај број смањивао, тако да данас постоје алгоритми код којих се ове речи не уклањају уопште [7]. Овај тренд је присутан, зато што се величина меморије и брзина којом се подаци обрађују повећавају експоненцијално, тако да коришћење ових речи не представља толики проблем. С друге стране, најновија истраживања показују да уклањање ових речи не доводи до побољшања алгоритма за пример моделовања тема [8]. Са појавом веб претраживача, корисност ових речи је постала већа, с обзиром да је из кратких упита потребно проценити његово значење и вратити најбоље одговоре. У том случају ове речи помажу да се лакше одреди контекст упита.

2.3. Одређивање основа речи (*stemming*)

Ова метода је оригинално представњена 1980. године у [9], али је исте године приказана у [10], што је најпознатија верзија овог алгоритма. Основна идеја је да речи које имају исту основу, имају слично значење. Због тога се суфикси свих речи уклањају и користе се само њихове основе чиме се смањује простор речи и количина података које треба анализирати. На пример речи представити, представа, представљање имају исту основу представ и слично семантичко значење и због тога само се каже да су оне различити облици те речи које се могу заједно представити својом основом.

Међутим, постоје случајеви у којима речи са истом основом немају слично значење, на пример речи лакирати и лакши имају исту основу лак. Због тога су настала побољшања ове методе као што су одређивање основа речи помоћу речника, при чему се води рачуна о контексту речи. Постоје и други мање коришћени алгоритми, као што су:

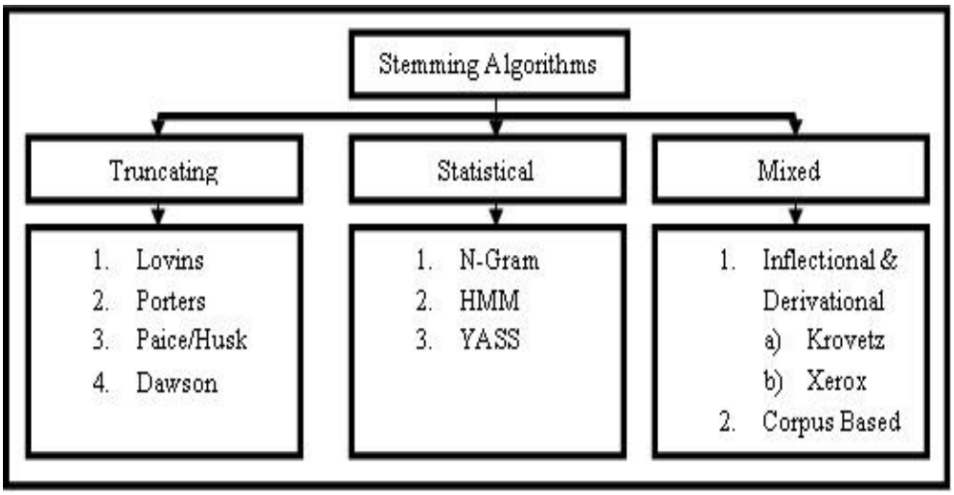
1) стохастички алгоритми,
2) анализа контекста н-грама,
3) хибридни приступи,
4) одређивање основе речи уклањањем префикса и суфикса и
5) алгоритми поклапања [11].

Подела алгоритама за одређивање основе речи може се видети на слици 2.3.1.

Након процесирања овом методом, може се десити да речи које су биле ретке имају основу која се често појављује у тексту, зато што постоје неке друге честе речи са истом основом. У неким случајевима ово је жељени излаз (нпр. код одређивања сличности појмова или реченица), док у другим случајевима где нам је врста речи и њен облик битан (нпр. при одређивању функција речи у реченици и структуре реченице) одбацивање суфикса не треба радити.

Примарна улога ове методе је била смањење потребног простора и времена за рад са текстом без обзира на губљење информација и контекста, као у случају лакши и лакирати. Међутим, неке новије методе не користе ово претпроцесирање зато што се доступни процесорски и меморијски ресурси повећавају експоненцијално и данас је битнија тачност алгоритма од његових перформанси. Треба напоменути да, као што је показано у табели у уводу у раду [10], за неке апликације не постоји велика разлика у тачности без обзира на то да ли је ова метода примењена или не, али генерално се може сматрати да ова метода смањује

прецизност и повећава комплетност (*recall*). [7] Дакле, коначна одлука о коришћењу одређивања основа речи зависи од касније примене и пошто резулати нису увек очигледни, некада треба испитати тачност система у оба случаја.



Слика 2.3.1. Подела алгоритама за одређивање корена речи [12]

Овај алгоритам је имплементиран у различитим програмским језицима. У језику *python* постоји функција библиотека *stemming* из које се може користити метода *porter2.stem*, у језику *Matlab* постоји имплементирана функција *porterStemmer*, а у језику Java постоје класе *Stemmer*, *PorterStemmer* и *Porter* које се могу наћи на вебу.

2.3.1. Опис алгоритма

У овом одељку описан је Портеров алгоритам за одређивање основе речи. Уколико се променљивом *C* обележе суседни сугласници у речи, а променљивом *V* суседни самогласници у речи, онда се реч може приказати као једна од следећих секвенци:

$$\begin{matrix} CVCV....C \\ CVCV...V \\ VCVC...C \\ VCVC...V \end{matrix} \qquad (2.3.1.1)$$

Све ове секвенце се могу описати формулом

$$[C]VCVC...C[V] \text{ или } [C](VC)^m[V] \qquad (2.3.1.2)$$

при чему угласте заграде указују да се та промељива може а не мора појавити. Променљива *m* казује колико се пута у речи појављује *VC* структура.

Када се свака реч дефинише на овакав начин, дефинише се и листа правила која указују када и код каквих структура речи треба искључити суфикс. Ова правила морају се дефинисати посебно за сваки језик. Правила су облика:

$$If\ (condition)\ S1\ then\ S1 \rightarrow S2 \qquad (2.3.1.3)$$

Уколико је испуњен услов *condition* и реч се завршава са *S1*, онда се *S1* замењује са *S2*, при чему је *S2* најчешће празна секвенца. У [10] су дефинисани услови и правила за енглески језик, тако да услов најчешће зависи од *m*, а *S1* су специфични суфикси који су интересантни за примену овог правила. Услови могу бити и комплексни, при чему се *S1* може заменити са више коњунктних/дисјунктних правила.

Одређивање основе речи, тј. сентимената за српски језик урађено је у раду [13] у коме је дат опис коришћених правила и метода машинског учења којима је то постигнуто.

2.4. Одређивање основа речи помоћу речника (*lemmatization*)

Лематизација је метода настала унапређењем претходне методе, при чему се она ослања на коришћење речника и морфолошку анализу речи како би одредила одговарајућу основу речи и њено тачно значење. За разлику од претходне методе која одсеца суфиксе само на основу сличности основе, ова метода настоји да разуме значење речи и на основу тога одлучује да ли је одсецање уопште потребно и како га урадити. [7] Укратко, лематизацију треба схватити као начин да се речи у тексту нормализују [14].

У истраживању [14], упоређиване су различите методе лематизације над словеначким језиком, који је по структури речи и типовима суфикса доста сличан српском језику. У раду су упоређене техника *if-then* правила (приказана у претходном потпоглављу) и техника мале измене правила (*Ripple-down rules*) која у *if-then* правила додаје изузетке при том не реметећи основна правила. За језике као што су српски и словеначки код којих већина правила има врло ретке изузетке, ова техника је дала убедљиво боље резултате. Тачност за технику малих измена правила је била 77,0 ± 0,6 у поређењу са стандардним *if-then* правилима за које је тачност била 62,6 ± 0,07.

У раду [15] упоређене су перформансе алгоритма за проналажење докумената када се у процесу претпроцесирања не уради одређивање основе речи, када се уради одређивање основе речи избацивањем суфикса (*stemming*) и када се уради одређивање основе речи помоћу речника (*lemmatization*). Резултати показују да је за ову апликацију лематизација у току претпроцесирања дала најбоље резултате у проналажењу 10 и 20 најбољих докумената за дати упит (0,623 и 0,544), лошији резултати су били употребом методе одређивања сентимената (0,614 и 0,510), а најгори резултати су добијени када ни једна од ове две методе није употребљена (0,601 и 0,490).

Ова метода је имплементирана у бројним алатима. Вероватно најкоришћенија је њена имплементација у *NLTK* алату у *python* програмском језику [16]. *Stanford* универзитет је направио имплементацију ове и многих других метода за рад са текстом у оквиру свог алата *CoreNLP (Core Natural Language Processing)* [17] који ради са програмским језиком *Java*, а има интерфејс и ка програмском језику *python*.

2.5. Издвајање токена (*tokenization*)

Циљ ове методе је да издвоји токене, при чему су токени најчешће речи. Ова методологија се употребљава за рад са програмским кодом и представља први корак у пројектовању преводиоца за програмске језике. У природном језику ова метода се може користити за издвајање речи како би се затим одредила њихова функција у реченици или неке друге особине речи.

Ова метода се лако може имплементирати ако се зна који знаци се налазе између токена, тј. речи. У сваком програмском језику постоје методе за поделу стринга у подстрингове ако су раздвојени одређеним знаком (најчешће се зове *split*) које се могу користити за имплементацију ове технике. Међутим, треба имати у виду специјалне случајеве приликом одређивања токена. Реч која има различите облике у једнини и множини, која се мења по падежима или глаголским временима и бројевима треба да означава исти токен у сваком од својих облика. Неке речи имају више облика, као на пример предлози ка/к и са/с, а сви ти

случајеви треба да буду представљени истим токеном. Такође, речи могу бити скраћене на различите начине, на пример, документ може да садржи једну од скраћеница САД или С.А.Д, а оба ова примера треба да представљају исти токен. На крају, треба водити рачуна о променама које настају додавањем апострофа, тако да речи ала и ал' треба да буду представљене истим токеном. Механизам којим се овакве речи проналазе и замењују истим токеном назива се нормализација. [7]

Методе за токенизацију су имплементиране и оптимизованње у одређеним бибилотекама и оквирима који су специјализовани за обраду текста, као што су *CoreNLP* [17], *OpenNLP (Open Natural Language Processing)* [18] и *NLTK* [19].

2.6. Синтаксно означавање речи (*tagging*)

Синтаксно означавање речи повезује речи са њиховим синтаксним значењем (нпр. именица, придев, глагол у природном тексту или променљива, оператор у програмском коду) [20]. Модел је настао 1996. године и у стању је да изврши мапирање на енглеском језику с тачношћу 96.6% [21]. Наравно, више речи ће имати исто синтаксно значење, а у природном тексту се може очекивати да понекад иста реч има више синтаксних значења, на пример у једном значењу је именица, а у другом глагол. Сваки језик мора имати дефинисан систем за ово мапирање. У енглеском језику је општеприхваћен *Penn Tree Bank* скуп тагова.

Ова метода се такође користи за мапирање изговорених речи и одговарајућих писаних речи, али се овај мастер рад не бави том применом.

Синтаксно означавање речи је имплементирано у бројним библиотекама које су специјализоване за обраду текста, а највише коришћене су *CoreNLP* [17], *OpenNLP* [18] и *NLTK* [19].

2.7. Груписање речи (*chunking*)

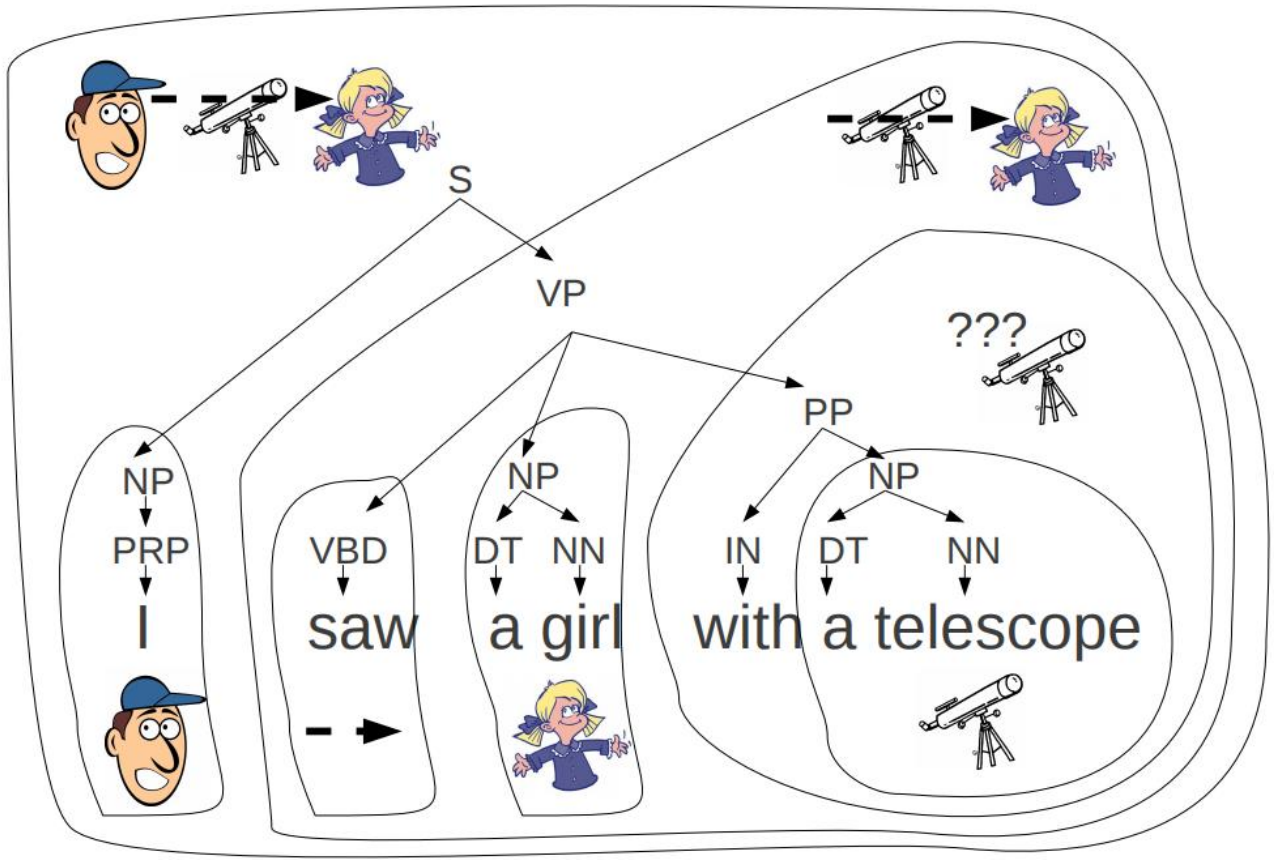
Ова методологија је слична као синтаксно означавање речи, међутим њен циљ је откривање фраза, група именица, група глагола [22,23] и сличних синтаксних целина. Један од примера овог метода је *NER (Named Entity Recognition)* модел који покушава да нађе фразе које се састоје из именица и атрибута, а заједно означавају неку особу (писац Ханс Кристијан), локацију (главни град Француске, Париз), временску одредбу, организацију и друго.

Ова метода се користи, зато што је парсирање целог текста скупа операција, а постоје случајеви у којима је одређивање фраза довољно за даљу обраду. Такође, парсирање није робустан метод. [22] У истраживању објављеном 1995. године приказан је алгоритам за одређивање група речи помоћу трансформационог учења, који је и данас један од најкоришћенијих алгоритама [24]. Он се базира на таговима додељеним речима над којима се унапред утрвиђеним правилима одређује да ли је нека група речи синтаксно слична. Правила се морају утврдити посебно за сваки језик и добро је користити помоћ лингвиста у њиховом креирању. За енглески језик, највећи скуп правила је *Penn Tree Bank*. Најбоље перформансе даје алгоритам имплементиран помоћу методе носећих вектора чија је тачност 95.77% [25].

2.8. Парсирање (*parsing*)

Улога синтаксног парсера у претпроцесирању текста јесте да одреди структуру реченица у тексту. Парсирањем се одређује улога речи у реченици, нпр. субјекат, објекат, место радње, време радње и други. Поред обраде природног текста, парсирање је обавезни корак у конструкцији преводиоца програмског кода.

За парсирање се користе тагови речи које означавају њихово значење. Парсирање се може обавити одређивањем односа зависности између речи или рекурзивним одређивањем структуре реченице. Одређивање структуре реченице се може представити као проблем предикције или као статистички проблем. [26] Пробабилистички парсери су засновани на статистици ручно парсираних реченица на основу којих се одређује највероватнија улога за неку реч у новој реченици [27]. Овакав парсер имплементиран је у *OpenNLP* [18]. Статистички модели се најчешће користе у пракси и дају добре резултате при парсирању, али имају велику временску сложеност.



Слика 2.8.1. Приказ рекурзивног парсирања реченице користећи тагове речи [24]

3. МЕТОДЕ ЗА МОДЕЛИРАЊЕ ЈЕЗИКА

Интензивна истраживања у области претпроцесирања и процесирања текста, трају више од педесет година. Зато је тешко обухватити све алгоритме у овој области, већ је аутор овог рада одлучио да прикаже основне идеје у претпроцесирању кроз кратак опис ових метода, а да се у опису процесирања текста бави превасходно најновијим и највише коришћеним методама.

У првом потпоглављу описана је техника *tf-idf* која је једна од основних техника за рад са текстом и често се користи као основа за поређење са другим алгоримима, а некад се користи и као техника претпроцесирања текста или за разумевање статистике речи у тексту. Такође, коришћење н-грама није новитет, али је истраживање у овој области још увек активно и н-грами се примењују као део нових метода и приступа, зато је та техника приказана у овом поглављу. Остале методе описане у овом тексту су базиране на дубоком учењу или методама које по резултатима могу да се упореде са методама дубоког учења као што је декомпозиција по сингуларним вредностма.

Аутор сматра да је прегледом ових технологија дат увид у историјски најбитније моделе у обради текста и да је допринесено разумевању употребе ових техника и разлика међу њима.

3.1. Метода фреквенције термина у документу и инверзне фреквенције термина у корпусу (*tf-idf*)

Ова методологија настала је као решење проблема претраживања докумената на основу упита из базе. Иако је алгоритам објављен још 1988. и данас се користи као основа за упоређивање са новим алгоритмима. Његове предности су: једноставност, могућност рада са било којим типом текста и прилично добри резултати. Често се користи и за разумевање карактеристика текста и докумената који се обрађују. Овај алгоритам не захтева знање или консултовање експерта, нити било какво лабелирање тежина, тј. релевантности речи у документу.

Пошто је рад у коме је овај алгоритам објављен из области проналажења информација из база текстова, он се оригинално бави документима и упитима. У овом мастер раду, користе се само различити документи, тако да ће се упити такође сматрати текстуалним документима приликом објашњавања алгоритма, чиме се суштина алгоритма не мења. С друге стране, да би се добила најбоља тачност у случају упита и докумената, понекад се користе различите формуле за рачунање статистике за упите и за документе, али се то неће разматрати у овом раду. Сматра се да постоји *n* термина/речи које представљају документ.

Нека је документ *Di* вектор термина $Di = (t_{i1}, t_{i2}, \dots, t_{in})$ и *Wi* је вектор тежина сваког од термина у том документу, $Wi = (w_{i1}, w_{i2}, \dots, w_{in})$, онда се сличност између докумената може представити нормализованом формулом:

$$sim(Di, Dj) = \frac{\sum_{k=1}^{n} w_{ik} * w_{jk}}{\sqrt{\sum_{k=1}^{n} (w_{ik})^2 * \sum_{k=1}^{n} (w_{jk})^2}} \quad (3.1.1)$$

Тежине термина казују значај тих речи за документ, а на основу формуле се види да се сличност између докумената дефинише као нормализовани производ значаја речи у том документу. У одређивању ове сличности постоје два проблема:

1. Који термини треба да буду укључени у репрезентацију докумената?
2. Како одредити тежине које могу да разликују битне речи за разумевање контекста?

Како би се решили ови проблеми, *tf-idf* узима у обзир фреквенцију термина унутар документа (*tf*) приликом одређивања тежине која треба да иде уз неку реч за тај документ. Међутим, узимање ове фреквенције није довољно јер многе честе речи заправо не носе много информација (речи типа: ја, није, он, је, на,...), због тога се узима у обзир и инверзна фреквенција термина у корпусу докумената (*idf*). Такође, коришћењем инверзне фреквенције термина у корпусу докумената, осигуравају се мале тежине за речи које се користе у многим документима. Овај механизам је битан при одређивању сличности између докумената и при налажењу неколико најбољих докумената. Уколико се *idf* не би користио, документ са великим бројем честих речи био би сродан са свим документима и не би се могли наћи релевантни документи. Узимајући у обзир обе ове компоненте, тежина уз реч се рачуна као производ фреквенце термина у документу и инверзне фреквенце термина у корпусу (*w = tf * idf)*. Обично се користе нормализоване тежине. [28] Скор сличности између два текста рачуна се као у формули 3.1.2.

$$sim(Di, Dj) = \sum_{t \in Di} tf_{t,d_j} * idf_t \tag{3.1.2}$$

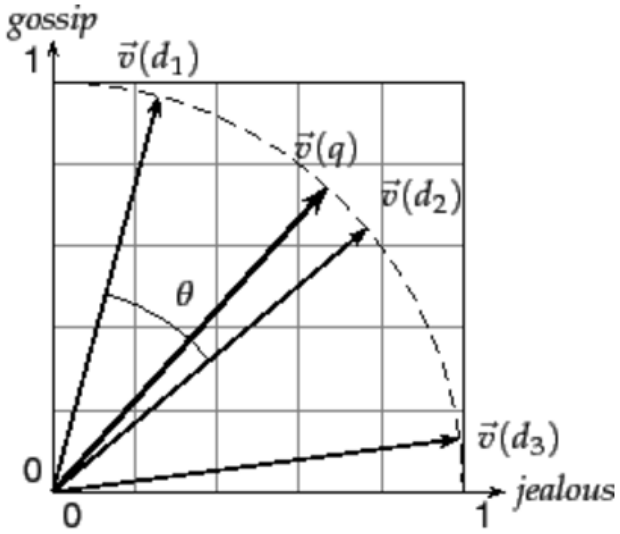
Табела 3.1.1. Типичне формуле за тежине [7]

Фреквенција термина	Фреквенција термина у корпусу докумената	Нормализација
n (природно) $tf_{t,d}$	n (нема) *1*	n (нема) *1*
l (логаритамски) $1 + log(t_{ft},d)$	t (idf) $log \frac{N}{df_t}$	c (косинусна) $\frac{1}{\sqrt{w_1^2 + w_2^2 + \cdots + w_M^2}}$
a (дупла нормализација 0.5) $0{,}5 + \frac{0{,}5 * tf_{t,d}}{max_t(tf_{t,d})}$	p (пробабилистички idf) $max\left\{0, log \frac{N - df_t}{df_t}\right\}$	u (инверзна јединствена) $\frac{1}{u}$
b (бинарно) $\begin{cases} 1, & \text{ако } tf_{t,d} > 0 \\ 0, & \text{иначе} \end{cases}$		b (величина бајта) $\frac{1}{(\text{дужина карактера})^{\alpha}}, \alpha < 1$
L (средње логаритамски) $\frac{1 + \log(tf_{t,d})}{1 + \log(avg_{t \in d}(tf_{t,d}))}$		

Користећи формулу 3.1.2. термини који се пуно пута понављају у документима D_i и D_j, а у исто време се не појављују у много докумената имаће највећу тежину. С друге стране, термини који се појављују у много докумената имаће малу тежину. [7]

Постоје различите формуле за *tf* и *idf* и њихову нормализацију које се користе у различитим ситуацијама. У обичним текстовима се на пример не очекује да нека реч буде двадесет пута јача него друга зато што се појављује 20 пута више. Због тога се за тежину радије узима логаритам броја појављивања приказан у другом реду табеле 3.1.1. Такође, најчешће коришћена формула за *idf* је логаритамска формула $log\,\frac{N}{df_t}$.

Данас се документи упоређују не само ради претраживања из базе текстова, већ и за класификцију и кластеризацију истих. Због тога се документи приказују у векторском простору (слика 3.1.1) и приликом одређивања сличности међу документима заправо се одређује скаларни производ између вектора докумената $\overrightarrow{V_{D_i}}$ и $\overrightarrow{V_{D_j}}$ (слика 3.1.1). Компоненте ових вектора могу да се рачунају на различите начине, а један од основних је коришћење *tf-idf* тежина за термине. У оваквој репрезентацији поједине реченице са другачијим значењем ће имати идентичне векторе. На пример, реченица „Марија је бржа од Јована“ и реченица „Јован је бржи од Марије“ ће имати исту векторску репрезентацију у оваквом моделу који се у литератури назива и врећа речи (*bag of words*).



Слика 3.1.1. Скаларни производ између вектора докумената [7]

У оваквом векторском простору докумената сличност два документа рачуна се по формули (при чему је · скаларни производ, тј. коначна формула представља косинусну сличност):

$$sim(Di, Dj) = \frac{\overrightarrow{V(D_i)} \cdot \overrightarrow{V(D_j)}}{\left|\overrightarrow{V(D_i)}\right| * \left|\overrightarrow{V(D_j)}\right|} \tag{3.1.3}$$

Као једна од најпопуларнијих метода у обради текста, *tf-idf* је имплементиран у различитим библиотекама за обраду текста. Међу њима најпознатије имплеметације су у *NLTK*, где постоје функције *idf(term), tf(term, text), tf_idf(term, text)* у пакету *nltk.text*. Пре употребе ових функција треба направити колекцију текстова као у примеру на слици 3.1.2. [29] У програмском језику јава постоји имплементација у класи *TFIDFCalculator* [30], а библиотека deeplearning4j такође има имплементацију за *tfidf* и за врећу речи [31]. Постоји још

и имплементација за ове методе за дистрибуирано програмирање имплементирана у јави, у библиотеци Apache Mahout [32].

```
import nltk.corpus
from nltk.text import TextCollection
from nltk.book import text1, text2, text3
gutenberg = TextCollection(nltk.corpus.gutenberg)
mytexts = TextCollection([text1, text2, text3])
tf('picture', text1)
idf('picture')
tf_idf('picture', text1)
```

Слика 3.1.2. Пример коришћења *tfidf* у библиотеци *NLTK* у програмском језику python [30]

3.2. Н-грами (*n-grams)*

Ова методологија је првобитно, у раним седамдесетим годинама, настала као алат за корекцију грешака приликом куцања и превођења [33]. Данас има и бројне друге примене:

- Детекција злонамерног кода у програмима [34]
- Откривање фраза у тексту или теме текста [35]
- Детекција плагијата у радовима, новинским текстовима или програмским кодовима [36]
- Проналажење аутора рада на основу сличности са претходним делима [37]
- Категоризација и класификација текстова [38]
- Аутоматска анотација биолошких термина [39] и екстракција шаблона из биолошких секвенци (ДНК, РНК, протеини,...) [40]
- Корекција текста насталог препознавањем говора или писања [41]
- Корекција грешака приликом срицања или машинског превођења [41]
- Креирање система за комуникацију за људе са поремећајем у говору [41]

Н-грам је секвенца од *n* јединица, при чему јединица може бити реч, слово, фонема, слог [42] и слично у зависности од модела и његове апликације. У овом раду ће сматрти да се ради с речима, осим ако није другачије наглашено. Најчешће се ради са н-грамима величине два до шест, а најпознатије су апликације са биграмима (н-грам величине два) и триграмима (н-грам величине три).

Предност ове технике у односу на претходну (tf-idf) јесте да она користи и редослед речи, дакле могуће је схватити контекст у коме се нека реч појављује тако да алгоритам разуме разлику између реченица „Марија је бржа од Јована“ и „Јован је бржи од Марије“.

3.2.1. Језички модели са н-грамима

Н-грами се сматрају пробабилистичком методом јер је помоћу њих могуће креирати пробабилистичке језичке моделе и на основу њих решавати горе наведене проблеме. Захваљујући н-грамима може се израчунати вероватноћа неке речи (нпр. „дан“) у зависности од речи које јој претходе (нпр. „данас нам је диван“) као у формули 3.2.1.1. Ова вероватноћа

се изражава као количник броја појављивања целе секвенце „данас нам је диван дан“ и појављивања предходних речи „данас нам је диван“.

$$P(\text{дан}|\text{данас нам је диван}) = \frac{C(\text{данас нам је диван дан})}{C(\text{данас нам је диван})} \quad (3.2.1.1)$$

Претходна формула се може може претворити у производ вероватноћа појављивања појединих речи у односу на речи које јој предходе користећи правило уланчавања вероватноћа:

$$P(w_1 w_2 \dots w_n) = P(w_1) * P(w_2|w_1) * P(w_3|w_1^2) * \dots * P(w_n|w_1^{n-1}) = \prod_{k=1}^{n} P(w_k|w_1^{k-1}) \quad (3.2.1.2)$$

Као што се види у формули 3.2.1.2 вероватноћа секвенце је производ вероватноћа тренутне речи у зависности од свих претходних речи. Ово израчунавање подразумева огроман број комбинација за који је потребно наћи вероватноће, а често се може десити да у датом корпусу таква секвенца не постоји (поготово за дуге секвенце), тј. да је вероватноћа једног од чиниоца 0. Идеја језичког модела н-грама је да се праћењем само неколико претходних речи и рачунањем вероватноће тренутне речи у односу на њих могу добити добре апроксимације вероватноће за тренутну реч. Код биграма се, на пример, за одређивање тренутне речи користи само претходна реч и овакав систем се назива Марковљева претпоставка (формула 3.2.1.3).

$$P(w_{\text{н}}|w_1^{n-1}) \approx P(w_{\text{н}}|w_{\text{n}-1}) \quad (3.2.1.3)$$

$$\text{P}(w_1 w_2 \dots w_n) = \prod_{k=1}^{n} P(w_k|w_1^{k-1}) \approx \prod_{k=1}^{n} P(w_{\text{k}}|w_{\text{k}-1}) \quad (3.2.1.4)$$

Захваљујући Марковљевој претпоставци се добијају много једноставније вероватноће секвенце као што се види у формули 3.2.1.4 која је само производ вероватноћа биграма за сваку реч у тој секвенци. Сличне се формуле могу извести за триграме или н-граме вишег реда, али је за њих потребно имати већи корпус података. [41]

Приликом креирања оваквих модела посебну пажњу треба обратити на н-граме чија је вероватноћа нула, тј. нису се никада до сада појавили у корпусу. Због тога је величина и разноврсност докумената који се користе за тренирање јако битна за овај алгоритам. Такође, треба имати у виду да н-грами вишег реда дају бољу прецизност што произлази директно из горе наведених формула, али је за њих потребна и већа количина података за тренирање.

Без обзира на величину корпуса увек се може десити да се у тренинг текстовима не налазе н-грами који се касније појаве у примени методе. Такође, пошто је језик жив и склон променама, могу настати нове речи или фразе. Случај нове речи се може решити игнорисањем н-грама који садрже реч која не постоји у речнику и аутоматским додавањем те речи у корпус. Проблем који настаје наиласком на н-граме који до сада нису виђени може се такође решити игнорисањем таквих н-грама, али треба користити боља решења као што су методе за углађивање (*smoothing*). Постоји више различитих приступа: додај-*1* углађивање, додај-*k* углађивање, просто повлачење, Кнесер-Нај углађивање [41], линеарна интерполација, добар Туринг, Лидстоново углађивање и Кацово повлачење [42].

Додај-*1* углађивање ради тако што се приликом рачунања вероватноће н-грама (као што је приказано у формули 3.2.1.1) додаје *1* у бројиоцу и *V* у имениоцу, при чему је *V* број речи у речнику. Овим се осигурава да вероватноћа никад неће бити нула, чак иако се неки н-грам никад није појавио у тексту. Углађена вероватноћа н-те речи у н-граму, знајући њене претходнике, се рачуна по формули 3.2.1.5 на основу додај-*1* углађивања.

$$P(w_{\text{n}}|w_1 w_2 \dots w_{\text{n}-1}) = \frac{C(w_1 w_2 \dots w_{\text{n}})+1}{C(w_1 w_2 \dots w_{\text{n}-1})+V} \quad (3.2.1.5)$$

3.2.2. Имплементација и примена н-грама

Пре имплементације н-грама углавном се ради претпроцесирање текста којим се уклањају знакови интерпункције, велика слова са почетка реченица и врши се токенизација речи (у случају н-грама где је основна јединица реч). Додатно, пре токенизације може се извршити уклањање суфикса. Постоји имплементација н-грама у различитим библиотекама и програмским језицима и махом све представљају углађену верзију н-грама јер је она најбоља за даљу примену. У *CoreNLP* библиотеци постоји *getNgrams* функција у класама *StringUtils* и *CollectionUtils*. У *OpenNLP* библиотеци ради се токенизација, па онда одређују н-грами кодом који је приказан на слици 3.2.2.1. [43] У NLTK библиотеци функција *ngrams* се налази у *util* модулу [29].

```
package com.denismigol.examples.nlp;

import opennlp.tools.ngram.NGramModel;
import opennlp.tools.tokenize.WhitespaceTokenizer;
import opennlp.tools.util.StringList;

/**
 * @author Denis Migol
 */
public class OpenNlpNGramDemo {
    public static void main(String[] args) {
        String text = "This is an example text for n-gram";
        System.out.println(text);

        StringList tokens = new StringList(WhitespaceTokenizer.INSTANCE.tokenize(text));
        System.out.println("Tokens: " + tokens);

        NGramModel nGramModel = new NGramModel();
        nGramModel.add(tokens, 2, 3);

        System.out.println("Total ngrams: " + nGramModel.numberOfGrams());
        for (StringList ngram : nGramModel) {
            System.out.println(nGramModel.getCount(ngram) + " - " + ngram);
        }
    }
}
```

Слика 3.2.2.1. Пример коришћења н-грама из *OpenNLP* библиотеке [43]

Google има пројекат скенирања књига широм света како би се дигитализовало знање и корисницима омогућила претрага књига. Због тога су развили Google Ngram Viewer, алат за онлине претрагу који израчунава фреквенцу било ког стринга у било којој години користећи н-граме из скенираних књига које датирају из времена 1500-2008. године и написане су на различитим језицима (енглески, једноставни кинески, француски, немачки, италијански, руски, шпански, хербејски) [44].

Н-грами код којих су основне јединице слова, базни парови или слогови су данас веома коришћени за разумевање ДНК, РНК и протеинских секвенци. Користе се за нелабелирано препознавање биолошких ентитета у којима је тешко лоцирати крајеве ентитета због њихове двосмислености. Такође, н-грами могу препознати семантичке односе између биолошких ентитета у научним радовима, што омогућава аутоматско креирање и анотацију биолошких процеса и попуњавање база података. [39, 40]

У овом раду је највише од интереса примена н-грама у категоризацији текстова [38] и откривања тема категорија [35]. Приликом категоризације докумената користе се статистичке карактеристике н-грама који се налазе у тим текстовима, тј. сваком н-граму из тог текста се припише вероватноћа појављивања у том тексту. На тај начин се зна који н-грами се колико често појављују у текстовим из одређене категорије. Потом се за сваки нови текст рачунају ове вероватноће и њему се додељује категорија у којој се налазе документи који имају

најсличније вероватноће за н-граме. У алгоритму у раду [38] категорија се додељује на основу минималног растојања профила (статистике н-грама) новог документа и профила постојећих категорија. На сличан начин се, уместо категорија, могу одредити аутори дела [37]. Специфични делови кода који могу бити потенцијално опасни [34] или делови текста који су преписани [36] могу се одредити упоређивањем статистике н-грама текста/програмског кода део по део са статистиком целог документа или корпуса.

Све ове методе захтевају лабелирање, али могу се развити и системи са нелабелираним текстовима. За кластеризацију текстова је онда потребно одредити статистике н-грама за те документе и на основу њих кластеризовати документе у категорије и одредити теме сваке од категорија. Теме се могу одредити као скуп од једног или више н-грама чија је вероватноћа појављивања у овој категорији много већа него у другим категоријама.

Кластеризација текстова је проблем којим се и овај рад бави у 4. и 5. поглављу, међутим у овом раду се користе новије технике, базиране на неуралним мрежама, за које се показало да дају боље резултате и имају боље временске перформансе. Наиме, пре настанка метода базираних на неуралним мрежама, н-грами су били најкоришћенији за моделовање језика, али су уједно били и велики проблем за све апликације због спорости.

3.3. Декомпозиција по сингуларним вредностима (*Singular value decomposition - SVD*)

Обе до сада приказане технике за моделирање језика имају проблем са великом бројем димензија у којој су документи приказани. Док је код *tf-idf* могуће избећи коришћење свих могућих речи из речника избором подскупа речи, код н-грама се претпоставља коришћење свих н-грама у документу, што утиче на брзину извршавања и применљивост алгоритма за велике податке. Због велике разноликости у језику, речници могу да садрже и милионе речи и много већи број н-грама (теоретски V^n, где је V број речи у речнику), при чему се у основном људском језику користи много мањи број ових комбинација.

Имајући у виду спорост и високодимензионост постојећих модела који су уз то и веома прорећени (за многе димензије вредности су приближно нула, тј. многе речи се не користе или се користе врло ретко) било би добро осмислити систем који би имао далеко мању димензионост, па самим тим много боље временске перформансе и то без губитка тачности. Ова и наредне методе представљају такве моделе.

Ако се језички модел прикаже као матрица код које реч представља ред, а документ представља колону и у њиховом пресеку се налази број појављивања речи у документу (табела 3.3.1), онда се редукција димензије такве матрице може добити декомпозицијом. Најчешће коришћени модели декомпозиција за овај проблем су *QR* факторизација и SVD декомпозиција. при чему је за другу методу показано да може да има перформансе као и најбољи постојећи алгоритми и због тога је у овом раду описана примена SVD декомпозиције. У зависности од проблема матрица језичког модела може имати и другачије колоне и врсте као што су н-грами, фразе, наслови, упити, реченице, параграфи и слично. [45]

Декомпозиција овако насталих матрица може се сматрати начином за налажење простора са мање димензија у коме се једнако добро може представити постојећи језички систем. Предност оваквог модела је у томе што број димензија може бити пуно мањи (и хиљадама пута мањи), а мана је то што човек не може да интерпретира нове димензије и не разуме њихово значење, за разлику од основних димензија.

Табела 3.3.1. Пример матрице језичког модела између докумената и речи (матрица *A*)

	Документ 1 (D_1)	Документ 2 (D_2)	Документ 3 (D_3)	Документ 3 (D_4)
Грешка	0	1	2	0
Нетачно	0	0	1	0
Порука	2	0	3	0
Форматирање	0	1	0	0
Неспособност	0	0	0	1
Компјутер	1	0	0	0

3.3.1. Теоретске основе декомпозиције по сингуларним вредностима

Дефиниција 3.3.1.1. Сингуларне вредности матрице A димензија m x n су корени сопствених вредности симетричне матрице A^TA димензија n x n, тако да су сопствене вредности поређане у опадајућем редоследу. [46]

Дефиниција 3.3.1.2. Нека је дата комплексна матрица U, њена конјугована матрица U^* и јединична матрица I. Уколико важи једначина 3.3.1.1, онда је U унитарна матрица.

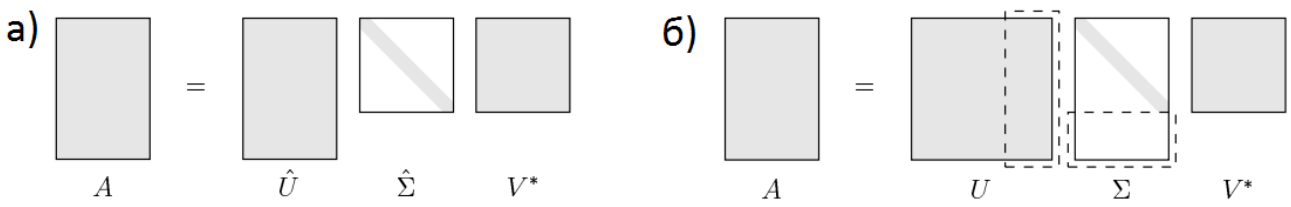
$$U^*U = UU^* = I \tag{3.3.1.1}$$

Дефиниција 3.3.1.3 Нека су m и n произвољне вредности и матрица $A \in C^{mxn}$, *о*нда је декомпозиција по сингуларним вредностима матрице A дата формулом 3.3.1.2. при чему важи да је $U \in C^{mxm}$ унитарна матрица, $V \in C^{nxn}$ је унитарна матрица и $\sum \in R^{mxn}$ је дијагонална матрица. [47]

$$A = U\sum V^T \tag{3.3.1.2}$$

У даљем разматрању сматраће се да је m веће или једнако са n ($m \geq n$). У случајевима када не важи једнакост између m и n, разликују два типа декомпозиције по сингуларним вредностима: потпуна и редукована. Пошто је $\sum$ дијагонална матрица, а $m \geq n$, то значи да се $\sum$ може представити као $\sum \in R^{mxm}$, а самим тим U има димензије mxn (Слика 3.3.1.1. а)). Оваква декомпозиција се зове редукована декомпозиција. Да би се добила потпуна декомпозиција $\sum$ матрици се додаје m-n редова код којих су све вредности 0, а матрици U се додаје m-n врста, тј. вектора који су ортонормални са постојећим векторима у U (Слика 3.3.1.1. б)). [47]

Редукована декомпозиција се користи у обради текста и на даље у овом документу ће се под декомпозицијом матрице по сингуларним вредностима сматрати редукована декомпозиција.



Слика 3.3.1.1. а) Редукована; б) Потпуна декомпозиција по сингуларним вредностима

3.3.2. Пример декомпозиције по сингуларним вредностима

У апликацији декомпозиције по сингуларним вредностима у обради текста, матрица A је увек реална матрица, те су матрице U и V заправо ортогоналне, а не унитарне матрице. Пошто се у дефиницији не захтева да матрица A има потпуни ранк треба имати у виду да постоји ранг матрице r, тако да је $r \leq n$. У том случају димензије матрица се могу редуковати у $U \in C^{mxr}$, $V \in C^{rxn}$ и $\sum \in R^{rxr}$. У овом случају U је ортогонална матрица чије колоне чине леве сингуларне векторе, V је ортогонална матрица чије колоне чине десне сингуларне векторе матрице A, а $\sum$ је дијагонална матрица на чијој дијагонали су сингуларне вредности матрице A. [45]

Овако дефинисана матрица V је скуп сопствених вектора коваријансне матрице A^TA (једначина 3.3.2.1), а матрица U је скуп сопствених вектора унутрашњег производа AA^T (једначина 3.3.2.2).

$$A^TA = (U\sum V^T)^T(U\sum V^T) = V\sum^2 V^T \quad (3.3.2.1)$$

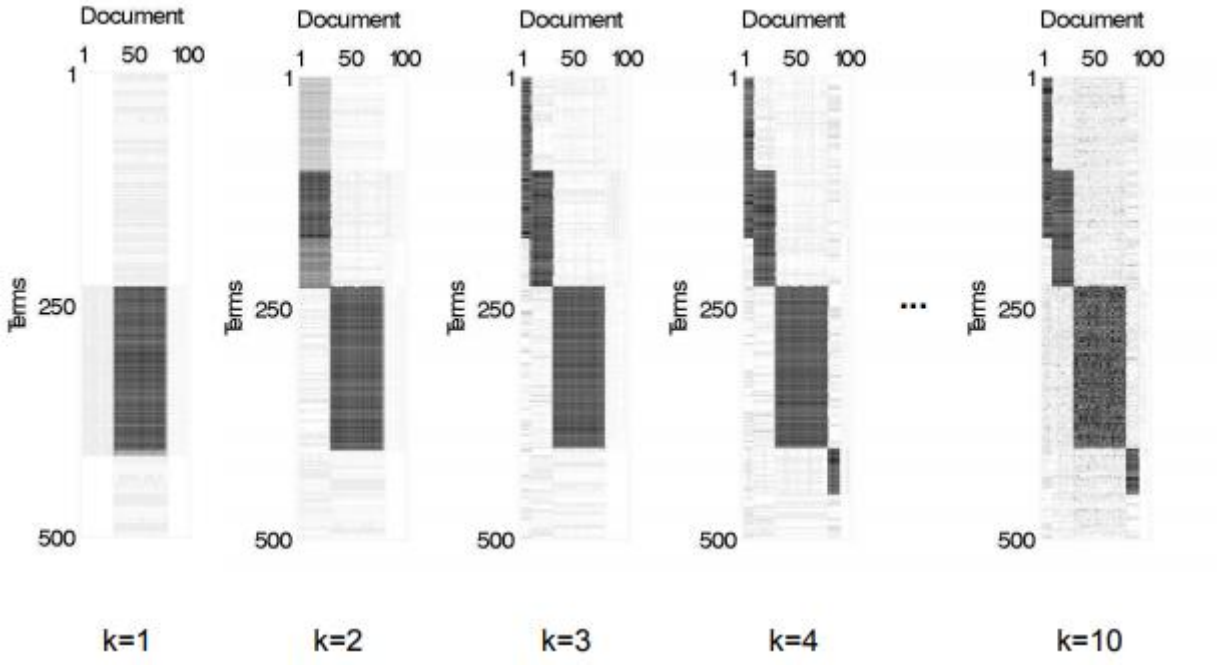
$$AA^T = (U\sum V^T)(U\sum V^T)^T = U\sum^2 U^T \quad (3.3.2.2)$$

Као што се може видети из претходних једначина декомпозиција матрице по сопственим вредностима је уско повезана са сопственим векторима, а самим тим и са техникама за селектовање одлика као што је *PCA* (*Principal Component Analysis*). [48]

При коришћењу декомпозиције по сингуларним вредностима, у обради текста је такође пожељно искористити карактеристике ове методе како би се смањила димензионост. Пошто се захваљујући овој методи матрица A, која има ранк r може представити као у једначини 3.3.2.3, тј. као сума матрица ранга један и како су сингуларне вредности поређане у опадајућем редоследу, онда је матрицу A могуће апроксимирати матрицом мањег ранга k (једначина 3.3.2.4).

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^T \quad (3.3.2.3)$$

$$A \approx A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T = U_k \sum_k V_k^t \quad (3.3.2.4)$$

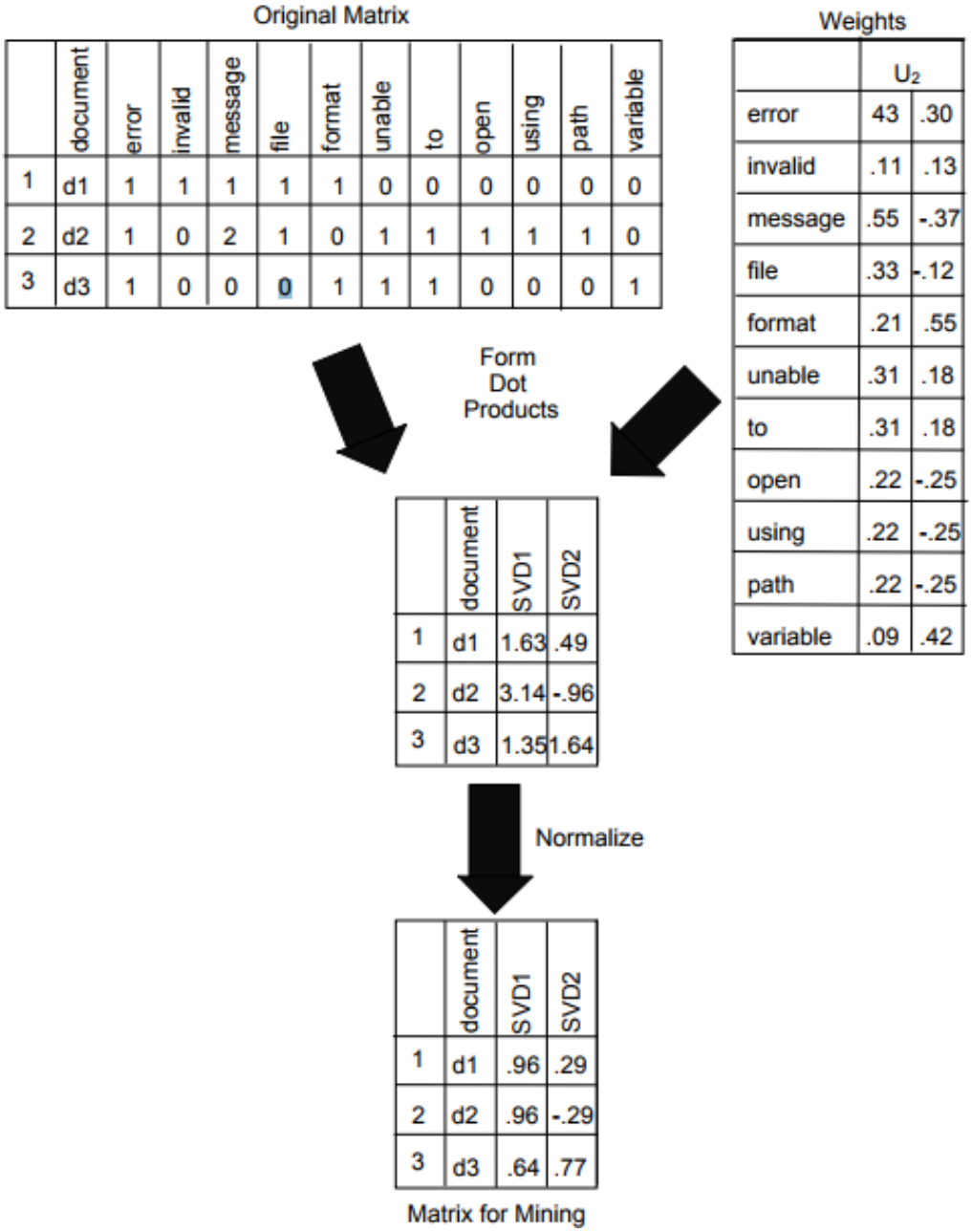


Слика 3.3.2.1. Апроксимација матрице A матрицом A_k за различите вредности k [45]

Може се показати да је A_k најприближнија апроксимација матрице A која има ранг k, тј. да је грешка која настаје приликом смањења ранга матрице на k најмања могућа за матрицу A_k. Због тога се ова метода користи приликом смањивања димензије у обради текста. У том случају се обично узима да је k много мање од n ($k<<n$). Апроксимација која се добија на овај начин се може видети на слици 3.3.2.1. за различите вредности k.

Уколико овакву имплементацију, пре свега формулу 3.3.2.3, применимо на матрицу A приказану у табели 3.3.1, онда су *Uk* и *Vk* ортонормалне основе за *k*-димензиони документ и простор термина, редом. Пошто је сваки документ иначе био представљен у m димензија, овај поступак представља пројекцију на k димензија што се може описати формулом 3.3.2.5.

$$\vec{d}_k = U_k^T \vec{d}_n \tag{3.3.2.5}$$



	document	error	invalid	message	file	format	unable	to	open	using	path	variable
1	d1	1	1	1	1	1	0	0	0	0	0	0
2	d2	1	0	2	1	0	1	1	1	1	1	0
3	d3	1	0	0	0	1	1	1	0	0	0	1

	U_2	
error	43	.30
invalid	.11	.13
message	.55	-.37
file	.33	-.12
format	.21	.55
unable	.31	.18
to	.31	.18
open	.22	-.25
using	.22	-.25
path	.22	-.25
variable	.09	.42

	document	SVD1	SVD2
1	d1	1.63	.49
2	d2	3.14	-.96
3	d3	1.35	1.64

	document	SVD1	SVD2
1	d1	.96	.29
2	d2	.96	-.29
3	d3	.64	.77

Слика 3.3.2.2. Приказ редукције димензија документа помоћу декомпозиције матрице по сингуларним вредностима [45]

Овако добијене репрезентације документа морају се нормализовати на крају. Треба још напоменути да је пре примене декомпозиције по сингуларним вредностима, могуће урадити скаларни производ између матрице A и матрице тежина за сваку реч. Ове тежине се обично генеришу као функције фреквенције тих речи у целој колекцији. Цео поступак је сажет на слици 3.3.2.2. [45]

3.3.3. Латентно семантичко индексирање (LSI – Latent Semantic Indexing)

Латентно семантичко индексирање је једна од стандардних метода за редукцију у проналажењу информација и она се ослања на декомпозицију по сингуларним вредностима. Овом методом се одређује семантичка структура, тј. веза између термина и докумената, на вишем нивоу, што значи да се помоћу ње могу уклонити двосмислености природног језика као што су синоними. Због тога је латентно семантичко индексирање постало популарно у проналажењу информација. [49]

Као што је описано у раду [50], ова метода користи чињеницу да су сингуларне вредности поређане у опадајућем редоследу, па се због тога редукцијом димензија очувавају битне разлике, а губе мале разлике међу терминима, као што су редундантност, двосмисленост и сличне карактеристике које су непожељне при претраживању докумената. У овом раду је показано да се за листу наслова текстова, може одредити векторски простор у коме су слични термини међусобно близу (слика 3.3.3.1), што је идеја и многих модернијих алгоритама. Пре примене латентног семантичког индексирања, аутори рада [50] су применили неке од техника претпроцесирања, као што су уклањање интерпункције и честих речи. Међутим, у раду није коришћено одређивање основе речи зато што се очекује да је сам алгоритам у стању да нађе сличност између речи са истом основом у случају да је њихов контекст исти.

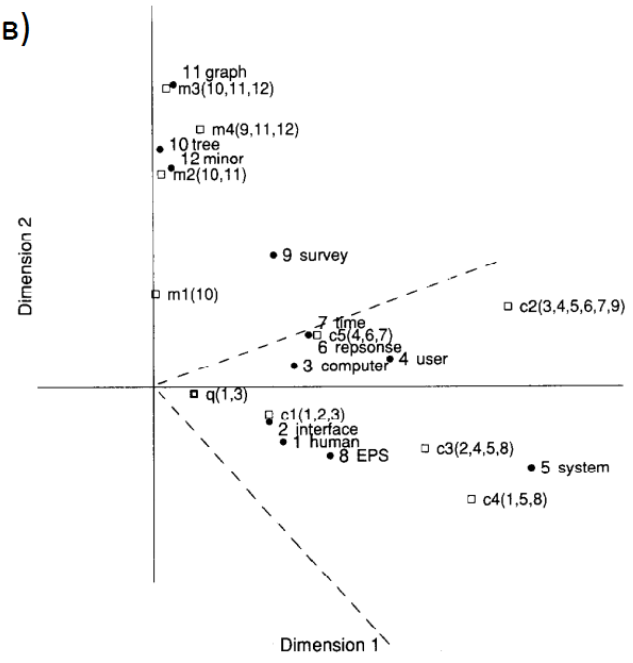
а)

	Titles
c1:	*Human* machine *interface* for Lab ABC *computer* applications
c2:	A *survey of user* opinion of *computer system response time*
c3:	The *EPS user interface* management *system*
c4:	*System* and *human system* engineering testing of *EPS*
c5:	Relation of *user*-perceived *response time* to error measurement
m1:	The generation of random, binary, unordered *trees*
m2:	The intersection *graph* of paths in *trees*
m3:	*Graph minors* IV: Widths of *trees* and well-quasi-ordering
m4:	*Graph minors: A survey*

б)

	Documents								
Terms	c1	c2	c3	c4	c5	m1	m2	m3	m4
human	1	0	0	1	0	0	0	0	0
interface	1	0	1	0	0	0	0	0	0
computer	1	1	0	0	0	0	0	0	0
user	0	1	1	0	1	0	0	0	0
system	0	1	1	2	0	0	0	0	0
response	0	1	0	0	1	0	0	0	0
time	0	1	0	0	1	0	0	0	0
EPS	0	0	1	1	0	0	0	0	0
survey	0	1	0	0	0	0	0	0	1
trees	0	0	0	0	0	1	1	1	0
graph	0	0	0	0	0	0	1	1	1
minors	0	0	0	0	0	0	0	1	1

в)



Слика 3.3.3.1. а) Наслови текстова које треба претражити; б) Табела учестаности термина у насловима; в) Груписање термина помоћу латентног семантичког индексирања [50]

Студије су показале да је латентна семантичка анализа је у стању да схвати сличности између термина на исти начин као људи. На тестовима у вези речника, познавања тема и њихових сличности, ова метода има једнако добре перформансе као човек. Она опонаша начин

на који људи сортирају и категоришу речи и тачно процењује кохерентност параграфа, могућност учења параграфа од стране студената и квалитет и квантитет знања садржаних у есејима. [51]

Ова техника није статистичка што значи да се помоћу ње не могу направити генератори, нити објаснити разлози за груписање речи. Због тога је у раду [52] развијено пробабилистичко латентно семантичко индексирање. Овај модел представља генерализацију алгоритма за максимизацију очекивања и може да схвати синониме и вишезначне речи као и латентно семантичко индексирање засновано на декомпозицији по сингуларним вредностима. Међутим, овај алгоритам такође дефинише одговарајући генеративни модел и на основу резултата у раду [52] може се видети да има боље перформансе.

3.3.4. Примена декомпозиције по сингуларним вредностима

Као што је напоменуто у опису претходних техника, једном изграђен језички модел може се користити за класификовање и кластеризацију докумената. У раду [53] аутори су мерили успешност кластеризације пре и после коришћења декомпозиције по сингуларним вредностима за грађење језичког модела. Резултати показују да је успешност кластеризације била боља уколико је у језички модел укључена декомпозиција по сингуларним вредностима, што значи да је овај приступ омогућио да се унутар кластера нађу документи који су сличнији међу собом.

Како је декомпозиција по сингуларним вредностима додатни корак у грађењу језичког модела, мора се поставити питање временске комплексности алгоритма. Сложеност овог метода је $O(n*n*m)$ при чему се сматра да је $m \geq n$.

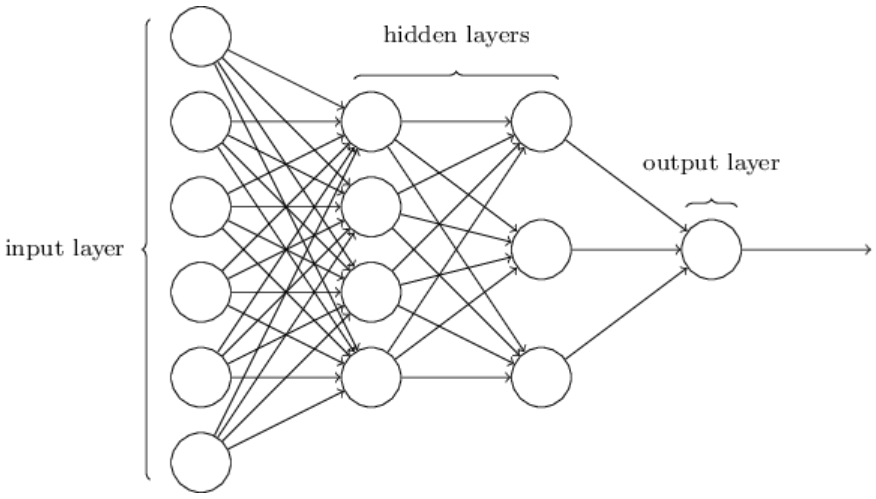
Декомпозиција сингуларних вредности је метод декомпозиције матрице и део је скоро свих стандардних библиотека за линеарну алгебру, тако да је његова имплементација доступна у многим програмским језицима (*python, java, mathematica, matlab, R*). [48] У бројним језицима се такође може наћи имплементација латентне семантичке анализе на пример у програмском језику *python* она је део модула *numpy.linalg*.

3.4. Креирање језичких модела помоћу неуралних мрежа

Иако су неуралне мреже теоретски настале још у 1980-им годинама, њихова примена је постала могућа тек почетком 21. века захваљујући експоненцијалном напретку у развоју брзине процесора и величине меморије компјутера. Вероватно највећи напредак неуралне мреже су донеле обради слика, говора и текстова, а први неурални језички модел био је развијен на обичним неуралним мрежама. У остатку овог поглавља (потпоглавња 3.4-3.7) биће представљени језички модели имплементирани у различитим врстама неуралних мрежа. При опису језичких модела у наредним потпоглављима биће прво речи о структури и основама неуралне мреже у којој је језички модел имплементиран.

3.4.1. Неуралне мреже (NN – neural networks)

Прве неуралне мреже (*feed-forward neural networks*) су се састојале од улазног, скривених и излазног слоја (нивоа). Најједноставније структуре неуралних мрежа садрже један скривени слој, при чему су неурони повезани само између суседних слојева. Данас постоје и примењују се много сложеније структуре неуралних мрежа, а неке од њих дате су у наредним одељцима. Пример неуралних мрежа дат је сликом 3.4.1.1.



Слика 3.4.1.1. Пример неуралне мреже [54]

Неуралне мреже у сваком неурону садрже активациону функцију која за одређени улаз добијен линеарном комбинацијом неурона у претходном слоју, даје излаз који се користи за рачунање следећег нивоа. Уколико је активациона функција линеарна, онда неуралне мрежа заправо ради слично као линеарна регресија. Најчешће коришћене активационе функције су логистичка (*sigmoid/logistic*) приказана једначином 3.4.1.1, тангенс-хиперболичка (*tanh-hiperbolic tangens*) приказана једначином 3.4.1.2, исправљена линеарна функција (*ReLu – rectified linear unit*) приказана једначином 3.4.1.3 и софтмакс (*softmax*) функција приказана једначином 3.4.1.4. Пошто је ReLu функција веома слична линеарној, она се сме користити само у скривеним слојевима како се неуралне мрежа не би свела на линеарну регресију. С друге стране, ова функција је популарна зато што убрзава конвергенцију и код ње не постоји проблем с ишчезавајућим градијентом.

$$f(x) = \frac{1}{1+e^{-x}} \tag{3.4.1.1}$$

$$f(x) = \frac{1-e^{-2x}}{1+e^{-2x}} \tag{3.4.1.2}$$

$$f(x) = \max(0, x) \tag{3.4.1.3}$$

$$f(z_j) = \frac{e^{z_j}}{\sum_{k=0}^{K} e^{z_k}} \tag{3.4.1.4}$$

3.4.2. Алгоритам пропагације у назад

Многе неуралне мреже користе овај алгоритам за учење, па је он зато описан у овом одељку. Сврха овог алгоритма је да се одреде тежине веза између неурона које ће за дате улазе предвидети најбољи излаз. Процес се назива учење зато што се неуралној мрежи задаје велика количина улаза за које су познати излази и на тај начин се она тренира, тј. одређују се тежине веза између неурона помоћу којих се касније од улаза добија одговарајући излаз за сваки пример из тренинг скупа. Тренинг скуп је пар вектора $(\overrightarrow{x_i}, \overrightarrow{y_i})$ у ком је унапред познат $\overrightarrow{y_i}$ за сваки вектор $\overrightarrow{x_i}$. Једном истренирана мрежа може да предвиди излазе за нове улазе који нису били део тренинг скупа.

Алгоритам пропагације у назад се састоји из два корака: пропагација у напред (*forward propagation*) и пропагација у назад (*backpropagation*). Током креирања архитектуре неуралне мреже задају се почетне тежине, које најчешће добијају вредности случајне величине између 0 и 1. Затим се за сваки пар вектора $(\overrightarrow{x_i}, \overrightarrow{y_i})$ ради корак пропагације у напред и корак

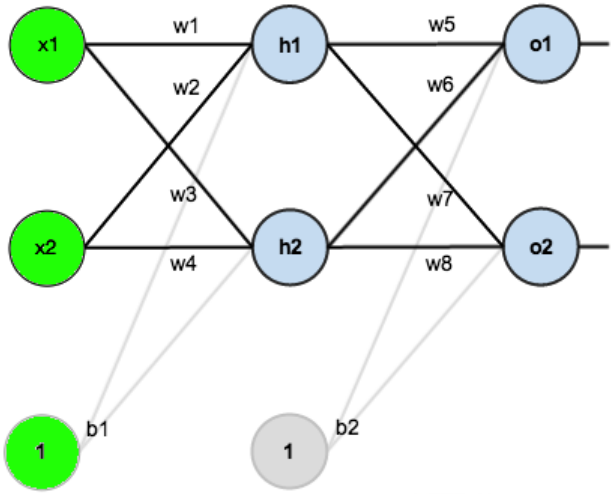
пропагације у назад чиме се тежине мењају све док њихове вредности не конвергирају, тј. док грешка коју добијамо између предвиђеног и стварног излаза не постане довољно мала. Тада је мрежа истренирана и може се тестирати над скуповима код којих излаз није познат.

Цео алгоритам пропагације у назад има сврху да се користећи постојеће тежине направи предвиђање излаза како би се израчунала грешка између предвиђања и стварног излаза (корак пропагације у напред). Затим се врши корак пропагације у назад са сврхом да се тежине веза између неурона промене како би се смањила грешка између предвиђеног и стварног излаза добијена приликом пропагација у напред.

$$h = \sigma_h(W_{hx}x + b_h) \quad (3.4.2.1)$$

$$o = \sigma_y(W_{yh}h + b_y) \quad (3.4.2.2)$$

Пропагација у напред се математички може описати формулама 3.4.2.1. и 3.4.2.2, где су W_{hx}, x, W_{yh}, h, o вектори, x представља улаз, W_{hx} тежине између улазног и скривеног слоја, o представља предвиђени излаз, W_{yh} тежине између скривеног и излазног слоја, σ_h, и σ_y су активационе функције улазног и скривеног слоја, а b_h и b_y представљају померај (*bias*) основног и скривеног слоја, редом.



Слика 3.4.2.1. Неурална мрежа са означеним тежинама [55]

За најједноставнији пример неуралне мреже (слика 3.4.2.1) пропагација унапред може се представити формулама 3.4.2.3 и 3.4.2.4 којима се рачунају вредности скривеног слоја, при чему је n_1 број неурона у улазном слоју, а b_1 је померај улазног слоја. Еквивалентне формуле би се користиле за наредне скривене слојеве и излазни слој, уколико такви постоје, осим што би улазне параметре и померај улазног слоја заменили параметри и померај из претходног скривеног слоја.

$$net_{hi} = b_1 + \sum_{j=1}^{n1} w_j * x_j \quad (3.4.2.3)$$

$$out_{hi} = \sigma(net_{hi}) = \frac{1}{1+e^{-net_{hi}}} \quad (3.4.2.4)$$

Када су израчунате скривене и предвиђене излазне вредности, треба одредити грешку предвиђања помоћу једначине 3.4.2.5. Затим се прелази на корак пропагације у назад чији је циљ да се промене тежине веза између неурона како би се смањила грешка за будућа предвиђања. То значи да треба минимизовати грешку коју сваки неурон у излазном слоју прави, зато што је коначна грешка збир грешака свих излазних неурона. Након тога се прелази

на претходни слој у ком се такође мењају тежине како би се смањила коначна грешка и поступак се понавља све док се не стигне до улазног слоја.

$$\mathrm{E} = \frac{1}{2n}\sum_i ||o(i) - y(i)||^2 \tag{3.4.2.5}$$

Промене које треба извршити над тежинама добијају се као изводи грешке по тој тежини као што је приказано у једначинама 3.4.2.6 до 3.4.2.8, где је w_{yhkl} тежина гране која повезује *k*-ти неурон у излазном слоју и *l*-ти неурон у скривеном слоју. Ову вредност је лако израчунати за излазни слој, али не и за претходне слојеве. Међутим, због веза између слојева сваки слој се може представити уланчавањем већ познатих извода грешке њихових следбенничких слојева. На пример, извод грешке неурона у скривеном слоју најближем излазном слоју, може се представити уланчавањем извода грешке неурона у излазном слоју и извода тежине скривеног и излазног слоја. Ово је приказано у једначинама 3.4.2.9 и 3.4.2.10. при чему w_{hxkl} повезује *k*-ти неурон из скривеног слоја и *l*-ти неурон у улазном слоју. У овом случају се претпоставља да постоји само један скривени слој, али би једначине биле сличне и за мрежу са више скривених слојева. У овим једначинама се такође сматра да је активациона функција у свим слојевима логистичка функција.

$$\frac{\partial \mathrm{E}}{\partial w_{yhkl}} = \frac{\partial \mathrm{E}_k}{\partial out_k} * \frac{\partial out_k}{\partial net_k} * \frac{\partial net_k}{\partial wyhkl} \tag{3.4.2.6}$$

$$\frac{\partial \mathrm{E}_k}{\partial w_{yhkl}} = \left(2 * \frac{1}{2}(out_k - y_k) + 0\right) * \left(out_k(1 - out_k)\right) * \left(1 * \ out_l * \ {w_{yhkl}}^{1-1} + 0 + 0\right) \tag{3.4.2.7}$$

$$\frac{\partial \mathrm{E}_k}{\partial w_{yhkl}} = (out_k - y_k) * out_k * (1 - out_k) * out_l \tag{3.4.2.8}$$

$$\frac{\partial \mathrm{E}_{total}}{\partial w_{hxkl}} = \frac{\partial \mathrm{E}_{total}}{\partial out_{hk}} * \frac{\partial out_{hk}}{\partial net_{hk}} * \frac{\partial net_{hk}}{\partial w_{hxkl}} \tag{3.4.2.9}$$

$$\frac{\partial \mathrm{E}}{\partial w_{hxkl}} = \left(\sum_{i=1}^{n_y} \frac{\partial E_i}{\partial out_{hk}}\right) * \left(out_{hk}(1 - out_{hk})\right) * (x_l) \tag{3.4.2.10}$$

Када се израчунају ови изводи, онда се морају променити тежине као што је приказано у формули 3.4.2.11, при чему је α коефицијент учења који је у (0,1], а извод грешке по тежини се рачуна на неки од претходно приказаних начина.

$$\mathrm{w} = \mathrm{w} - \ \alpha * \frac{\partial E}{\partial w} \tag{3.4..2.11}$$

Када су све тежине промењене, овај алгоритам се извршава за нови пар улаза и излаза све док се не искористе сви парови из тренинг скупа или док се не постигне грешка која је мања од неке предефинисане вредности (онда се сматра да је алгоритам конвергирао). Коефицијент α је опциони, његова улога је да смањи утицај одређеног пара улаза и излаза на укупни резултат. Тиме се постиже спорија конвергенција, али и мање осцилирање. Често се поставља α = 0,5.

Како је јачала моћ компјутера, тако су и модели неуралних мрежа постајали већи, па је данас популарна област дубоког учења која користи различите типове неуралних мрежа са великим бројем слојева како би што тачније научила машину да ради ствари које човек може, као што су препознавање и описивање слика, превођење текста и многи други проблеми.

Међутим, коришћење дубоких мрежа доводи до великог броја чиниоца у уланчаним рачунањима извода грешке по тежинама. Како су ови градијенти бројеви у интервалу (-1, 1), онда се њиховим уланчавањем коначни производ смањује. Код неуралних мрежа са великим бројем слојева тежине међу првим слојевима мењају се мало јер су њихови градијенти веома мали због уланчавања. Овај проблем се назива исчезавање градијента (*vanishing gradient*), а постоји и код рекурентних неуралних мрежа (без обзира на број слојева које оне имају).

3.4.3. Неурални пробабилистички језички модел

Као што је приказано код језичког модела базираног на н-грамима његова димензионост је сувише велика и главни је разлог спорости модела. Такође, постоје бројни проблеми који настају због промењљивости језика и самим тим наиласка на нове н-граме и нове речи. Разрешавање оваквих случајева је ограничено. Трећи проблем н-грама је чињеница да се помоћу њих не могу одредити синоними или вишезначне речи. На крају, код н-грама се у обзир узима само пар речи пре речи коју посматрамо. Због тога се трагало за новим решењима која се могу поделити на три гране: декпомозиција матрица, коришћење разних неуралних мрежа и статистички модели. Како је у претходном потпоглављу описан модел декомпозиције матрица у овом и наредним потпоглављима представљају се различити неурални језички модели. Важна разлика између неуралног пробабилистичког језичког модела и латентног семантичког индексирања је да се у неуралном моделу заједно учи репрезентација одлика речи и статистички модел.

Прво ће бити представљен најједноставнији модел који се ослања на основне неуралне мреже и за који је развијен и статистички модел.

Идеја овог модела се може сумирати у три корака:

1. Представити сваку реч у речнику вектором одлика у простору R^m
2. Пронаћи заједничку функцију вероватноће секвенце речи у зависности од вектора одлика речи у тим секвенцама
3. Научити истовремено векторе одлика речи и параметре за функцију вероватноће

Вектор одлика представља различите аспекте речи, а број одлика може бити различит, али је пуно пута мањи од укупног броја речи у речнику. У раду [56] у ком се предлаже ова методологија урађени су експерименти за m = 30, 60 и 100, што је знатно мање од 17000 речи колико има цели речник у њиховом случају. Функција вероватноће је производ условних вероватноћа за тренутну реч имајући у виду претходне речи. Овако изражена функција вероватноће може се максимизовати користећи логаритамски количник вероватноћа над тренинг подацима. Вектори одлика се уче, али се такође могу изразити на основу претходних сазнања.

У представљеном моделу се очекује да сличне речи имају сличне векторе одлика, тј. да буду близу међусобно у m димензионом простору. Ово је последица чињенице да је функција вероватноће глатка функција вектора одлика. Због тога мала промена у вектору одлика има за последицу малу промену у вероватноћи. У овом мастер раду фокус је на методологијама које уче језичке моделе у зависности од дистрибуција речи или секвенца речи, док постоје и бројни приступи који се баве разумевањем улога речи у реченици. Због тога је у овом одељку описан рад [56].

Циљ овог рада је да се научи функција 3.4.3.1, при чему је $w_1, w_2, \dots w_T$ секвенца речи из речника V. При томе се сматра да за било који избор w_1^{t-1}, важи једнакост 3.4.3.2.

$$f(w_t, \dots, w_{t-n+1}) = P(w_t | w_1^{t-1}) \quad (3.4.3.1)$$

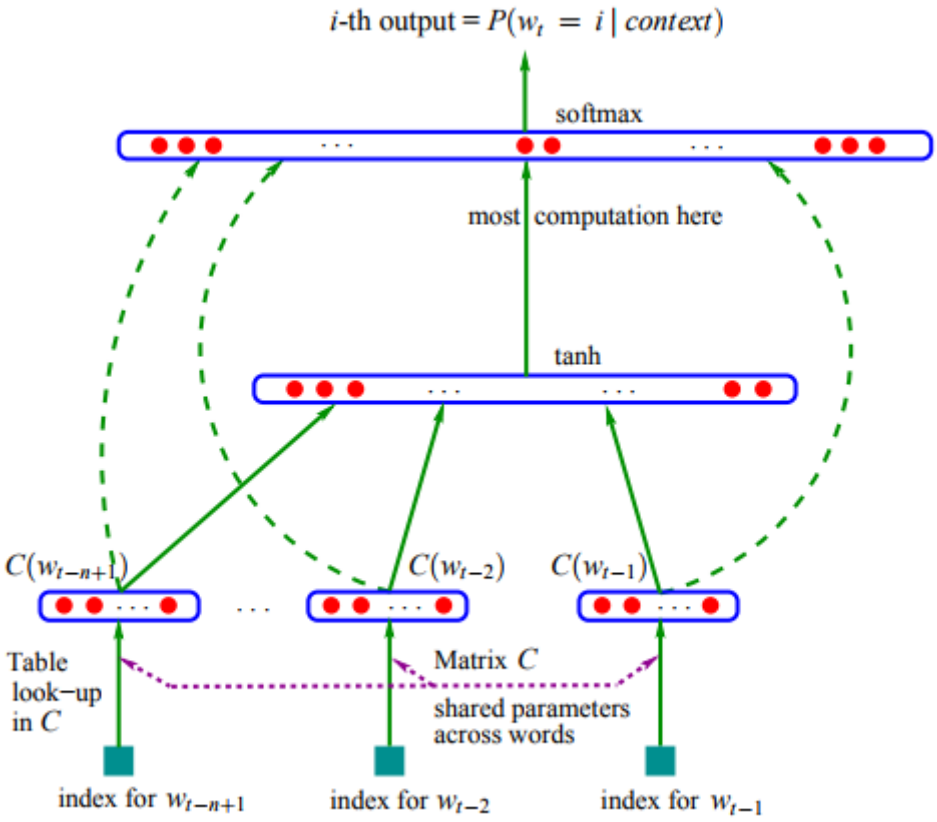
$$\sum_{i=1}^{|V|} f(i, w_{t-1}, \dots, w_{t-n+1}) = 1 \quad (3.4.3.2)$$

Функција 3.4.3.1 се може представити као композиција две функције:

1. Мапирања C речи i из речника V у реалне векторе $C(i) \in R^m$. Излаз овог мапирања је дистрибуирани вектор одлика које асоцирају на одговарајуће речи, те је његова коначна димензија $|V| \times m$.

2. Функције вероватноће (g) која мапира улазну секвенцу вектора одлика речи (једначина 3.4.3.3) у дистрибуцију условне вероватноће над речима из речника. Излаз овог мапирања је вектор чији i-ти елемент има вероватноћу $P(i|w_1{}^{t-1})$. Ова функција је моделована неуралном мрежом, а цела композиција је приказана на слици 3.4.3.1.

$$f(i, w_{t-1}, \dots, w_{t-n+1}) = g(i, C(w_{t-1}), \dots, C(w_{t-n+1})) \quad (3.4.3.3)$$



Слика 3.4.3.1. Мрежна архитектура која одговара композицији у формули 3.4.3.3. *g* је неуралне мрежа, а *C(i)* је *i*-ти вектор одлика речи. [56]

Функција g може се представити помоћу основне (*feed-forward*) или рекурентне неуралне мреже или помоћу неке друге параметризоване функције са параметрима ω. С тога се скуп параметара може дефинисати као $\theta = (C, \omega)$. Било који од ових модела треба истренирати тако да максимизују функцију приказану у 3.4.3.4, где је $R(\theta)$ регуларизација.

$$L = \frac{1}{T}\sum_t log f(w_t, w_{t-1}, \dots, w_{t-n+1};\ \theta) + R(\theta) \quad (3.4.3.4)$$

У моделу приказаном изнад број параметара се повећава линеарно са бројем речи у речнику и са фактором скалирања n. На слици 3.4.3.1. може се видети да дата неуралне мрежа има два скривена нивоа: 1) дељени ниво одлика C који је линеаран, тј. не додаје корисну функцију у модел, већ је ту само ради учења параметара и 2) други ниво који је представљен хиперболичким тангенсом. Излазни слој користи софтмакс функцију дату једначином 3.4.1.4 чиме се осигурава позитивна вредност излаза и то да је њихова сума једнака 1 (услов дат једначином 3.4.3.2). Условна вероватноћа речи која зависи од секвенце речи које јој претходе, може се дефинисати једначином 3.4.3.5.

$$P(w_t|w_{t-1}w_{t-2}\dots w_{t-n+1}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}} \quad (3.4.3.5)$$

Излази y_i су ненормализоване логаритамске вероватноће за сваки излаз речи *i*, које се добијају по формули 3.4.3.6, при чему су x = (C(w_{t-1}), C(w_{t-2}),…, C(w_{t-n+1})), *b* и *d* помераји, а *H*, *W* и *U* тежине грана између слојева. *W* је опциона тежина, тј. ова тежина се може дефинисати са 0, тј. тај линк не мора да постоји.

$$y = b + Wx + Utanh(d + Hx) \quad (3.4.3.6)$$

Број параметара које оваква мрежа има састоји се од |*V*| помераја за излаз, *h* помераја за скривени слој, |*V*| * *h* тежина *U* које повезују скривени и излазни слој, |*V*| * *(n-1)* * *m* тежина *W* које повезују одлике речи са излазом, *h* * *(n-1)* * *m* тежина *H* које повезују два скривена слоја и |*V*| * *m* тежина *C* које повезују речи са њиховим одликама, те се стога скуп параметара може дефинисати као у 3.4.3.7, а укупан број параметара је дат формулом 3.4.3.8.

$$\theta = (b, d, W, U, H, C) \quad (3.4.3.7)$$

$$O = |V| * (1 + nm + h) + h * (1 + (n - 1) * m) \quad (3.4.3.8)$$

Највећу тежину, тј. доминанти фактор у формули 3.4.3.8 има члан |*V*|*(*nm* + *h)* који представља тежине које повезују улазни и скривени слој с излазом. У раду [56] за учење неуронске мреже користи се стохастичко градијентно уздизање (stochastic gradient ascent) који итеративно ажурира тежине за речи користећи формулу 3.4.3.9, где θ представља одговарајући параметар, а α је коефицијент учења. Треба имати у виду да велика количина параметара не треба да се ажурира у сваком кораку, зато што не одговарају речима које су тренутно на улазу.

$$\theta = \theta + \alpha \frac{\partial log P(w_t|w_{t-1},\dots,w_{t-n+1})}{\partial \theta} \quad (3.4.3.9)$$

У раду [56] је показано да се резултати оваквог модела могу додатно побољшати комбинацијом вероватноћа предикције неуралних мрежа са моделом интерполираних триграма, при чему се за комбинацију може користити фиксна тежина 0,5, научене тежине или скуп тежина зависан од фреквенције контекста. Овакве методе су већ постојале, али су се бавиле комбиновањем униграма, биграма и триграма како би добиле бољу тачност.

Имајући у виду колико параметара треба изменити, долази се до схватања да неуралне мреже такође могу бити временски веома захтевне, чак и пуно захтевније од н-грама код којих је *n* прихватљиво мало (обично је *n* у опсегу један до шест). Међутим, постоје три начина за драстично смањење временске комплексности овог алгоритма:

1. У сваком кораку се мењају само тежине које морају, тј. само оне које су зависне од тренутног улаза.
2. У случају јако малих тежина неких грана, те гране се укидају.
3. Структуру неуралних мрежа је могуће дистрибуирати, тако да се израчунавања могу паралелизовати и самим тим се убрзава рад целокупног алгоритма. Данас постоје софтверски модули који су тако имплементирани на сваком више коришћеном језику, а постоје и хардвери који су креирани специфично за рад са оваквим израчунавањима.

У раду [56] су описани детаљи паралелизоване имплементације језичког модела помоћу неуралних мрежа, што излази из опсега овог рада. У истом раду је показано да су резултати добијени помоћу неуралних мрежа бољи од резултата добијених н-грамима.

Дакле, овај модел је по резултатима бољи од н-грама, али је генерално спорији. С друге стране, могуће је имплементирати га у паралелизованом систему. Такође, овај модел је у стању

да разазна сличности између речи или вишезначност код речи што н-грами не могу. На крају, он такође може да одреди следећу реч на основу много већег контекста него што то раде н-грами.

Сличним проблемом бави се и рад [57] који користи неуралне мреже у препознавању говора. Основна идеја овог рада је да се оцене вероватноће за језички модел у континуалном простору.

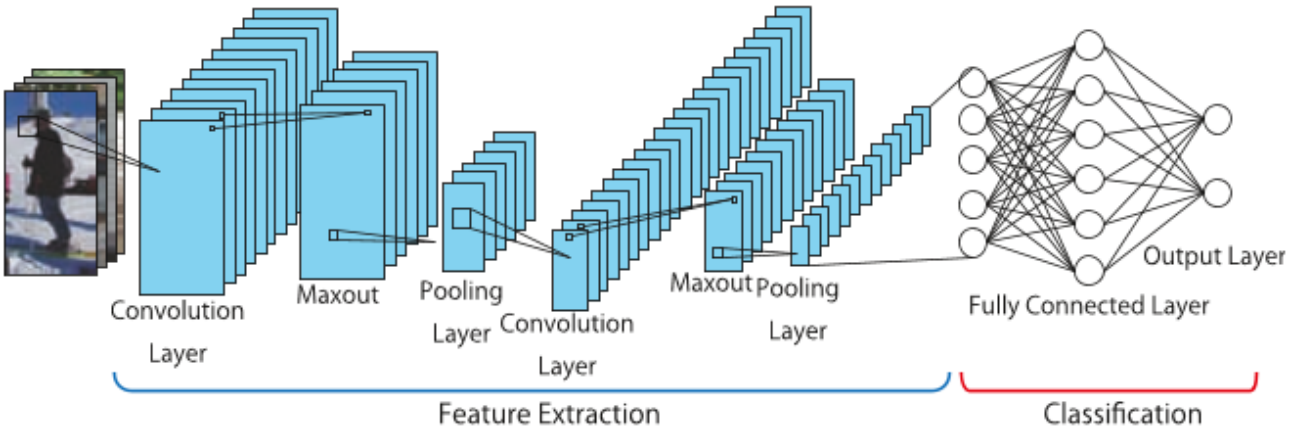
3.5. Креирање језичких модела помоћу конволуционих неуралних мрежа

Конволуционе неуралне мреже су настале као побољшање основних неуралних мрежа при обради разних врста сигнала и других временских улаза. Ове мреже су посебно популарне код обраде слика и видео материјала јер дају најбоље резултате у поређењу са осталим техникама. Међутим, оне се такође користе за обраду других врста сигнала, као што су ЕКГ, звучни сигнали, говор, текст и слично. Ове мреже су инспирисане функционисањем и везама неурона у људском мозгу.

3.5.1. Конволуционе неуралне мреже (CNN)

При процесуирању велике количине података као што је пример са сликама, тектом, говором, видео материјалима, потпуно повезане неуралне мреже су веома временски захтевне зато што се број тежина с којима треба радити значајно повећава с повећањем броја података или величине мреже, као што се може видети у одељку 3.4.3. Додатно, у случају великог броја неурона или слојева, може доћи до прекомерног уклапања (*overfitting*) уколико не постоји велика количина података у тренинг скупу. Конволуционе неуралне мреже су настале као покушај решавања тих проблема уз побољшање резултата, а идеја за модел је добијена кроз спознавање начина на који људски мозак функционише.

Неурони код конволуционих неуралних мрежа су организовани у тако да имају додатну димензију дубине унутар једног слоја. Такође, слој може имати ширину и дубину у случају да се ради обрада дводимензионих сигнала (као што су слике) или само ширину у случају обраде једнодимензионих сигнала (говор, текст, ЕКГ). Дубином се овде сматра чињеница да се унутар једног скривеног слоја налази више паралелних скривених слојева који ће бити детаљно описани касније. [58]



Слика 3.5.1.1. Приказ примера конволуционе мреже [59]

Ефикасност конволуционих неуралних мрежа увелико зависи од великог броја слојева које ова мрежа има као и од њихове разноврсности. Она може имати улазни, конволуциони,

ReLu, удружени (*pooling*), нормализациони, потпуно повезани и излазни слој. Најбитнији су конволуциони, удружени и потпуно повезани слој, међутим често се ови нивои користе више пута или се између њих додају неки други од горе наведених слојева. Због ове разноврсности, не постоји јединствена архитектура конволуционих мрежа, већ модел зависи од проблема, а најчешће се користи архитектура као на слици 3.5.1.1.

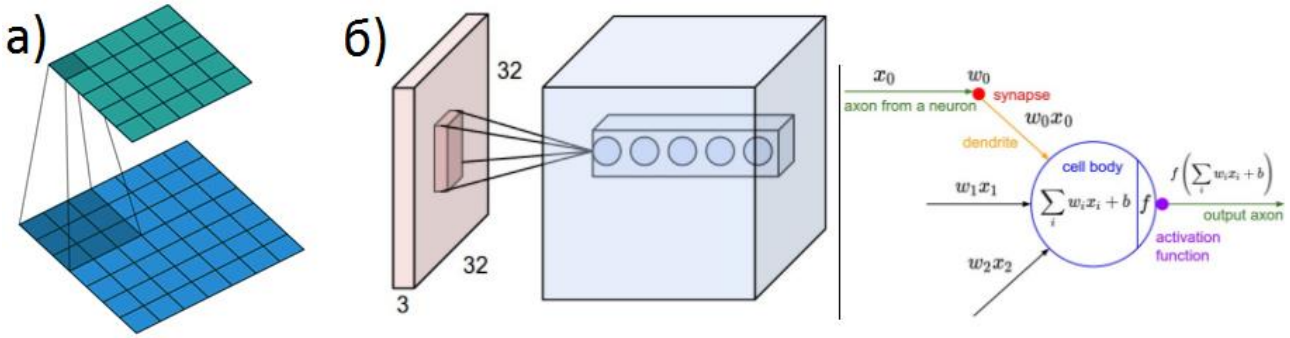
i) Конволуциони слој

Конволуциони слој је најбитнији део ових мрежа и он ради највећу количину рачунања. Овај слој се састоји из више паралелних слојева неуралних мрежа (дубина слоја) који ће се од сада звати активационе мапе. У случају обраде слика и других дводимензионих сигнала, свака активациона мапа ће имати дводимензиону структуру (висина и ширина). Пошто се овај рад бави обрадом текста, који је једнодимензиони сигнал, активациона мапа је исто једнодимензиона (ширина), па је слој скуп паралелних једнодимензионих активационих мапа тј. има дубину и ширину.

Сви неурони унутар једне активационе мапе имају исте тежине ка претходном (улазном) нивоу и скуп ових тежина се назива филтер, док су филтери различити за сваку активациону мапу. Чињеница да једна активациона мапа има један филтер за све локалне улазе значи да ће се из сваког дела улаза издвојити исте одлике на основу тог филтера, а уз то је број тежина које се користе много мањи него кад би те тежине биле различите. Дакле, филтер је функција која пресликава део по део улазног сигнала у неуроне активационе мапу. Основне карактерстике филтерске функције су:

1. пресликавањем се смањује ширина (и висина у случају слике)
2. број филтера у једном слоју (дубина) је већа од дубине претходног слоја
3. улаз у филтер се формира тако што се прозор ширине филтера помера кроз улаз за s позиција и тако се долази до новог дела улаза који се затим помоћу филтера слика у неурон активационе мапе (слика 3.5.1.2 а)).

Филтерском функцијом се само мали део улаза повезује са сваким неуроном активационе мапе за разлику од основних неуралних мрежа код којих су сви улази били повезани са сваким неуроном у срквеном слоју. Тиме се постиже смањење броја тежина и самим тим убрзавање рачунања. Вредност s представља померај (stride) филтера и његова вредност се рачуна по формули 3.5.1.1, где је w_0 ширина улаза, w_1 ширина активационе мапе, а w_f је ширина филтера. Еквивалентно за висину код дводимензионих улаза.



Слика 3.5.1.2. а) Пример коришћења филтера да би се од улаза (доњи слој) добила активациона мапа (горњи слој) мањих димензија. [60] б) Детаљи мапирања улаза у активациону мапу [58]

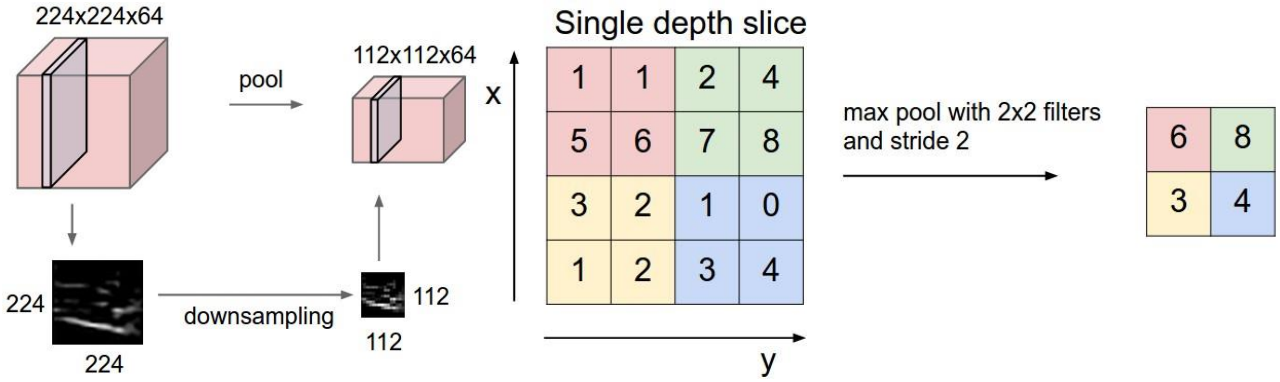
$$s = floor(\frac{w_0 - w_1}{w_f}) + 1 \qquad (3.5.1.1)$$

Конволуциони слој је добио то име зато што се извршава конволуција између локалног улаза и филтера како би се креирале активационе мапе. Због овог се филтер често назива и кернел. Свака ћелија у активационој мапи добија вредност рачунањем скаларног производа локалног улаза и одговарајућег филтера, а затим се примењује активациона функција као што је приказано на слици 3.5.1.2 б).

Понекад се улаз може проширити нулама (*zero-padding*) око ивица улазног сигнала. Ово се посебно користи за слике када се узимају у обзир и детаљи који се налазе на њеним крајевима. У том случају се бројиоцу у формули 3.5.1.1 додаје величина којом је улаз проширен.

Израчунавања у конволуционим слојевима се могу додатно убрзати уколико се прикажу као множење матрица улаза и тежина јер постоје специјализовани оптимизовани системи који матрице могу да множе веома брзо.

Алгоритам пропагације у назад за операцију конволуције је такође конволуција, али са просторно-обрнутим филтерима. [58]



Слика 3.5.1.3. Удруживање помоћу функције максимизације [58]

i) Удружени (pooling) слој

Овај слој је познат и под називом слој за узимање узорака (*downsampling layer*) зато што је његова основна улога да смањи величину активационих мапа, а самим тим и параметара конволуционе мреже узимањем узорака из постојећих мапа. Овим се такође контролише и прекомерно уклапање унутар мреже. Удруживање (тј. редимензионисање) се ради на свакој активационој мапи посебно као што је приказано на слици 3.5.1.3, а најчешће се користи функција максимизације. Активациона мапа се подели на делове и за сваки део се нађе максимална вредност која се пресликава у наредни слој.

Друга могућност је да се постави корак (*stride*) *s* и прозор ширине *f* такав да се нови слој креира применом функције на прозор ширине *f*, а затим се прозор помера за корак *s* и понавља се операција док се не обради цела активациона мапа. Овај поступак се ради за сваку активациону мапу, тако да је број паралелних активационих мапа пре и после удруживања исти, док је ширина новог слоја дата једначином 3.5.1.2, при чему су w_1 и w_2 ширине активационих мапа пре и после удруживања.

Поред функције максимизације, удруживање се може вршити и другим функцијама, од чега су најкоришћеније функција средње вредности и удруживање на основу L_2 норме (једначина 3.5.1.3).

$$w_2 = \frac{w_1 - f}{s} + 1 \quad (3.5.1.2)$$

$$f(\vec{x})\sqrt{\sum_{k=1}^{n} |x_k|^2} \quad (3.5.1.3)$$

Током алгоритма пропагације у назад, у првом кораку (пропагација у напред) чува се индекс неурона из претходног слоја који је имао највећу вредност, тако да се он може искористити у другом кораку (пропагација у назад). [58]

ii) Нормализациони слој

Различите нормализационе шеме могу се наћи у овом кораку. Постоје и покушаји имплементације инхибиционих шема које постоје у мозгу. Међутим, експерименти су показали да овај слој не доприноси много коначном резултату, тако да се он често ни не користи у данашњим конволуционим неуралним мрежама. [58]

iii) Потпуно повезан слој

Ово је крајњи слој конволуционих неуралних мрежа. Неурони из овог слоја су потпуно повезани са неуроним из претходног слоја као код обичних неуралних мрежа. Пре овог корака се очекује да су се одлике већ издвојиле у претходним корацима и да се помоћу овог слоја ради коначна класификација. Такође, очекује се да су претходни слојеви смањили димензионост у односу на улаз.

3.5.2. Магистралне неуралне мреже (Highway Networks)

С појавом дубоког учења истраживањима у тој области показано је да дубље мреже (са више нивоа) дају боље резултате. Међутим, што је неуралне мрежа дубља то је теже истренирати је због великог број тежина које треба одредити и због проблема са ишчезавајућим и великим градинјентима. Због тога се у [61] описује нова архитектура, магистралне неуралне мреже која је дизајнирана да поједностави тренирање дубоких неуралних мрежа. Магистралне неуралне мреже омогућавају неометан пролаз информација кроз неколико слојева користећи магистрални слој. Ова мрежа омогућава оптимизацију мреже без обзира на њену дубину користећи механизам капија по угледу на дуге краткотрајне мреже (о којима ће бити речи у једном од наредних потпоглавља). Магистрални слој омогућава неуралној мрежи путању кроз коју информација тече кроз неколико слојева без слабљења.

Код основних неуралних мрежа вредности излаза из неурона рачунају се по формулама 3.4.2.1. и 3.4.2.2. у зависности од слоја. Магистрални слој представља нелинеарну трансформацију основне идеје укључивањем трансформационе капије *T* (описана једначином 3.5.2.1) и капије домашаја *C* (описана једначином 3.5.2.2), тако да се она рачуна по једначини 3.5.2.3.

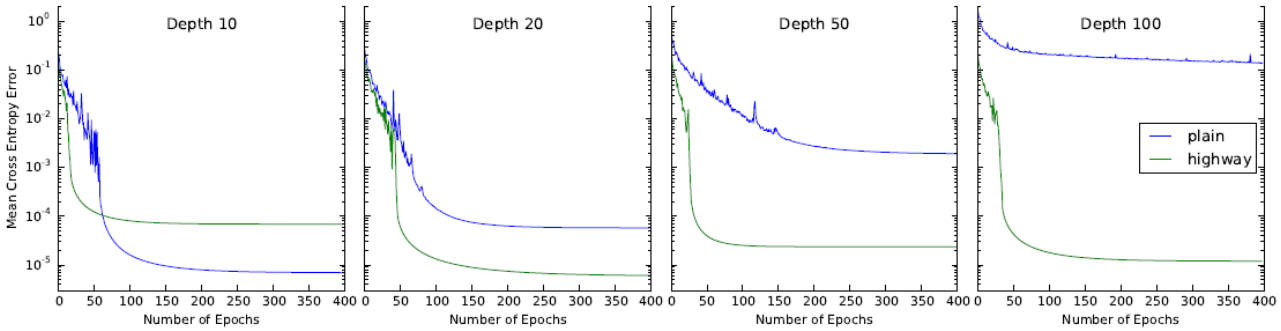
$$T(x) = \sigma(W_T x + b_T) \quad (3.5.2.1)$$

$$C(x) = 1 - \mathrm{T}(\mathrm{x}) \quad (3.5.2.2)$$

$$hn(x) = h(x) * \mathrm{T}(\mathrm{x}) + x * \mathrm{C}(\mathrm{x}) = \sigma_h(W_{hx} x + b_h) * \sigma(W_T x + b_T) + (1 - \sigma(W_T x + b_T) \quad (3.5.2.3)$$

У овим једначинама *h(x)* је формула излаза из неурона код основних неуралних мрежа, W_T и b_T су тежина и померај трансформационе капије, а h_n представља излаз из магистралног слоја. У [61] капија домашаја је узета као разлика јединице и трансформационе капије ради једноставности. Последица таквог избора је да у случају *T(x) = 1* излаз је еквивалентан излазу из неурона код основних неуралних мрежа, а у случају *T(x) = 0* излаз је једнак улазу у тај неурон, тј. сигнал се само пропушта даље.

У [61] су трениране мреже дубине до 100 слојева и упоређивани су резултати за основне и магистралне мреже. Показано је да је тренирање магистралне архитектуре независно од броја слојева, док основне мреже имају велике проблеме како се број слојева повећава (слика 3.5.2.1). У раду се такође напомињу прелиминеарни резулатати над 900 слојева који се могу тренирати стохастичким градијентим спустом са моментумом захваљујући овој архитектури. Овакви резултати су разлог зашто се магистрални слој комбинује са другим слојевим код дубоког учења. Он се такође користи при конструисању језичког модела помоћу конволуционих неуралних мрежа о чему ће бити речи у следећем одељку.



Слика 3.5.2.1. Средња грешка за основне и магистралне мреже при дубини 10, 20, 50 и 100 слојева користећи стохастички градијентни спуст за тренирање

3.5.3. Језички модел конволуционих неуралних мрежа

Као што је већ поменуто у одељку о неуралним мрежама ови модели се могу користи за моделирање реченица и семантичке улоге речи у реченици. Више детаља о тој примени може се наћи у раду [62] који описује динамичку конволуциону неуралну мрежу и њену примену на семантичко моделирање реченица. У њиховој мрежи се користи динамичко удруживање са функцијом која изабира *k* максимума. Улаз је линеаран у форми реченица које могу бити различитих дужина, а мрежа учи граф одлика над реченицама. Овакав модел је у стању да схвати кратке и дуге фразе и релације између речи. Добра страна овог модела је да не зависи од парсирања текста и због тога се може користити лако код сваког језика. У раду се модел тестира над различитим скуповима података укључујући ту и кратке текстове. У поређењу са другим техникама које су базиране на н-грамима, парсерима, врећама речи имплементираним помоћу неуралних мрежа, технике базиране на конволуционим неуралним мрежама дају најбоље резултате на основу [62].

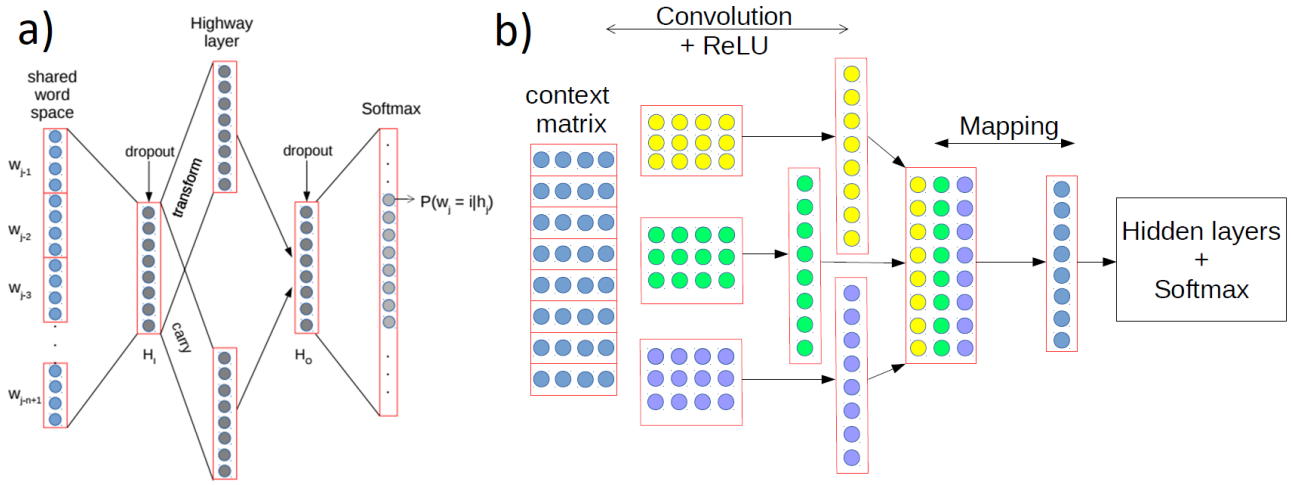
Разумевање и кластеризовање кратких текстова, као што су поставке на друштвеним мрежама (*twitter, facebook, reddit* и другим) постали су веома битни задаци у последњих пар година. Због великог броја чланова ових мрежа, поставке објављене на њима имају великог утицаја на формирање мишљења о важним темама, на успех производа и услуга, па чак и на изборне процесе. Због тога постоје модели који раде само са кратким текстовима и покушавају да направе најбољи језички модел за њих. Архитектура базирана на конволуционим неуралним мрежама који кластеризује кратке текстове описана је у [63].

Проблем код кратких текстова је велика расејаност због мале дужине и огромног броја поставки, тј. докумената. У раду [63] се одлике речи претварају у бинарне кодове који омогућавају да се очувају ограничења, а затим се речи постављају као улази у конволуциону неуралну мрежу која учи одлике тако да се излазни неурони тренирају са бинарним кодовима. Када се на тај начин добију репрезентације речи, користи се алгоритам *k*-средњих вредности

за кластеризацију (технике за кластеризацију биће описане у поглављу четири). У раду [63] се показује да овакав систем даје боље резултате него технике засноване на различитим кластеризацијама излаза из *tf-idf*. Данас постоје боље технике за рад са кратким текстовима и многе од њих су настале у исто време кад и овај рад, а неке од њих биће поменуте у наставку текста.

Претходне примене конволуционих мрежа у обради текста су само кратко описане јер како је већ објашњено у овом мастер раду фокус је на коришћењу методологија за креирање језичких модела. У [64] користе се конволуционе неуралне мреже за проналажење одлика текстова и креирање језичких модела на основу тога. Овај алгоритам је бољи од раније представљених, али је такође компетитиван са моделима базираним на рекурентним неуралним мрежама који ће бити представљени у наредном потпоглављу.

У раду [64] модел базиран на неуралним мрежама (приказан у одељку 3.4.3) се унапређује додавањем магистралних слојева и регуларизацијом имплементираном испадањем података (*dropout*) чиме се побољшавају резултати. Магистралним слојевима се омогућава да чак и градијенти којима је активација 0 могу да имају пропагацију у назад користећи капију домашаја. Након свих измена основног модела, а пре примене конволуционих слојева, мрежа изгледа као на слици 3.5.3.1а).



Слика 3.5.3.1. а) Приказ архитектуре унапређене основне неуралне мреже; б) Додавање конволуционог слоја у архитектуру унапређене основне неуралне мреже [64]

Конволуциона мрежа се креира додавањем конволуционог слоја одмах после улазног слоја, при чему се у улазном слоју речи пројектују у одговарајуће векторске комбинације. Уместо да се улазни вектори надовежу у један дугачки вектор, они се надовезују трансверзално правећи матрицу димензија *n x m*, при чему је *n* величина улаза (број речи), а *m* величина вектора којим је реч репрезентована. Овако надовезани улази се убацују у конволуциони слој у ком се тежине филтера скаларно множе са делом улаза, затим се прозор помери за корак *s* и поступак се понавља за наредни део улаза.

На овако израчунате вредности, примењује се исправљена линеарна функција којом се одбацују негативне вредности. Овај поступак се понавља *m* пута пута користећи другачије филтере у том конволуционом слоју (тј. учећи различите одлике), тако да је дубина овог слоја *m*. У раду [64] излаз из овог слоја је исто *n x m*. Над излазима овог слоја врши се нормализација којом се врши и регуларизација научених репрезентација речи. На крају се добијена дводимензиона структура неурона слика у потпуно повезан једнодимензиони слој како би се извршили даљи кораци унапређене основне неуралне мреже. Структура основне мреже којој

се додаје конволуциони слој као што је описано у овом параграфу приказана је на слици 3.5.3.1 б).

У раду [64] такође се креира структура за моделирање језика помоћу конволуционог слоја који садржи различите филтере, а дају се и друга унапређења и варијанте описаног алгоритма, међутим то превазилази оквире овог рада. Овај алгоритам даје боље резултате од других метода за моделирање језика (укључујући и рекурентне неуралне мреже), осим дугих краткотрајних мрежа које дају бољи резултат. Зато се у наредном поглављу описују рекурентне мреже и дуге краткотрајне меморије као њихова подврста.

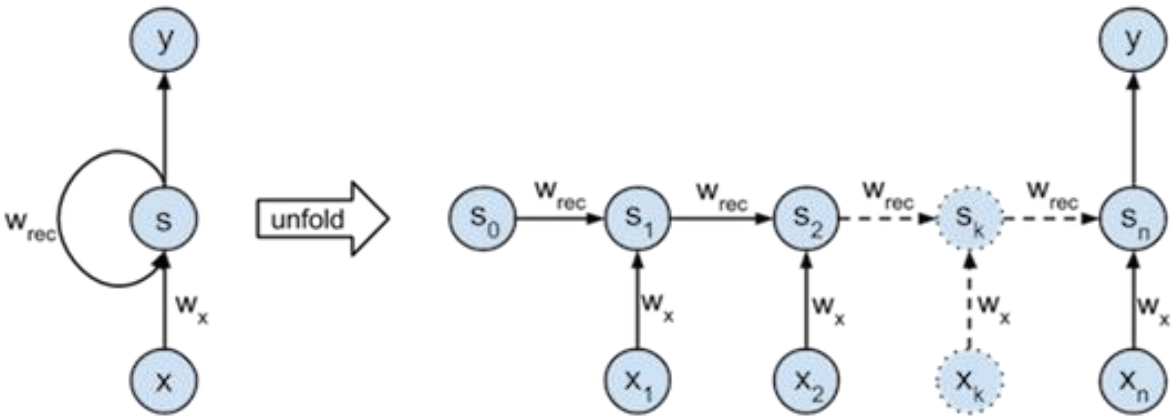
3.6. Креирање језичких модела помоћу рекурентних неуралних мрежа

3.6.1. Рекурентне неуралне мреже (RNN)

Основне неуралне мреже (*feed-forward neural networks*) су добар предикциони модел када је у питању улаз у ком су примери међусобно независни. Међутим, многи проблеми се могу решити боље уколико се укључи контекст у ком је одређен пример настао или ако се улазни примери посматрају зависно један од другог. Ово је случај и код обраде текстова код којих је битно које речи су пре или после тренутног улаза. На пример, за разумевање значења неке речи често је потребно знати које речи се налазе око ње или како гласи реченица у којој се она налази.

Рекурентне неуралне мреже су дизајниране да раде са временски зависним улазима и да предикцију врше не само на основу тренутног улаза, већ и на основу целокупног контекста. Основна разлика у односу на обичне неуралне мреже јесте да су неурони унутар истог слоја повезани међусобом тако да кроз њих информација тече с лева на десно, чиме се узима у обзир редослед улаза.

По дефиницији рекурентне неуралне мреже су вештачке неуралне мреже код којих везе између неурона формирају усмерено затворено коло због чега ове мреже разумеју контекст процесуирне произвољне улазне секвенце (слика 3.6.1.1). Постоје разне архитектуре и унапређења основног модела. Поред основних рекурентних мрежа, најпознатије су дуге краткотрајне меморије.



Слика 3.6.1.1. Пример рекурентне мреже. Размотан слој рекурзивне неуралне мреже (x_t је улаз, s_t је скривено стање, y је излаз у тренутку t) [64]

Док су основне неуралне мреже уређене у слојевима и информација путује само у једном смеру, од улазног ка излазном слоју, рекурентне неуралне мреже имају за циљ да што прецизније опонашају мозак. Због тога су првобитна ограничења укинута, те код рекурентних неуралних мрежа у ширем смислу, неурони могу бити повезани на различите начине, чак и са самима собом.

Како би објашњења била јаснија, у наставку ће бити речи о рекурентним неуралним мрежама са једним скривеним слојем, иако у пракси може бити неограничено много слојева. Рекурентне мреже уче из секвенци, где се секвенца дефинише као листа парова вектора $(\overrightarrow{x_t}, \overrightarrow{y_t})$, где је $\overrightarrow{x_t}$ улаз у мрежу у тренутку *i*, а $\overrightarrow{y_t}$ је одговарајући излаз. Скупови података се састоје из великог броја секвенци или из једне веома дуге секвенце.

Скривени слој се обележава са $\overrightarrow{h_t}$ и његове вредности зависе од улаза и од информација у скривеном слоју у претходном тренутку $\overrightarrow{h_{t-1}}$. Управо ова зависност је разлог што рекурентне неуралне мреже могу да разумеју контекст и да обрађују секвенце. Поред тога, у тренутку *i*, рекурентна мрежа има вектор тежина који повезује улазни и скривени слој W_{hx}, вектор тежина који повезује неуроне у скривеном слоју W_{hh}, вектор тежина који повезује скривени и излазни слој W_{yh} и предвиђени излаз o_i.

Скривено стање у тренутку i се формира по једначини 3.6.1.1, где је σ_h активациона функција за скривени слој, а b_h је померај улазног слоја. Излазни вектор се онда на основу тога рачуна помоћу формуле 3.6.1.2, где је σ_y активациона функција за излазни слој, а b_y је померај скривеног слоја. У неким варијантама рекурентне неуралне мреже излазни слој не садржи активациону функцију.

$$h_i = \sigma_h(W_{hh}h_{i-1} + W_{hx}x_i + b_h) \quad (3.6.1.1)$$

$$\mathrm{o}_i = \sigma_y(W_{yh}h_i + b_y) \quad (3.6.1.2)$$

Избор архитектуре рекурентне неуралне мреже зависи од проблема који се решава. Ове мреже уче помоћу алгоритма пропагације у назад кроз време (*backpropagation through time*) који ће бити описан у наредном одељку.

При имплементацији и коришћењу рекурентних неуралних мрежа треба имати у виду проблеме са исчезавајућим и превеликим градијентима који се могу појавити као последица уланчавања извода. У случају веома малих градијената, рекурентне мреже веома тешко уче, док веома велики градијенти производе нестабилност у систему и доводе до нумеричких грешки. Веома велики градијенти се решавају ограничавањем величине градијента, при чему се вредност градијента смањује уколико је већа од ограничења. Исчезавајући градијенти се решавају избацивањем конекција на којима се они налазе, чиме се убрзава процес. Нажалост ово решење није скалабилно.

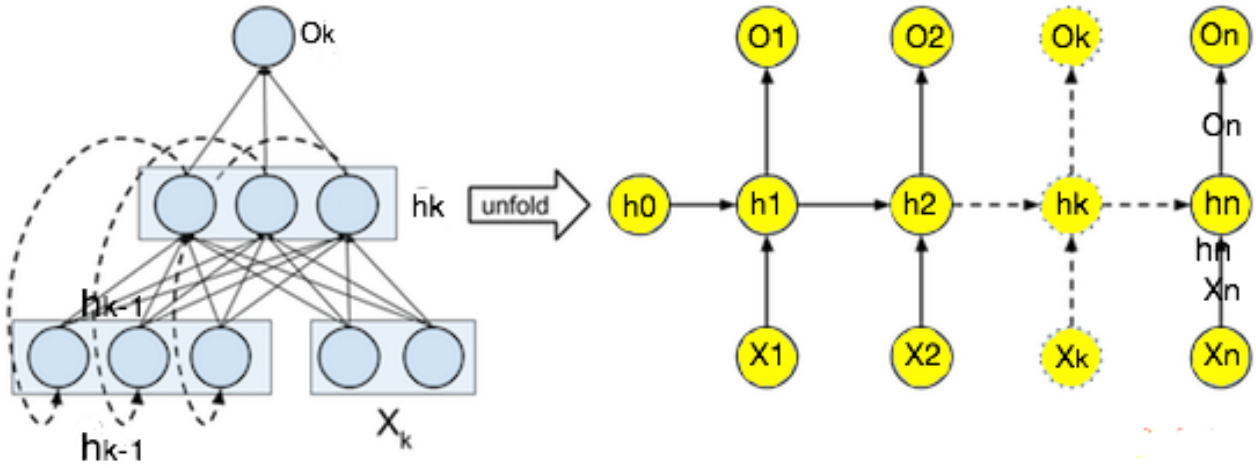
3.6.2. Алгоритам пропагације у назад кроз време

Како би се истренирала рекурентна неуралн мрежа, потребно је генерализовати алгоритам пропагације у назад. Постоје два начина да се изведе алгоритам пропагације у назад кроз време:

1. Дефинисањем метрике за грешку и парцијалном деривацијом грешке у односу на тежине или
2. Базирањем на алгоритам пропагације у назад са експанзијом неуралне мреже кроз време

У даљем тексту ће се користити други од горе поменутих алгоритама. Овај алгоритам се увелико ослаја на алгоритам пропагације у назад. Почетне тежине изабирају се као случајне вредности између 0 и 1. Ако је *t* дужина секвенце, онда треба расклопити рекурентну неуралну мрежу кроз време *t* пута тако да се креира стандардна неуронска мрежа и онда се над таквом мрежом ради пропагација у назад (слика 3.6.2.1).

Улазна секвенца на почетку има дужину један, затим 2, итд. све до тренутка *t* у ком има дужину *t*. Међутим, како би се опонашала основна неуронска мрежа, потребно је да дужина секвенце увек буде иста. Због тога се улаз допуњава нулама тамо где је дужина секвенца мања од *t*. Оваквим размотавањем настаје основна неуронска мрежа са улазним, излазним и *t-1* скривених слојева. Код ње се сада могу извршити кораци: пропагација у напред и пропагације у назад, али при том не треба заборавити да се улаз у скривени слој састоји од информације из улазног слоја и информације из скривеног слоја у претходном тренутку.



Слика 3.6.2.1. Расклапање рекурентне неуралне мреже кроз време

3.6.3. Креирање језичког модела помоћу рекурентних неуралних мрежа

Као што је већ описано циљ статистичког моделирања језика је да предвиди следећу реч на основу контекста, што је проблем предикције са секвенцијалним улазом. Архитектура рекурентних неуралних мрежа је створена да решава управо овакве проблеме. Као и код моделирања језика конволуционим мрежама, моделирање језика помоћу рекурентних неуралних мрежа се ослања на моделирање језика основним неуралним мрежама. Рекурентне неуралне мреже поред бољих резултата доносе и могућност да се сагледа цела секвенца текста, што је код основних неуралних мрежа ограничено на одређену унапред дефинисану дужину.

У раду [66] описује се једноставна рекурентна неуронска мрежа за моделирање језика која се састоји из улазног, скривеног и излазног нивоа. Скривени слој користи логистичку активациону функцију (једначина 3.4.1.1), а излазни слој користи софтмакс активациону функцију (једначина 3.4.1.4). Пропагацијом уназад кроз време тренира се мрежа на чији се улаз доводи текстуална секвенца.

Да би се побољшале перформансе, све речи чија је вероватноћа испод неке граничне вредности се представљају специјалним ретким токеном (једначина 3.6.3.1), при чему је C_{retko} број речи у речнику које се појављују ређе од неке граничне вредности.

$$P\big(x_i(t+1)\big|x(t),h(t-1)\big)=\begin{cases}\frac{y_{retko}(t)}{C_{retko}}, \text{ако је } x_i(t+1) \text{ ретко}\\ y_i(t), \text{иначе}\end{cases} \quad (3.6.3.1)$$

У [66] је показано да оваква архитектура даје боље резултате од обичних неуралних мрежа. Аутори су имплементирали статичку и динамичку верзију (код које се модел ажурира сваки пут кад се нови пример тестира над моделом) и показано је да динамичка верзија даје боље резултате.

Међутим, проблем брзине језичких модела имплементираних неуралним мрежама остаје. Број параметара с којима треба радити у описаној рекурентној мрежи дат је једначином 3.6.3.2 у којој се примећује да он расте линеарно са бројем речи у речнику, са бројем неурона у скривеном слоју и са бројем корака коришћених за пропагацију у назад кроз време (τ). Због тога су у [67] предложене измене језичког модела базираног на рекурентним неуралним мрежама како смањио број параметара.

$$O=(1+H)*H*\ \tau+H*V \quad (3.6.3.2)$$

То се може постићи ако се претпостави да свака реч припада једној од *C* класа, па се самим тим вероватноћа појаве речи у зависности од контекста може разложити као у једначини 3.6.3.3. Оваквим разлагањем се смањује сложеност и нова сложеност се може приказати једначином 3.6.3.4. Ово је значајно унапређење јер је број речи у речнику највећа димензија у једначини 3.6.3.2, а она је сада замењена много мањим бројем класа *C*.

$$P(x_i|kontekst)=P(x_i|c_i)*P(c_i|kontekst) \quad (3.6.3.3)$$

$$O=(1+H)*H*\ \tau+H*C \quad (3.6.3.4)$$

У овом случају реални контекст се замењује контекстом сачуваним у скривеном слоју (једначина 3.6.3.5). Уместо рачунања вероватноће дистрибуције над свим речима довољно је оценити вероватноћу дистрибуције над класама и онда дистрибуције речи унутар сваке класе.

$$P(x_i|kontekst)=P(x_i|c_i,h(t))*P(c_i|h(t)) \quad (3.6.3.5)$$

Резултати у [67] показују да се увођењем класа смањује тачност резултата за максимално 5%, али се алгоритам убрзава и до 15 пута.

Овај [68] модел се унапређује тако што се вектори који представљају речи дефинишу тако да зависе од контекста, тј. од реченице у којој се реч налази. Овакав улаз се формира коришћењем латентног Дирихлеовог додељивања (*LDA – Latent Dirichlet Allocation*). Латентно Дирихлеово додељивање је још један начин за моделирање језика са великом комплексношћу који није приказан у овом раду због тога што се данас углавном уместо њега користе модели неуралних мрежа који се могу дистрибуирати.

Да би се добио овај модел, у претходно дефинисану архитектуру рекурентне неуралне мреже, додаје се слој одлика *f(t)* тако да се он повезује као улаз у скривени и излазни слој. Слој одлика представља додатни улазни вектор који садржи информације комплементарне информацијама улазног вектора *x(t)*. У [68] слој одлика је заправо информација о теми којом се реченица у којој се налази та реч бави. На овај начин се исто решава проблем исчезавајућег градијента. Вредности у скривеном и излазном слоју се рачунају формулама 3.6.3.6 и 3.6.3.7, при чему су *f* и *g* логистичка и софтмакс функција, респективно, а са *W* су обележене тежине веза између слојева.

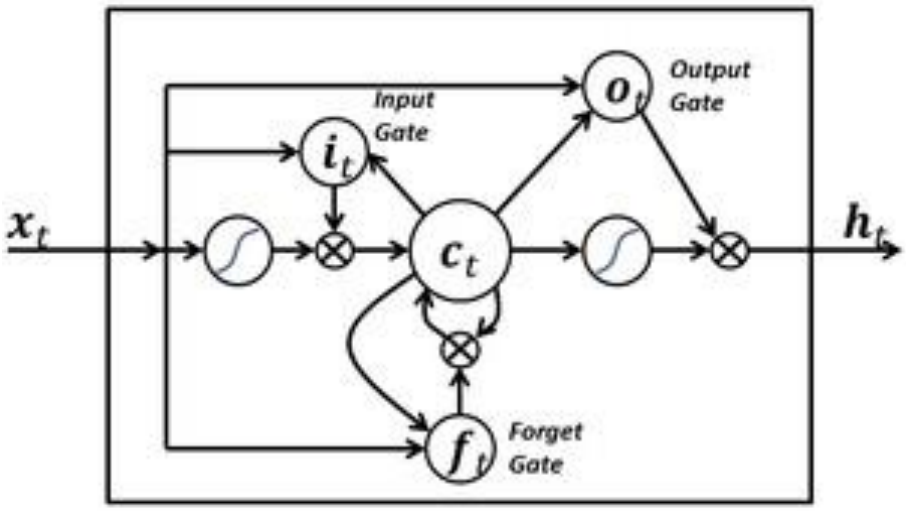
$$h(t)=f(W_{hi}x(t)+W_{hh}h(t-1)+W_{hf}f(t)) \quad (3.6.3.6)$$

$$o(t)=g(W_{yh}x(t)+W_{yf}f(t)) \quad (3.6.3.6)$$

Због бројних предности и мана имплементације језичких модела рекурентним неуралним мрежама постоје разне друге екстензије овог модела које решавају неке од проблема док занемарују друге. Највећи изазови су смањење временске комплексности и броја параметара, велики и исчезавајући градијенти и ограничења у стварној величини речника који може да се користи. У [69] дат је преглед и поређење ових унапређења који су ван опсега овог рада.

3.6.4. Дуге краткотрајне меморије (LSTM – long short-term memories)

Дуге краткотрајне меморије су популаран избор за решавање секвенцијалних проблема неуралним мрежама, а често дају најбоље резултате. Главна разлика између рекурентних неуралних мрежа и дугих краткотрајних меморија јесте у изгледу неурона у скривеном слоју који омогућава краткотрајно памћење унутар дугих краткотрајних меморија.

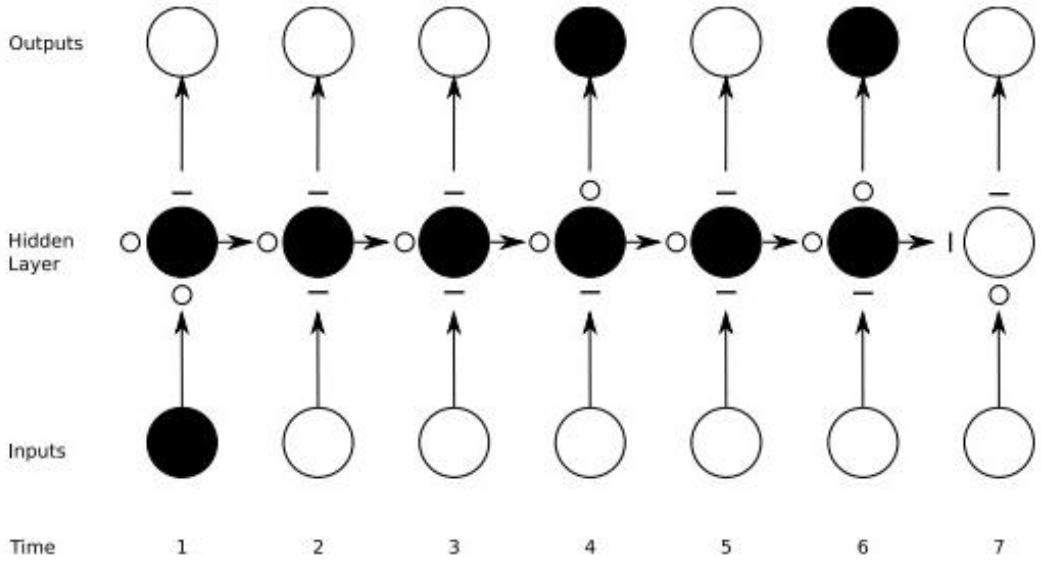


Слика 3.6.4.1. Архитектура ћелије дуге краткотрајне меморије

Код дугих краткотрајних меморија неурони у скривеном слоју су замењени ћелијама са архитектуром која може да памти (слика 3.6.4.1), а која се састоји из четири дела:

- Улазна капија (i_t) одлучује да ли ће се тренутни улаз додати у ћелију;
- Капија заборава (f_t) одлучује да ли ће се неко знање из ћелије заборавити;
- Излазна капија (o_t) одлучује да ли треба ишчитати (даље проследити) тренутну вредност ћелије;
- Меморијска ћелија (c_t) складишти и памти информацију.

За одлучивање се користи нека од активационих функција приказана у једначинама 3.4.1.1 до 3.4.1.4, али се најчешће користи логистичка функција (3.4.1.1) и тангенс хиперболички (3.4.1.2). Дуга краткотрајна меморија може да сачува грешку која се пропагира кроз време и слојеве. На овај начин, омогућава се учење током много већег броја корака него што је то било код основних рекурентних неуралних мрежа. Такође, постојање меморијске ћелије омогућава складиштење информације колико год времена је потребно да би се одредила што боља веза између улаза и излаза.



Слика 3.6.4.2. Пропагација сигнала кроз меморијске ћелије дуге краткотрајне мреже [70]

Горе поменуте капије раде на основу сигнала који примају и тежина које су им додељене у том тренутку и на основу ових информација блокирају или пропуштају информацију (слика 3.6.4.2). Тежине на капијама се мењају током учења на сличан начин као код рекурентних неуралних мрежа. Дакле, ћелије уче када треба да прихвате податак, када да га избришу из меморије, а када да га пропусте следећем слоју кроз итеративни процес, користећи алгоритам пропагације у назад кроз време и мењајући своје тежине на основу градијената. На слици 3.6.4.2. се може видети како се ћелије додају у скривени слој дугих краткотрајних меморија које су развијене кроз време. Цртице означавају затворене капије, а мали кружићи означавају отворене капије. Велики црни кругови су они кроз које информација тече од улаза ка излазу.

Као што се може видети ова архитектура изгледа врло слично као развијена рекурентна мрежа, те се такође може користити алгоритам пропагације у назад кроз време. Испод су описани кораци пропагације у назад и затим пропагације у напред за тренутак *t*. Треба имати у виду да ћелије у том тренутку имају садржај из претходне итерације, тј. из тренутка *t-1*.

У наредним једначинама и сликама биће представљен алгоритам пропагације у назад за дуге краткотрајне меморије. Индекси у једначинама остаће потписани ради конзистентности са претходним потпоглављима, а исти ти индекси на сликама су натписани. У тренутку *t* ћелија добија вредности улаза x_t и претходног скривеног слоја h_{t-1}. За ово стање улаз и капије се могу описати једначинама 3.6.4.1 до 3.6.4.4.

$$a_t = \tanh(W_c x_t + U_c h_{t-1}) = \tanh(\widehat{a_t}) \quad (3.6.4.1)$$

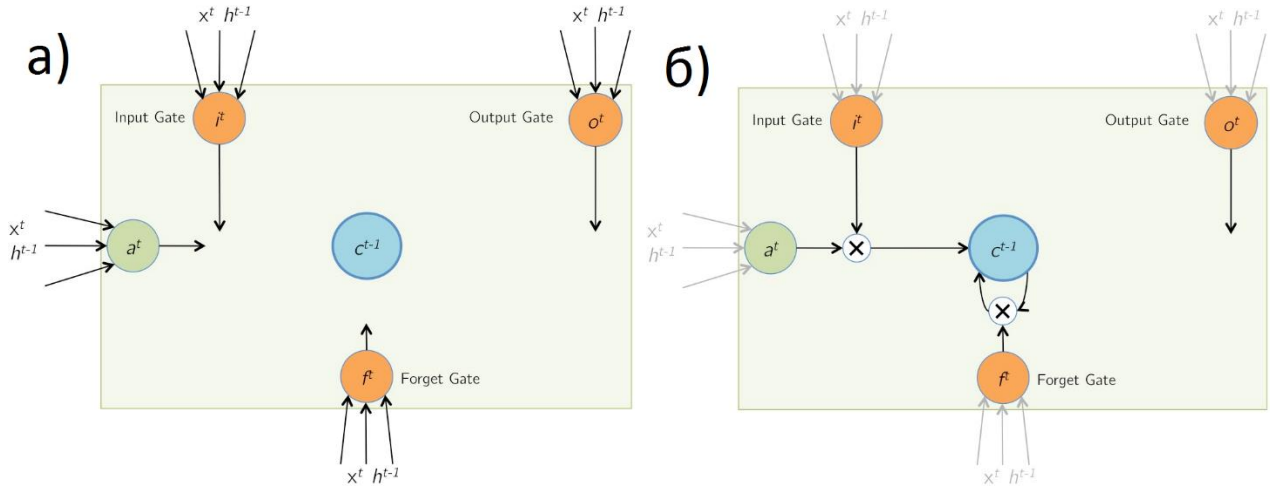
$$i_t = \sigma(W_i x_t + U_i h_{t-1}) = \sigma(\widehat{i_t}) \quad (3.6.4.2)$$

$$f_t = \sigma(W_f x_t + U_f h_{t-1}) = \sigma(\widehat{f_t}) \quad (3.6.4.3)$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1}) = \sigma(\widehat{o_t}) \quad (3.6.4.4)$$

Ово стање се може приказати и векторски као у једначини 3.6.4.5, а систем се може видети на слици 3.6.4.3а).

$$z_t = \begin{bmatrix} \widehat{a}_t \\ \widehat{\imath}_t \\ \widehat{f}_t \\ \widehat{o}_t \end{bmatrix} = \begin{bmatrix} W_c & U_c \\ W_i & U_i \\ W_f & U_f \\ W_o & U_o \end{bmatrix} [x_t \quad h_{t-1}] = WI_t \tag{3.6.4.5}$$



Slika 3.6.4.3. Пропагација у напред: а) корак 1: улаз и рачунање вредности капија; б) корак 2: ажурирање меморијске ћелије [71]

Вредности меморијских ћелија се ажурирају комбинацијом a_t и претходног садржаја ћелије c_{t-1} (слика 3.6.4.3 б)). Комбинација је базирана на тежинама улазне капије и капије заборава (једначина 3.6.4.6), при чему ° означава Хадамардов производ елемената матрице $(A°B)_{i,j} = (A)_{i,j}(B)_{i,j}$.

$$c_t = i_t°a_t + f_t°c_{t-1} \tag{3.6.4.6}$$

Коначно, рачуна се излаз коришћењем тежине излазне капије и прослеђивањем нове вредности ћелије (једначина 3.6.4.7) као што је приказано на слици 3.6.4.4 а).

$$h_i = o_t°tanh(c_t) \tag{3.6.4.7}$$

Креирањем излаза завршава се пропагација у напред. Након пропагације у напред, као и код претходно описаних мрежа, ради се пропагација у назад која ће такође бити описана за тренутак t. То значи да се кораци понављају у обрнутом редоследу, само што се сада прослеђују одговарајући градијенти. Градијенти за излаз и излазну капију рачунају се као што је приказано у једначинама 3.6.4.8 до 3.6.4.12, а стање система у том тренутку приказано је на слици 3.6.4.4 б). Једначине 3.6.4.9. и 3.6.4.11. су приказане за k-ти неурон у скривеном слој.

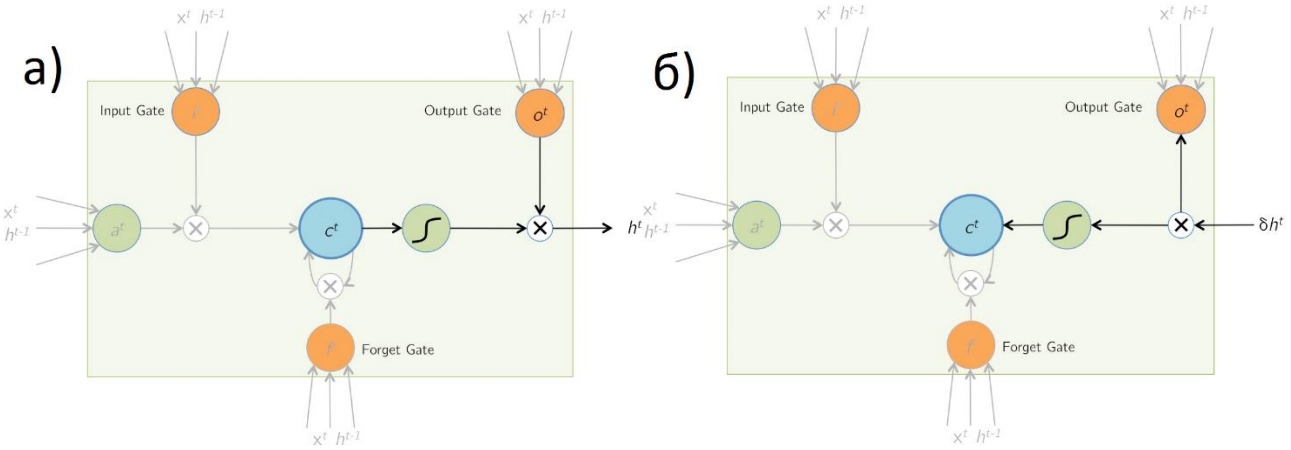
$$\partial h_t = \frac{\partial E}{\partial h_t} \tag{3.6.4.8}$$

$$\frac{\partial E}{\partial o_{tk}} = \frac{\partial E}{\partial h_{tk}} * \frac{\partial h_{tk}}{\partial o_{tk}} = \partial h_{tk} * \tanh(c_{tk}) \tag{3.6.4.9}$$

$$\partial o_t = \partial h_t \circ \tanh(c_t) \tag{3.6.4.10}$$

$$\frac{\partial E}{\partial c_{tk}} = \frac{\partial E}{\partial h_{tk}} * \frac{\partial h_{tk}}{\partial c_{tk}} = \partial h_{tk} * o_{tk} * (1 - tanh^2(c_t) \tag{3.6.4.11}$$

$$\partial c_t = \partial h_t°o_t°(1 - tanh^2(c_t)) \tag{3.6.4.12}$$



Слика 3.6.4.4. а) Пропагација у напред, корак 3: креирање излаза: б) Пропагација у назад, корак 1: рачунање градијента за излаз [71]

Градијент израчунат у последњој формули додаје се на градијент из времена t-1 који се рачуна у једначинама 3.6.4.13 до 3.6.4.20, при чему су једначине 3.6.4.13, 3.6.4.15, 3.6.4.17 и 3.6.4.19 израчунате за *k*-ти неурон у скривеном слоју. Овај корак приказан је на слици 3.6.4.5. а).

$$\frac{\partial E}{\partial i_{tk}} = \frac{\partial E}{\partial c_{tk}} * \frac{\partial c_{tk}}{\partial i_{tk}} = \partial c_{tk} * a_{tk} \quad (3.6.4.13)$$

$$\partial i_t = \partial c_t \circ a_t \quad (3.6.4.14)$$

$$\frac{\partial E}{\partial f_{tk}} = \frac{\partial E}{\partial c_{tk}} * \frac{\partial c_{tk}}{\partial f_{tk}} = \partial c_{tk} * c_{(t-1)k} \quad (3.6.4.15)$$

$$\partial f_t = \partial c_t \circ c_{t-1} \quad (3.6.4.16)$$

$$\frac{\partial E}{\partial a_{tk}} = \frac{\partial E}{\partial c_{tk}} * \frac{\partial c_{tk}}{\partial a_{tk}} = \partial c_{tk} * i_{tk} \quad (3.6.4.17)$$

$$\partial a_t = \partial c_t \circ i_t \quad (3.6.4.18)$$

$$\frac{\partial E}{\partial c_{(t-1)k}} = \frac{\partial E}{\partial c_{tk}} * \frac{\partial c_{tk}}{\partial f_{(t-1)k}} = \partial c_{tk} * f_{tk} \quad (3.6.4.19)$$

$$\partial c_{t-1} = \partial c_t \circ f_t \quad (3.6.4.20)$$

Користећи претходно израчунате градијенте и формулу 3.6.4.5 можемо одредити вредности улаза и тиме завршити пропагацију у назад као што је приказано на слици 3.6.4.5. б). Формуле за израчунавање излаза дате су једначинама 3.6.4.21. до 3.6.4.25.

$$\partial \widehat{a_t} = \partial a_t \circ (1 - tanh^2(\widehat{a_t})) \quad (3.6.4.21)$$

$$\partial \widehat{i_t} = \partial i_t \circ i_t \circ (1 - i_t) \quad (3.6.4.22)$$

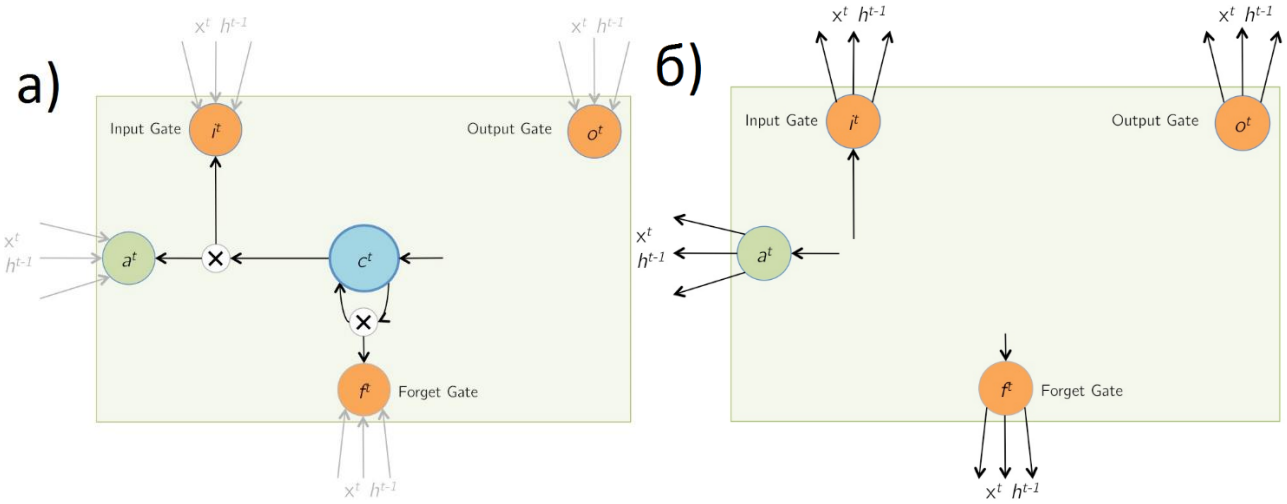
$$\partial \widehat{f_t} = \partial f_t \circ f_t \circ (1 - f_t) \quad (3.6.4.23)$$

$$\partial \widehat{o_t} = \partial o_t \circ o_t \circ (1 - o_t) \quad (3.6.4.24)$$

$$\partial I_t = W^T \partial z_t \quad (3.6.4.25)$$

У једначинама 3.6.4.21. до 3.6.4.24. израчунати су сви градијенти компонената вектора z_t, који је дефинисан формулом 3.6.4.5. На основу тих формула може се израчунати градијент за I_t формулом 3.6.4.25. Како је $I_t = \begin{bmatrix} x_t \\ h_{(t-1)} \end{bmatrix}$, могу се одредити градијенти ∂h_{t-1} и ∂x_t. На крају треба ажурирати вредности тежина по формули 3.6.4.26.

$$\partial W = \partial z_t (I_t)^T \tag{3.6.4.26}$$

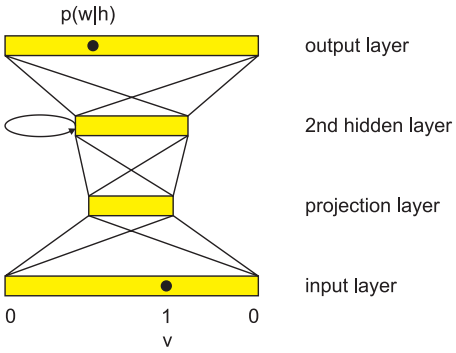


Слика 3.6.4.5. Пропагација у назад: а) корак 2: рачунање градијената за ћелију; б) корак 3: рачунање градијената за улазе и капије [71]

3.6.5. Креирање језичког модела помоћу дугих краткотрајних меморија

Као што је већ речено, највећи проблем рекурентних мрежа је то што су тешке за тренирање због ишчезавајућих градијената који се могу решити помоћу дугих краткотрајних меморија. У [72] креиран је језички модел помоћу дугих краткотрајних мрежа са два скривена слоја (слика 3.6.5.1). При улазу је примењена линеарна трансформација међу векторима речи тако да је варијанса скупа један и средња вредност је нула (трансформација у нормалну дистрибуцију), међутим аутори наводе да је то знатно успорило конвергенцију система. Уместо тога, трансформација бељења би била боља, али је тешко изводљива због велике димензионости улаза.

Постоје два скривена слоја, од којих први ради пројекцију улазних речи у континуални простор, а други се састоји од дугих краткотрајних меморија које су искомбиноване са различитим пројекцијама из претходног слоја. За експерименте је у другом скривеном слоју коришћено 150 и 200 неурона. Као што је већ коментарисано у 3.6.3.3. ради убрзања рада дугих краткотрајних меморија речи се могу поделити у класе и онда се рачуна производ вероватноћа да је реч у одређеној класи и да је та класа изабрана због претходног контекста (једначине 3.6.3.3 и 3.6.3.5).



Слика 3.6.5.1. Архитектура дуге краткотрајне меморије која моделује језик [72]

У излазном слоју је коришћена софтмакс функција како би се добиле одговарајуће нормализоване вредности вероватноће.

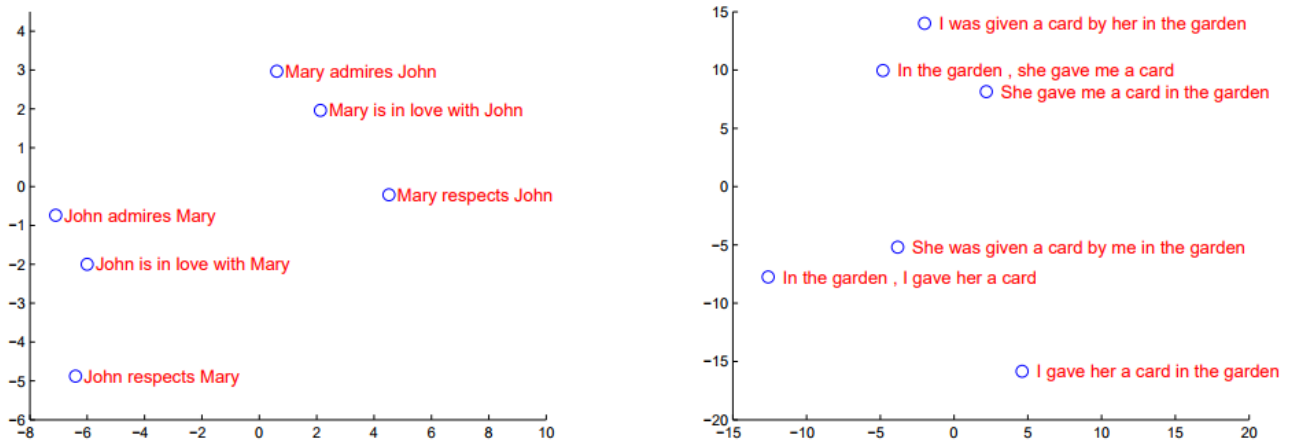
Као што се може применити овај модел је веома сличан моделима представљеним у поглављу о моделирању језика рекурентнм мрежама. Међутим коришћење дугих краткотрајних меморија у скривеном слоју даје 8% боље резултате [72] и најбоље резултате у поређењу са свим до сада представљеним техникама.

У раду [73] аутори су користећи исту методологију уместо једне речи предвидели секвенцу речи која следи на основу претходне секвенце (једначина 3.6.4.1), где је v векторска репрезентација секвенце. Тренирање је извршено тако да се максимизује логистичка вероватноћа тачне транслације T (секвенце која се предвиђа) изворне реченице S (једначина 3.6.4.2) у тренинг скупу $SENT$. По окончању тренинга, транслација се одређује формулом 3.6.4.3. приликом коришћења истренираног модела.

$$p(w_{n+1}, \dots, w_{n+m} | w_1, \dots, w_n) = \prod_{t=n+1}^{n+m} p(w_t | v, w_1, \dots, w_n) \quad (3.6.4.1)$$

$$\frac{1}{|SENT|} \sum_{(T,S,) \in SENT} \log p(T|S) \quad (3.6.4.2)$$

$$T = \arg\max_{T} p(T|S) \quad (3.6.4.3)$$



Слика 3.6.5.2. Репрезентација реченица векторима приказана у дводимензионом простору [73]

У овом раду коришћена су 4 слоја са 1000 неурона у сваком слоју и 1000 димензионим репрезентацијама речи. Улазни речник је био величине 160000, а излазни 80000. Резултати показују да мреже са више слојева раде боље и да сваки нови слој побољшава резултат за додатних 10%. Због величине ове мреже, она је имплементирана тако да се може извршавати паралелно на више рачунара. Пошто су на овај начин реченице представљене векторима, ти вектори се могу пројектовати на две димензије и приказати на графику (слика 3.6.4.2). Оваквим поступком се може и визуелно видети да су синтаксички и семнатички сличне реченице ближе међусобно. У претходним поглављима је објашњено да ће у овом раду бити речи о алгоритмима за моделирање језика, али не и о алгоритмима за синтаксичку анализу текста. Разлог томе је што су најмодернији алгоритми за моделирање језика у могућности да разумеју синтаксичку и семантичку сличност између речи, реченица и текстова, те се стога истим алгоритмима могу решити оба проблема.

3.7. Претварање речи у вектор (*word2vec*)

Главни проблем свих досадашњих алгоритама за моделовање језика помоћу неуралних мрежа је велика комплексност која онемогућава рад са огромном количином речи и великим речницима чак ни у случају паралелизације. За потребе компанија који раде са огромном количином података као што је *Google* овакви алгоритми нису били довољни за прављење лако доступних и брзих решења за преводиоце, претраживаче и друге алате. Због тога су они развили знатно бржи алгоритам под називом word2vec са бројним екстензијама и могућностима за коришћење.

Циљ овог модела је рачунање континуалне векторске репрезентације речи на основу веома великих скупова података. Модел треба да има пуно мању комплексност, а једнаке или боље резултате од до сада представљених језичких модела имплементираних неуралним мрежама. Остали модели могу да раде са стотинама милиона речи које су презентоване векторима са 50-100 димензија, док је у [74] конструисан алгоритам који може да ради са милијардама речи из речника који има кардиналност милион.

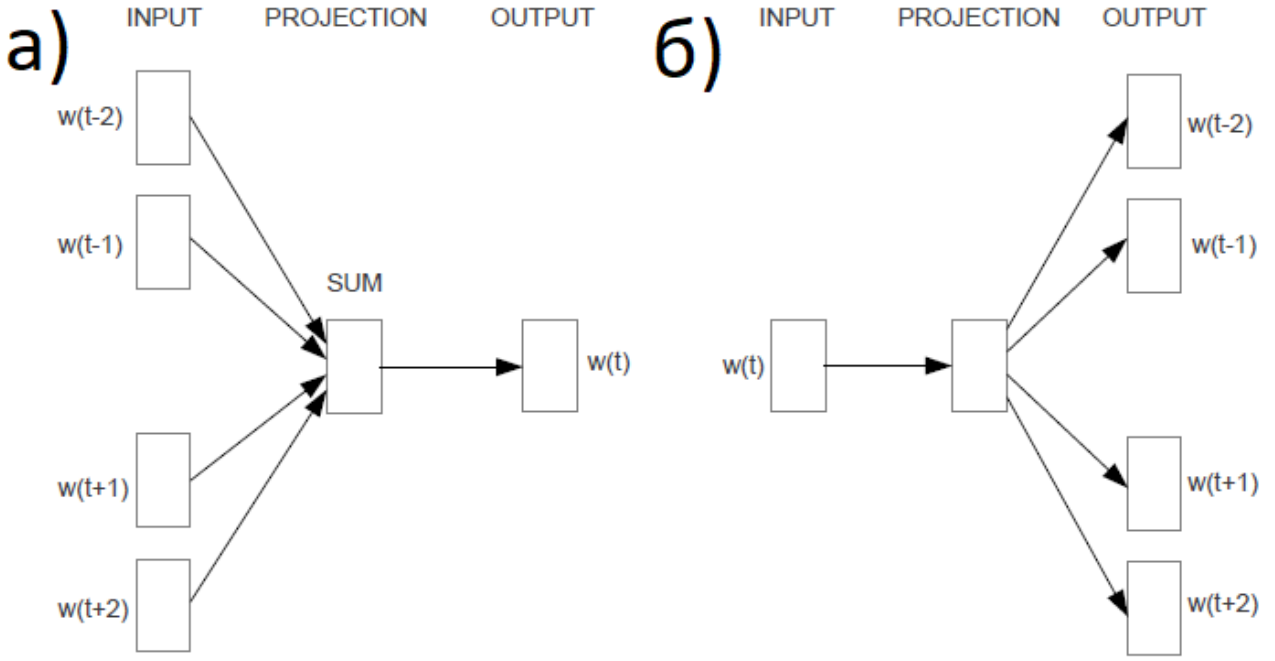
У овом раду се такође показује да речи имају више нивоа сличности. На пример, речи могу бити сличне зато што се завршавају сличним суфиксима (дакле у истом су падежу или глаголском времену). С друге стране, речи могу имати слично семантичко значење. Ове односе је могуће видети и представити у простору. У [74] показано је да је могуће постићи и више од тога. Наиме, линеарном комбинацијом вектора речи могуће је показати односе међу њима. У примеру датом једначином 3.7.1. показује се линеарна комбинација речи краљ, човек и жена, а у раду је показано да таква линеарна комбинација најприближније одређује вредност речи краљица што значи да је алгоритам у стању да разуме релације између речи.

$$vector(\text{"King"}) - vector(\text{"Man"}) + \text{vector ("Woman")} \approx vector(\text{"}Queen\text{"}) \qquad (3.7.1)$$

При опису претходних алгоритама говорено је о комплексности у смислу броја параметара, али треба имати у виду да сваки од тих параметара треба да се измени више пута. Ако је *O* број параметара, *E* је број епоха које се користе за тренинг и *T* је број речи у тренинг скупу, онда се укупна комплексност алгоритма може представити једначином 3.7.2. Једини параметар који се може смањити помоћу модела у овој једначини је број параметара. У једначинама 3.4.3.8 (број параметара код основних неуралних мрежа) и 3.6.3.2. (број параметара код рекурентних неуралних мрежа) се види да највећи број параметара представља сабирак H x V. То се може решити тако што се речник чија је кардиналност $|V|$ представи Хуфмановим бинарним дрветом тако да су најчешће речи ближе корену. Овим се комплексност тог сабирка смањује на $H \; x \; log_2 V$ и онда највећу комплексност даје нелинеарни скривени слој.

$$Q = O * E * T \qquad (3.7.2)$$

Идеја алгоритма представљеног у [74] је да се избаци нелинеарни скривени слој и архитектура се састоји од улазног слоја, пројекције улазног слоја и излазног слоја. Постоје два типа модела, континуална врећа речи (*continuous bag of words – CBOW*) и континуално изостављање н-грама (*continuous skip-gram*).



Слика 3.7.1. а) Модел континуалне вреће речи; б) Модел континуалног изостављања н-грама [74]

3.7.1. Континуална врећа речи

Континуална врећа речи (слика 3.7.1. а)) је слична језичком моделу заснованом на основним неуралним мрежама, осим што је избрисан нелинеарни скривени слој и пројекциони слој је дељен између свих речи. Због тога се све речи пројектују у исту позицију, тј. њихови вектори су усредњени. Како би се што боље искористио овакав систем улаз представља *N/2* речи које се појављују пре речи коју треба предвидети и *N/2* речи које се појављују после те речи. Дакле, ово је први алгоритам који одређује реч на основу њене прошлости и будућности.

Комплексност овог алгоритма је дата једначином 3.7.3, код које је *N* број речи на улазу, а *M* је број паралелних пројекција. Алгоритам је такав да је *N* = 10 или сличне величине, а *M* може имати величину 50 до 200 неурона. Овај модел користи континуалну дистрибуирану репрезентацију контекста за разлику од стандардне вреће речи.

$$O = N * M + M * log_2(V) \tag{3.7.3}$$

3.7.2. Континуално изостављање н-грама

Континуално изостављање н-грама (слика 3.7.1. б)) је слично архитектури претходног модела, али се на основу једне речи предвиђа *N/2* речи које су јој претходиле и *N/2* речи које су је следиле. У раду [74] је установљено да коришћење већег опсега *N* унапређује крајњи резултат, међутим успорава време извршавања. Пошто су речи које су даље од улазне речи у контексту мање везане за ту реч онда се њима даје мање значење мањим узорковањем на тим позицијама током тренинга. Циљ овог алгоритма је да максимизује средњу логаритамску вероватноћу дату једначином 3.7.2.1.

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t) \tag{3.7.2.1}$$

Комплексност овог алгоритма дата зависна је од растојања од улазне речи и величине пројекционог слоја и дата је једначином 3.7.2.2.

$$O = \frac{N}{2} * M + \frac{N}{2} * M * log_2(V) \quad (3.7.2.2)$$

Континуално изостављање н-грама даје најбољу тачност при одређивању семантичких веза у поређењу са другим алгоритмима заснованим на неуралним мрежама. У овом задатку континуална вреће речи има јако лоше перформансе као што је и очекивано јер вреће речи не садржи контексте. С друге стране оба алгоритма дају одличне резултате код синтаксичких веза. Брзина тренирања оваквих модела је много већа од свих претходно наведених модела. На пример, у поређењу са најбржим алгоритмом заснованим на основним неуралним мрежама, овај алгоритам је 9 пута бржи иако су вектори код неуралних мрежа имали 10 пута мању димензионост. Пример релација које алгоритам може да установи дат је на слици 3.7.2.1.

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Слика 3.7.2.1. Примери релација установљеним континуалним изостављањем н-грама [74]

Иако су резултати алгоритма континуалног изостављања н-грама фасцинантни у погледу брзине и тачности, аутори су успели чак и додатно да их побољшају у раду [75] изменом неколико детаља у алгоритму. Фокус је пре свега на алгоритам континуалног изостављања н-грама јер се он показао као бољи на основу резултата претходног рада.

Прво побољшање се односи на коришћење софтмакс функције у излазном слоју. Формулација софтмакс фунције је непрактична јер је рачунање градијента за њу пропорционално броју речи, што је ред величине 10^5-10^7. Због тога се у [75] предлаже измена названа хијерархијски софтмакс и алтернатива названа негативно узроковање.

i) Хијерархијски софтмакс

Ово је ефикасна апроксимација софтмакс функције, где се уместо евалуације свих излаза (|*V*|) како би се добила дистрибуција вероватноће, евалуира само $log_2(W)$ излаза. Хијерархијски софтмакс користи бинарно дрво за репрезентовање излазног слоја, при чему су листови неурони излазног слоја. Унутрашњи чворови представљају релативну вероватноћу између њихове деце чворова. Користећи овакву струкутуру вероватноће речи се могу одредити помоћу случајне шетње (*random walk*).

Свака реч се може достићи путањом од корена ка листовима. Нека је $n(w, j)$ j-ти неурон на стази од корена до речи w и $L(w)$ је дужина ове путање, тако да је $n(w, 1) = root$ и $n(w, L(w)) = w$. Нека је n унутрашни чвор, $ch(n)$ произвољно фиксно дете чвора n и нека $[\![x]\!]$

има вредност 1 ако је *x* тачно и -1 иначе. Онда се хијерархијски софтмакс може дефинисати формулом 3.7.2.3, при чему је σ логаритамска функција.

$$p(w|w_I) = \prod_{j=1}^{L(w)-1} \sigma([\![n(w,j+1) = ch\big(n(w,j)\big)]\!] * {v'_{n(w,j)}}^T v_{w_I}) \qquad (3.7.2.3)$$

На основу ове једначине види се да је сума свих вероватноћа један, а такође и да је цена рачунања градијената пропорционална дужини путање. Најбоље је за имплементацију користити бинарно Хуфманово дрво.

ii) Негативно узроковање

Негативно узорковање засновано је на одређивању контрастног шума (*NCE – Noise Contrastive Estimation*) која користи тезу да добар модел може да разликује сигнал од шума користећи логистичку регресију. Одређивање контрастног шума максимизује резултате софтмакс функције, али у алгоритму континуалног изостављања, довољно је само да се науче најбоље векторске репрезентације. За овај проблем могуће је одређивање контрастног шума поједноставити као што је приказано једначином 3.7.2.4. и то је дефинисано као негативно узорковање у [75].

$$\log \sigma\big({v'_{w_O}}^T v_{w_I}\big) + \sum_{i=1}^{k} E_{w_i \sim P_n(w)}[\log \sigma(-{v'_{w_i}}^T v_{w_I})] \qquad (3.7.2.4)$$

Ова функција се користи да се измени сваки логаритам вероватноће у алгоритму континуалног изостављања н-грама. Задатак је да се одреди реч из дистрибуције шума користећи логистичку регресију. Негативно узроковање користи само узорке, а не и дистрибуције вероватноћа за одређивање контрастног шума. Код оба алгоритма је $P_n(w)$ неодређени параметар који треба утврдити и у случају дистрибуције униграма $U(w)$ у [75] је одређено да најбоље резултате даје $P_n(w) = \frac{U(w)^{\frac{3}{4}}}{Z}$.

iii) Узорковање честих речи

Друга начин за убрзање алгоритма континуалног изостављања н-грама јесте да се речи узоркују с неком функцијом, тако да се узорковање честих речи смањи, а да се не промени редослед учесталости речи. Таква функција дата је једначином 3.7.2.5 која је описана у [75], али сличне особине имају и друге степене функције са степеном између 0 и 1. У тој формули $f(w_i)$ је фреквенција речи, а t је праг преко ког се смањује узорковање. Обично се вредност t поставља на 10^{-5}. Ова трансформација убрзава учење и значајно повећава тачност научених вектора.

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}} \qquad (3.7.2.5)$$

iv) Учење фраза

До сада је говорено само о речима, међутим, многе ствари су означене фразма, као што су главни град, научни часопис, дом здравља, итд. Због тога је битно научити фразе, тј. н-граме дужине веће од један и у [75] дато је побољшање које то и омогућава. Идеја је да се фразе формирају уколико се делови тих фраза појављују ретко, а цела фраза се појављује често. Ово се може добити формулом 3.7.2.6, где је δ коефицијент снижења који осигурава да се не формирају фразе које нису много честе.

$$score\big(w_i, w_j\big) = \frac{count(w_i w_j) - \delta}{count(w_i) * count(w_j)} \qquad (3.7.2.6.)$$

Овакав систем се може извршити 2-4 пута како би се одредиле фразе са више речи, а не само фразе са две речи. Фразе које су пронађене на овај начин могу се видети на слици 3.7.2.2.

Newspapers			
New York	New York Times	Baltimore	Baltimore Sun
San Jose	San Jose Mercury News	Cincinnati	Cincinnati Enquirer
NHL Teams			
Boston	Boston Bruins	Montreal	Montreal Canadiens
Phoenix	Phoenix Coyotes	Nashville	Nashville Predators
NBA Teams			
Detroit	Detroit Pistons	Toronto	Toronto Raptors
Oakland	Golden State Warriors	Memphis	Memphis Grizzlies
Airlines			
Austria	Austrian Airlines	Spain	Spainair
Belgium	Brussels Airlines	Greece	Aegean Airlines
Company executives			
Steve Ballmer	Microsoft	Larry Page	Google
Samuel J. Palmisano	IBM	Werner Vogels	Amazon

Слика 3.7.2.2. Примери откривених фраза [75]

3.8. Глобални вектори за репрезентацију речи (GloVe)

До сада су представљена два типа алгоритма: алгоритми засновани на факторизацији матрица и алгоритми засновани на неуралним мрежама, међутим они су међусобно комплементарни у смислу да први користи статистичке информације, али даје слабе резултате одређујући аналогије међу речима, а други даје веома добре аналогије, али не користи статистичке информације система јер је базиран на локалном контексту. Због тога су на универзитету *Stanford* развили алгоритам *GloVe (GlobalVectors)* који има одличне перформасе у одређивању аналогија, а у исто време користи глобални контекст [76].

Овај метод се базира на моделу најмањих квадрата који имају различите тежине и он се тренинра користећи број заједничких појављивања речи на глобалном нивоу. Ако је са X_{ij} означено заједничко појављивање речи i и j, онда се може дефинисати матрица заједничких појављивања X. Вероватноћа појављивања речи j у контексту речи i се може одредити формулом 3.8.1.

$$P_{ij} = P(j|i) = \frac{X_{ij}}{X_i} \tag{3.8.1}$$

Основна идеја алгоритма је да уколико се једна реч много чешће појављује у контексту једне речи него у контексту друге речи онда је та реч сличнија са првом него са другом. Овакав систем се може описати једначином 3.8.2, где је F функција коју треба одредити и повезати са вероватноћама, реч која нас интересује је $w_к$, а упоређујемо је са речима w_i и w_j.

$$F(w_i, w_j, w_k) = \frac{P_{ik}}{P_{jk}} \tag{3.8.2}$$

Пошто нас интересује однос између речи w_i и w_j, онда има смисла да F прикажемо као функцију разлике ова два параметра. Такође, пошто не треба мешати димензије вектора $w_к$ и разлике између вектора w_i и w_j, најбоље је одредити представити F као функцију скаларног производа ова два вектора. Уношењем ових измена у формулу 3.8.2. добијамо једначину 3.8.3.

$$F\left(\left(w_i - w_j\right)^T w_k\right) = \frac{P_{ik}}{P_{jk}} = \frac{F(w_i^T w_k)}{F(w_j^T w_k)} \tag{3.8.3}$$

Одавде се види да је за функцију F најбоље узети експоненцијалну функцију, чиме настају једначине 3.8.4 и 3.8.5. Функција $\log(X_i)$ из једначине 3.8.4. је константа, тј. не мења се у зависности од k, тако да се она може представити кроз померај b_i. Такође, проблем са логаритмом је да је недефинисан у нули, зато се у аргументу може додати јединица како би се избегли такви случајеви.

$$w_i^T w_k = \log(P_{ik}) = \log(X_{ik}) - \log(X_i) \tag{3.8.4}$$

$$w_i^T * w_k + b_i + b_k = \log(X_{ij}) \tag{3.8.5}$$

Идеја факторизације логаритма матрице заједничког понављања блиска је идеји латентне семантичке анализе која је заснована на декомпозицији матрица, али главни проблем је да су тежине свих парова једнаке, чак иако се неки парови врло ретко појављују. Због тога овај алгоритам као коначни циљ има оптимизацију модела код ког су заједничка понављања дата одређеном тежином као што је приказано формулом 3.8.6.

$$\mathrm{J} = \sum_{i,j}^{V} f\left(x_{ij}\right)\left(w_i^T w_j + b_i + b_j - \log X_{ij}\right)^2 \tag{3.8.6}$$

У тој формули треба одредити функцију $f\left(x_{ij}\right)$ која представља тежине сваког члана суме при чему та функција мора да има следеће особине:

1. $f(0) = 0$, ово је континуална функција која нестаје у 0;
2. Функција мора да буде неопадајућа како би ретка заједничка појављивања увек била мања од чешћих заједничких појављивања;
3. Функција треба да буде релативно мала за велике вредности параметра, тако да се честа заједничка појављивања не би додатно повећала вредност функције.

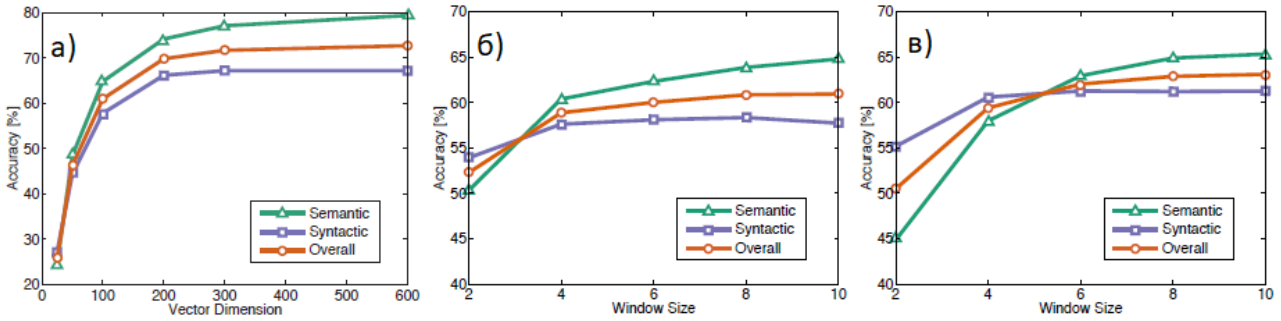
У [76] су предложили да функција има облик као у формули 3.8.7. јер је таква функција дала добре резултате у експерименталном истраживају. Интересантно је да је експонент који се користи у овој формули такође коришћен у *word2vec* алгоритму јер је и тамо дао најбоље резултате.

$$f(x) = \begin{cases} \left(\frac{x}{x_{max}}\right)^{\alpha}, & \text{ако је } x < x_{max} \\ 1, & \text{иначе} \end{cases} \tag{3.8.7}$$

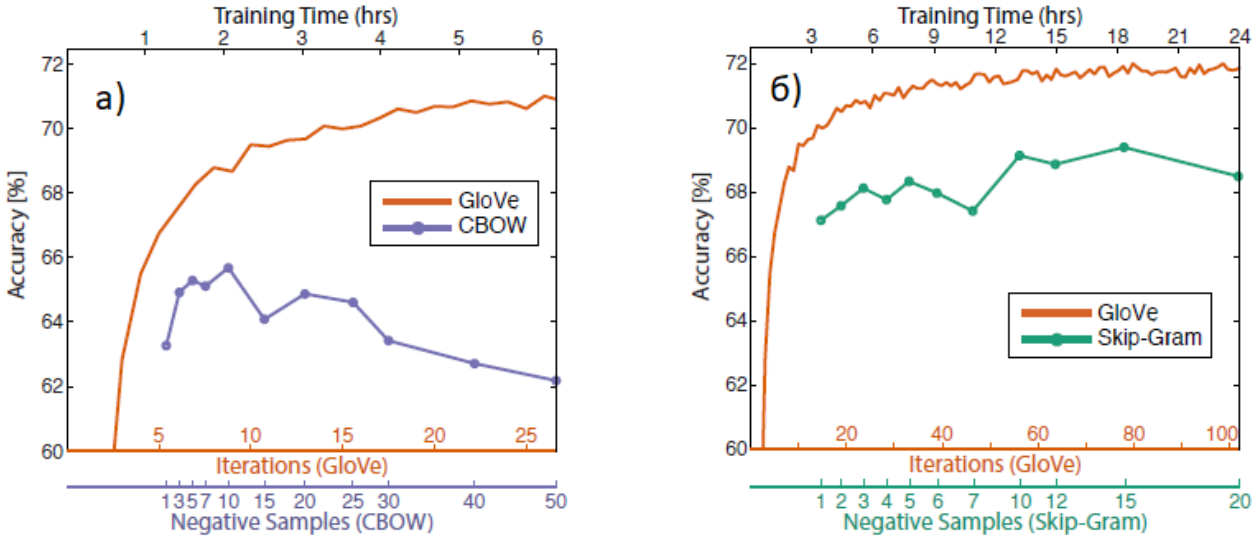
Комплексност модела зависи од броја елемената у X који су различити од нуле. Сама матрица има велике димензије, али су често многе вредности унутар ње нула, а број нула се може одредити ако се сматра да је број заједничких појављивања прати закон степена (power-low), тј. функција има велику вредност на почетку а затим брзо опада, што знали да има дугачак реп. [76]

Овај алгоритам зависи од величине вектора којим се представљају речи, од ширине прозора који се гледа, као и од тога да ли је прозор симетричан или не. Резултати за различите вредности за сваки од ових параметара дати су на слици 3.8.1. Време извршавања је зависно од брзине попуњавања матрице X и времена тренирања модела.

У резултатима [76] је показано да он даје најбоље резултате у поређењу са другим алгоримима у овој области. Интересантно је погледати како се овај алгоритам пореди са *word2vec* алгоритмом што се може видети на слици 3.8.2, где се мери тачност над различитим бројем итерација.



Слика 3.8.1. а) тачност за различите вредности димензија вектора; б) тачност за различите врсте ширине прозора (симетрични контекст); в) тачност за различите врсте ширине прозора (асиметрични контекст) [76]



Слика 3.8.2. а) поређење тачности између алгоритама GloVe и CBOW; б) поређење тачности између алгоритама GloVe и Skip-gram [76]

3.9. Имплементација језичких модела помоћу неуралних мрежа

Као што је се може видети у претходним поглављима, неуралне мреже су постале веома популарне за имплементацију језичког модела зато што дају боље резултате од претходних методологија и зато што се могу имплементирати у дистрибуираном окружењу те се могу убрзати паралелним извршавањем. Такође, настанком *word2vec* и *GloVe* алгоритама и сама комплексност алгоритма је смањена драстично. Због свих ових предности постоје вишеструке имплементације неуралних језичких модела у разним језицима и библиотекама.

Овде треба напоменути и да је област дубоког учења и машинског учења неуралним мрежама врло развијена, тако да су разни модели неуралних мрежа имплементирани и оптимизовани у бројним окружењима. Постоје чак и хардвери и софтвери који су направљени првенствено за рад са оваквим проблемима као што је *TensorFlow*.

У [77] дат је преглед најкоришћенијих оквира за рад са неуралним мрежама које су описане у даљем тексту.

TensorFlow је направљен од стране компаније Google на ниском нивоу као библиотека за *python* програмски језик. Међутим, сада је доступан и за Java, C и Go програмске језике. Поред тога што су у њему имплементиране све горе наведене мреже, у њему су такође већ

имплементирани *word2vec* и језички модел помоћу рекурентних неуралних мрежа и дугих краткотрајних меморија. Поред тога имплементиран је и модел који омогућава векторизацију реченица као што је описано у другом делу одељка 3.6.5. Проблем ове библиотеке јесте што ради на веома ниском нивоу, па се уместо ње често користи *Keras* који је на вишем нивоу, а базиран је на овој библиотеци.

Keras је библиотека за *python* програмски језик која може да ради над библиотекама *TensorFlow* или *Theano* јер је библиотека вишег нивоа. Има добру документацију и треба је користити за брзу имплементацију неуралних мрежа по већ постојећим моделима.

Theano је једна од најстаријих библиотека за рад са неуралним мрежама, ради са програмским језиком *python*, веома је стабилна и такође је имплементирана на ниском нивоу.

Lasagne су библиотека имплементирана на високом нивоу која ради над библиотеком *Theano*. У њој се могу наћи имплементације за разне неуралне мреже које се могу одмах користити.

Caffe може да ради са програмским језицима *python* и *matlab*, али уколико је потребно додати нову имплементацију, то се може урадити у програмским језицима *C++* и *CUDA*. Његова предност је што је веома стабилан, међутим документација је веома лоша.

Torch је постао веома познат зато што је био коришћен од стране компанија *Facebook* и *DeepMind*, међутим с настанком библиотеке *TensorFlow* његова популарност се губи. Највећи проблем је што је имплементиран над језиком *Lua*.

Mxnet је библотека која може да ради над вероватно највише језика, међу којима су *python*, *R*, *C++* и *Julia*. Компанија *Amazon* је користи за дубоко учење и рад са неуралним мрежама пре свега због добре скалабилности.

DL4J је библиотека за програмске језике *Java*, *Scala* и *Closure* у којој су имплементиране многе неуралне структуре. Има јако добру документацију и прати трендове у овој области.

Cognitive Toolkit је развијен од стране компаније *Microsoft* који може да скалира и хоризонтално и вертикално веома добро. Он подржава језике *python* и *C++*.

Ове библиотеке имају имплементиране различите неуралне мреже чији се слојеви могу комбиновати међусобно или користити за имплементацију језичких модела. Међутим, ово је списак само најпознатијих библиотека у овој области, док се много шири преглед може наћи у [78]. Тамо се такође налазе линкови ка специфичним скриптама и имплементацијама много ширег распона, а за сваки је дат кратак опис.

У овом раду је фокус пре свега на језичке моделе који су представљени и за сваки од њих постоји бар једна имплементација. Језички модел базиран на неуралним мрежама имплементиран у *python* програмском језику је доступан за коришћење у [79]. Имплементација језичког модела помоћу конволуционих неуралних мрежа креирана помоћу *torch* библиотеке и конволуционих модула тестираних на језику *CUDA* дата је у [80]. С друге стране у [81] дат је туторијал и код за класификацију текстова коришћењем библиотеке *TensorFlow*.

Иако многе горе поменуте библиотеке садрже готове имплементације за рекурентне неуралне мреже у [82] дат је опис и код за имплементирање истих користећи програмски језик *python* и библиотеку *Theano*. Као што је већ речено ова библиотека је на ниском нивоу па сама не садржи имплементацију рекурентих неуралних мрежа, међутим може се коритити библиотека *Lasagne* или туторијал за имплементацију истих и њихову употребу за моделирање

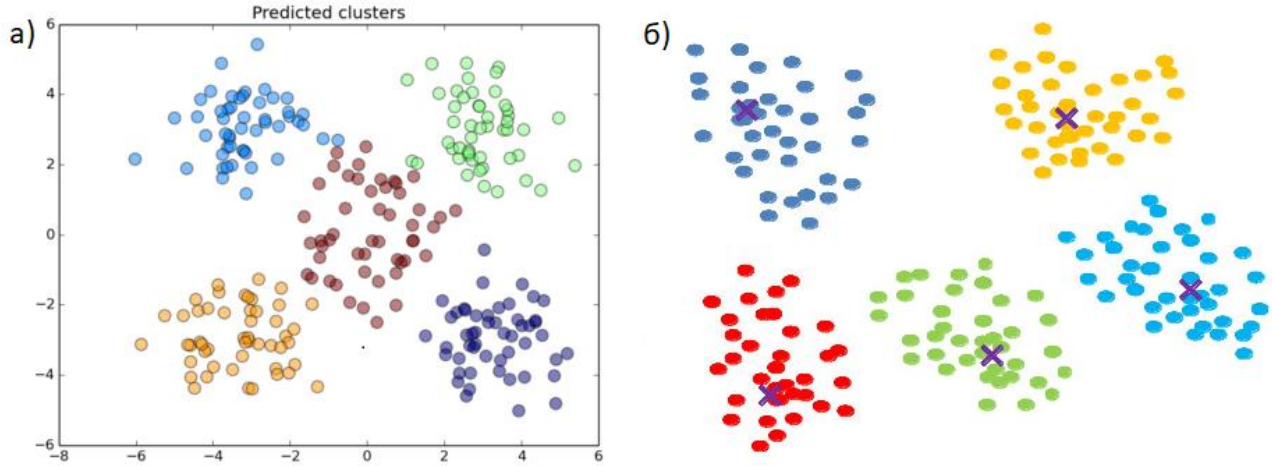
језика. У [83] дата је имплементација језичког модела помоћу рекурентних мрежа користећи библиотеку *TensorFlow*.

Постоје и многе друге имплементације за ове алгориме и као што се у претходном тексту могло видети сваки од алгоритама има унапређења која се могу користити, тако да коначни избор језика, библиотеке и алгоритма зависи од проблема и преференци оног ко имплементира решење. Пример решавања једног проблема овим алгоримима од стране аутора овог мастер рада дат је у поглављу 5.

4. МЕТОДЕ ЗА КЛАСТЕРИЗАЦИЈУ ТЕКСТОВА

Пошто је циљ овог рада да кластеризује научне радове у групе на основу њихових апстракта и наслова, испод су дате основне технике кластеризације које су и данас веома популарне, као и неколико техника које су базиране на неуралним мрежама зато што су то углавном варијанте већ представљених модела. Аутор је изабрао кластеризацију јер сматра да је ненадгледано (*unsupervised*) учење једини прави приступ при обради овако велике количине текста коју је врло тешко лабелирати.

Идеја свих ових алгоритама јесте да су неки улази ближи међу собом и да се због тога могу наћи подскупови улазног скупа такви да су тачке унутар сваке од група близу једне другима, а да су те групе одвојене међусобом (слика 4.1 а)). Постоје случајеви када групе могу да се преклапају, а такође је могуће користити фази кластеризацију која би одредила вероватоћу припадности неког елемента свакој од група, међутим ови алгоритми нису у опсгу овог рада.



Слика 4.1. а) Пример резултата кластеризације [84]; б) Пример кластеризације методом *k* средњих вредности [85]

4.1. Кластеризација методом *k* средњих вредности

Ово је итеративни алгоритам који дели улазни скуп у *k* група, где се *k* унапред задаје и алгоритам је према томе добио име. Свака група има особину да су тачке које се налазе у тој групи унутар кружнице са центром који је налази на основу улазних тачака. Почетни центри се задају као улаз у прву итерацију алгоритма. Алгоритам се састоји из два корака који се извршавају с понављањем:

1. Израчунају се групе на основу задатих центара, тако да се минимизује сума квадратних растојања улаза од центра групе којој се тај улаз додељује (једначина 4.1.1). C_k представља *k*-ту групу, m_k је центар те групе, а N је број улазних примера. [86]
2. Центри се ажурирају тако да представљају средину улаза које се налазе у одговарајућим групама (једначина 4.1.2). [87]

$$\mathrm{E} = \sum_{\mathrm{к}=1}^{K} \sum_{x \in C_k} d(x, m_k) = \sum_{\mathrm{к}=1}^{K} \sum_{x \in C_k} \sum_{n=1}^{N} (x_n - m_{kn})^2 \qquad (4.1.1)$$

$$m_k = \frac{\sum_{i=1}^{n_k} x_n}{n_k} \tag{4.1.2}$$

Овај алгоритам се често употребљава јер је веома једноставан и с обзиром на то даје прилично добре резултате. Међутим, постоји пуно детаља на које треба обратити пажњу при имплементацији. Алгоритам не стиже увек до глобалног оптимума и заправо је вероватније да ће завршити у локалном оптимуму. Због тога је иницијализација, тј. постављање првих центара јако битан корак. Центре нипошто не треба иницијализовати у истој тачки и најбоље је иницијализовати их случајно генерисаним вредностима или случајно изабрати примере из улазног скупа.

Како би се даље побољшао резултат, алгоритам се може извршити више пута са различитим иницијализацијама центра и онда се може узети најбоље решење које не мора да буде и глобални минимум. Дакле, при имплементацији се вага између брзине извршавања и тачности, с тим да веће време извршавања не мора да значи да ће се наћи боље решење. Сложеност овог алгоритма је *O(tKn)*, где је *t* број итерација.

Недостатак овог метода је што треба специфицирати *k* унапред. Такође, овај алгоритам не уме да се ради добро са шумом и изузецима (outlier). У стварном свету скупови су ретко подељени тако да се могу представити непреклапајућим кружницама, као што овај алгоритам ради, тако је понекад боље користити алгоритме који деле улаз у неконвексне скупове. На крају, овај алгоритам не може да ради са категоричким подацима због коришћења средњих вредности. За неке од ових проблема постоје екстензије основног алгоритма које их решавају.

Добијени кластери могу се оценити различитим методама. Уколико се унапред зна који улаз припада ком кластеру, могуће је оценити тачност, комплетност, *F1* скор, међутим најчешће то није случај, као ни у овом раду. За ненадгледану евалуацију користи се сума квадрата, нормализовани заједнички скор (*NMI – normalized mutual information*), хомогеност, комплетност, *V* мера (*V-measure*), *FMI* (*Fowlkes-Mallows Index*), Шилоетов коефицијент (*Silhouette Coefficient*) [88] и случајни индекс (*Rand Index*) [89]. У наредним одељцима ће се детаљније представити хомогеност, нормализовани заједнички скор и случајни индекс.

4.1.1. Хомогеност

Да би се израчунала хомогеност, сваком кластеру се додели класа тако да највећи број елемената унутар те групе припада тој класи, а онда се тачност кластеризације рачуна као количник суме коректно додљених улаза за сваки кластер и укупног броја улаза (једначина 4.1.1.1). Ω је скуп кластера, а C је скуп додељених класа. Велику хомогеност је лако достићи када је број кластера велики, на пример хомогеност је максимална (једнака 1) ако је сваки улаз класа за себе. Оваква особина мере није добра јер се квалитет кластера не одређује на одговарајући начин. [89]

$$f(\Omega, C) = \frac{1}{N}\sum_k \max_j |w_k \cap c_j| \tag{4.1.1.1}$$

4.1.2. Нормализовани заједнички скор

Нормализовани заједнички скор решава проблем мере хомогености јер је помоћу нормализованог заједничког скора могуће вагати између броја класа и квалитета кластера. Нормализовани заједнички скор је однос између заједничке информације (која се рачуна између стварних класа и добијених кластера) и ентропије кластера (која је већа што је већи број кластера). Овај скор дат је једначином 4.1.2.1, при чему је количина заједничке информације дата формулом 4.1.2.2, а ентропија је дата формулом 4.1.2.3. [89]

$$NMI(\Omega, C) = \frac{I(\Omega,C)}{\frac{|H(\Omega)+H(C)|}{2}} \quad (4.1.2.1)$$

$$I(\Omega, C) = \sum_k \sum_j P(w_k \cap c_j) log \frac{P(w_k \cap c_j)}{P(w_k)P(c_j)} = \sum_k \sum_j \frac{|w_k \cap c_j|}{N} log \frac{N|w_k \cap c_j|}{|w_k||c_j|} \quad (4.1.2.2)$$

$$H(\Omega) = -\sum_k P(w_k) log P(w_k) = -\sum_k \frac{|w_k|}{N} log \frac{|w_k|}{N} \quad (4.1.2.3)$$

4.1.3. Случајни индекс

Случајни индекс се рачуна помоћу тачних позитивних (TP), тачних негативних (TN), нетачних позитивних (FP) и нетачних негативих (FN) вредности. Вредност је позитивна ако припада истом кластеру, а негативна ако не припада истом кластеру. Вредност је тачна ако припада истој класи, а нетачна ако припада различитој класи. На основу ових дефиниција, случајни индекс се дефинише као количиних тачних вредности и свих вредности (једначина 4.1.2.4). [89]

$$RI = \frac{TP+TN}{TP+FP+FN+TN} \quad (4.1.2.4)$$

4.1.4. Имплементација алгоритма k средњих вредности

Овај алгоритам је најчешће коришћени ненадгледани алгоритам, тако да његова имплементација постоји у свим језицима и многим библиотекама. У програмском језику *python* је најбоља библиотека *scikit-learn*, у језицима *Java* и *C/C++* је део библиотеке за машинско учење, а у језику *matlab* је имплементиран директно као функција.

4.1.5. Имплементација евалуационих функција

Све горе наведене функције су имплементиране у библиотеци *scikit-learn* [88] на програмском језику *python*. Оне су такође имплементиране у језику *matlab*, а постоје и библиотеке за имплементацију нормализованог заједничког скора на језику *C++*.

4.2. Хијерархијска кластеризација

За имплементацију хијерархијске кластеризације потребно је одредити матрицу сличности између улаза *X*. Затим се над том матрицом примењују следећи кораци:

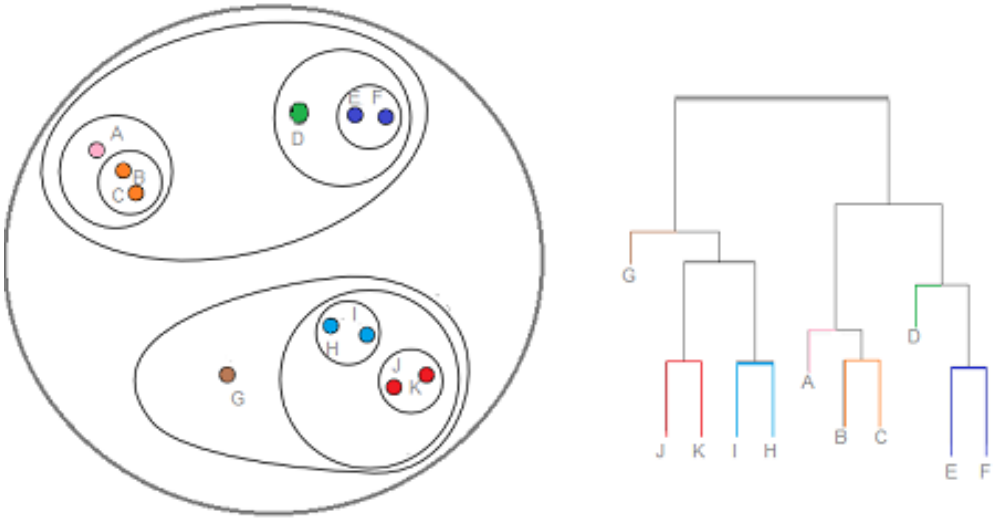
1. Сваком улазном елементу се додели кластер, тако да има укупно *N* кластера (исто као број елемената)
2. Наћи најближи пар кластера и ставити их у исти кластер, чиме се број кластера смањује за један.
3. Израчунати растојања између новог кластера и сваког од старих кластера
4. Понављати кораке 2 и 3 све док се не добије један кластер величине *N*

Корак три се може урадити на различите начине, према чему се разликују кластеризације базиране на минималној вези (single-linkage) од оне базиране на максималној вези (complete-linkage) и оне базиране на средњој вези (average-linkage).

Код кластеризовања заснованог на минималној вези, растојања између кластера се рачунају као растојања између најближих елемената тих кластера. С друге стране, код кластеризације засноване на максималној вези растојања се рачунају између најдаљих

елемената два кластера, а код средње везе између средње вредности растојања свих елемента из једног кластера са свим елементима из другог кластера. [90]

На основу овог алгоритма формира се стабло улаза, тако да се са сваком новом итерацијом (понављање корака 2 и 3) повећава ниво стабла, што значи да се на сваком нивоу спајају два кластера у један (слика 4.2.1).



Слика 4.2.1. Резултат хијерархијске кластеризације и дрво које тим путем настаје [91]

Хијерархијска кластеризација се користи врло често, а има примену такође код кластеризације комплексних мрежа (*complex networks*), тако да има бројне имплементације. У програмском језику *python* је део библиотеке *scipy*. У [92] је дата имплементација у програмском језику *Java*, а постоји слична имплементација и на језику *C*. У језику *matlab* је описано у документацији како имплементирати ову методу.

4.3. Спектрална кластеризација

Спектрална кластеризација је једна од најчешћих метода за груписање и биће коришћена у овом раду такође. Прво се као и код хијерархијске кластеризације формира матрица сличности између елемената које треба кластеризовати. Алгоритам користи спектар те матрице, што су заправо сопствене вредности, да смањи димензионост улазног скупа. На крају се над тим скупом врши кластеризација неком другом методологијом.

Ако се улази посматрају као чворови графа, а сличности између њих као тежине линкова између чворова, онда се може дефинисати степен чвора. Уколико је тежина линка нула сматра се да тај линк не постоји. Степен чвора је број линкова (тј. тежина већих од 0) које повезују тај чвор са другим чворовима. Дијагонална матрица D је матрица степена чворова, тј. d_{ii} је степен чвора i, а сви остали елементи су 0. Нека је X матрица сличности између елемената, онда се може дефинисати Лапласова матрица L као у формули 4.3.1. Постоје и други начини да се дефинише Лапласова матрица, а најчешће се користе формуле 4.3.2 или 4.3.3. које су нормализоване. [93]

$$L = D - X \tag{4.3.1}$$

$$L_{sym} = I - D^{\frac{1}{2}} X D^{-\frac{1}{2}} \tag{4.3.2}$$

$$L_{rw} = I - D^{-1} X \tag{4.3.3}$$

Матрица L има следеће особине:

1. За сваки вектор $f \in R^n$ важи једначина 4.3.4;
2. То је симентрична и позитивна полу-дефинитна матрица;
3. Најмања сопствена вредност матрице је *0*, а одговарајући сопствени вектор је јединични вектор;
4. Има ненегативне и реалне сопствене вредности $0 \leq \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n$. [93]

$$f'Lf = \frac{1}{2}\sum_{i.j=1}^{n} w_{ij}(f_i - f_j)^2 \quad (4.3.4)$$

Нормализована Лапласова матрица има следеће особине:

1. За сваки вектор $f \in R^n$ важи једначина 4.3.5;
2. λ је сопствена вредност матрице L_{rw} са сопственим векторима u акко је λ је сопствена вредност матрице L_{sym} са јединичним векторима $w = D^{\frac{1}{2}}u$;
3. λ је сопствена вредност матрице L_{rw} са сопственим векторима u акко је λ и u решавају једначину $Lu = \lambda Du$;
4. *0* је сопствена вредност матрице L_{rw} са константним јединичним сопственим вектором. *0* је сопствена вредност матрице L_{sym} са константним јединичним сопственим вектором $D^{\frac{1}{2}}$;
5. L_{sym} и L_{rw} су позитивне полу-дефинитне матрице и имају n ненегативних и реалних сопствених вредности $0 \leq \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n$. [93]

$$f'Lf = \frac{1}{2}\sum_{i.j=1}^{n} w_{ij}(\frac{f_i}{\sqrt{d_i}} - \frac{f_j}{\sqrt{d_j}})^2 \quad (4.3.5)$$

Због различитих дефиниција Лапласове матрице и других детаља овог алгоритма постоји више верзија спектралне кластеризације, а у наредним одељцима ће бити представљени кораци за три верзије.

4.3.1. Ненормализована спектрална кластеризација

1. Израчунати матрицу сличности и на основу ње израчунати Лапласову матрицу користећи формулу 4.3.1
2. Одредити првих *k* сопствених вектора $u_1, u_2, ..., u_k$ Лапласове матрице *L*.
3. Нека је $U \in R^{nxk}$ матрица чије су колоне сопствени вектори $u_1, u_2, ..., u_k$.
4. Кластеризовати редове матрице U користећи алгоритам *k* средњих вредности. [93]

4.3.2. Нормализована спектрална кластеризација (Shi i Malik - 2000)

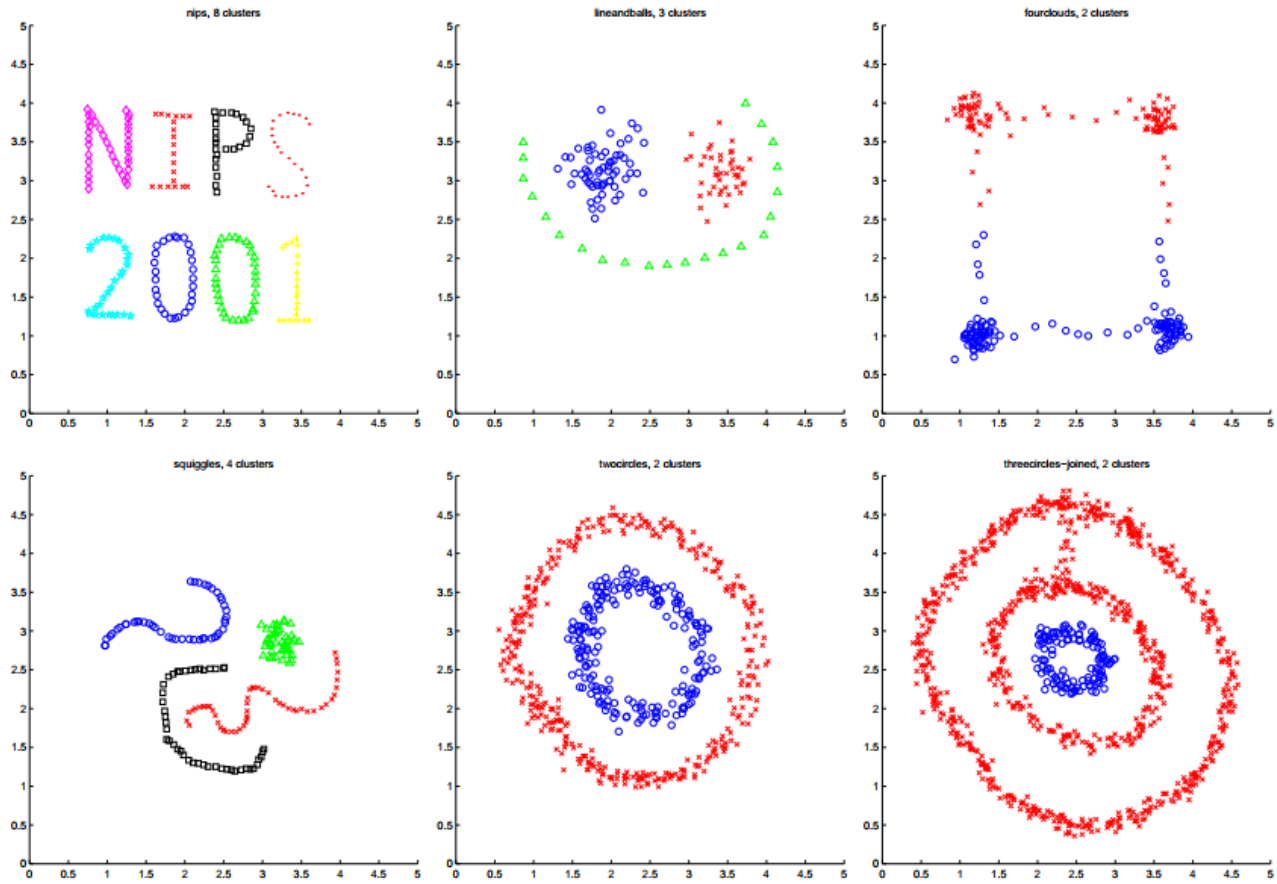
1. Израчунати матрицу сличности и на основу ње израчунати Лапласову матрицу користећи формулу 4.3.3.
2. Одредити првих *k* сопствених вектора $u_1, u_2, ..., u_k$ Лапласове матрице L_{rw}.
3. Нека је $U \in R^{nxk}$ матрица чије су колоне сопствени вектори $u_1, u_2, ..., u_k$.

4. Кластеризовати редове матрице U користећи алгоритам k средњих вредности. [93]

4.3.3. Нормализована спектрална кластеризација (Ng, Jordan и Weiss - 2001)

1. Израчунати матрицу сличности и на основу ње израчунати нормализовану Лапласову матрицу користећи формулу 4.3.2.
2. Одредити првих k сопствених вектора $u_1, u_2,..., u_k$ Лапласове матрице L_{sym}.
3. Нека је $U \in R^{nxk}$ матрица чије су колоне сопствени вектори $u_1, u_2,..., u_k$.
4. Формирати матрицу T $\in R^{nxk}$ нормализацијом редова матрице U на норму 1, тако да се елемент матрице T може представити као $t_{ij} = \frac{u_{ij}}{(\sum_k u_{ik}^2)^{\frac{1}{2}}}$.
5. Кластеризовати редове матрице T користећи алгоритам k средњих вредности. [93]

Спектрална кластеризација је једна од бољих и коришћенијих техника, али нису све варијанте одговарајуће, нити ће увек дати добро решење. Ng, Jordan и Weiss објашњавају у њиховом раду [94] детаљне услове под којима ће њихов алгоритам дати добро решење и приказују резултате њиховог алгоритма за различите улазе. Неки примери дати су на слици 4.3.3.1, где се може видети да за разлику од алгоритма k средњих вредности, спектрална кластеризација може да одреди групе различитих облика.



Слика 4.3.3.1. Примери разултата спектралне кластеризације [94]

4.4. Кластеризација помоћу рекурентних неуралних мрежа

У овом и наредном потпоглављу биће описане технике кластеризације помоћу неуралних мрежа. Ове технике нису тако популарне, али се често могу користити заједно са моделирањем језика помоћу ових неуралних мрежа и због тога су укратко описане овде.

У потпоглављу 3.7 описан је алгоритам који на основу речи из околине (при чему је околина фиксне дужине) одређује дату реч или на основу дате речи одређује околину фиксне дужине. Међутим, реченице, параграфи и друге смислене семантичке целине у језику немају фиксну дужину, па је питање да ли постоји начин да се семантичке целине претворе у вектор, а затим кластеризују. Наравно, једна могућност је да се саберу вектори речи и фраза унутар те семантичке целине, али то не гарантује да ће решење бити одговарајуће.

У [95] развијен је алгоритам за представљање параграфа (а може се користити и за друге семантичке целине) помоћу вектора. Да бисмо кластеризовали документе, апстракте, наслове, реченице, параграфе, потребно је само да урадимо једну од метода за кластеризацију над овако добијеним векторима. То је један од алгоритама који ће бити тестиран у наредном поглављу.

Овај алгоритам се базира на алгоритму представљеном у потпоглављу 3.7. Постоје такође два типа алгоритама, један базиран на континуалној врећи речи и други базиран на изостављању н-грама. Структура мреже остаје иста у оба случаја, мења се само врста улаза.

4.4.1. Дистрибуирани меморијски модел вектора параграфа

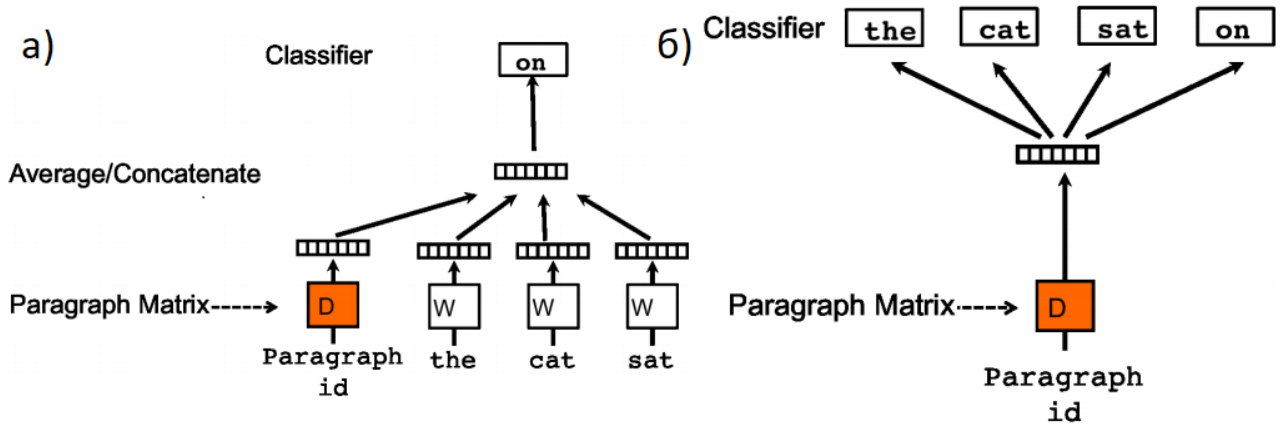
Дистрибуирани меморијски модел вектора параграфа (*Distributed Memory Model of Paragraph Vectors – PV-DM*) се ослања на алгоритам континуалне вреће речи, представљен у 3.7.1. Структура мреже остаје иста, али се улази мењају као што се може видети на слици 4.4.1.а). Као први улаз се поставња вектор параграфа, а остали улази су вектори речи које припадају делу тог параграфа. У средњем слоју се ти вектори могу усредњити или надовезати. У експерименталном делу рада [95], користи се надовезивање вектора. Излаз је следећа реч у контексту која се надовезује на речи на улазу и део је тог параграфа.

У овом контексту вектор параграфа се може сматрати као још једна реч која може да запамти контекст или тему тренутног параграфа. Број улазних речи је константан и добија се померањем прозора кроз параграф, а вектор параграфа је заједнички за све улазе из тог параграфа. Вектор неке речи је заједнички за све параграфе. Вектори параграфа и речи се добијају применом стохастичког градијентног спуста, при чему се градијенти рачунају пропагацијом у назад за мрежу. У току тестирања, користе се фиксни вектори за речи који су научени у фази тренирања, а уче се вектори нових параграфа.

Сложеност овог алгоритма дата је једначином 4.4.1.1, при чему је N број параграфа у корпусу, M број речи у речнику, p је број димензија вектора параграфа, а q је број димензија вектора речи. Иако је N велико, а самим тим је и сложеност велика, често се ажурира само неколико параметара у мрежи, тако да је алгоритам ефикасан.

$$O = N * p + M * q \qquad (4.4.1.1)$$

По добијању вектора параграфа, они се могу кластеризовати помоћу алгоритма k средњих вредности или неког другог од горе наведених алгоритама. Предност овог алгоритма је у томе што се вектори параграфа уче ненадгледано и због тога се могу имплементирати над било којим текстовима. [95]



Слика 4.4.1. а) Дистрибуирани меморијски модел вектора параграфа; б) Одређивање вектора параграфа моделом дистрибуиране вреће речи [95]

4.4.2. Вектор параграфа базиран на дистрибуираној врећи речи

Вектор параграфа базиран на дистрибуираној врећи речи (*Distributed Bag of Words version of Paragraph Vector - PV-DBOW*) се заснива на континуалном изостављању н-грама представљеном у одељку 3.7.2. У овом алгоритму једини улаз је вектор параграфа, а излази су случајно изабране речи из тог параграфа. Да би се истренирао овакав модел речи на излазу се постављају тако што се изабира случајни прозор у параграфу, а затим се изабира случајна реч у том прозору. У овом моделу је довољно складиштити само тежине излаза, а није потребно складиштити векторе речи као у претходном моделу. [95]

Пошто постоје два начина на које се могу добити вектори параграфа, ти вектори се могу комбиновати за још боље решење. У раду [95] упоређивани су резултати различитих алгоритама са векторима параграфа и комбиноваање вектора добијених помоћу *PV-DM* и *PV-DBOW* је дало најбоље резултате. У поглављу пет овог мастер рада, биће упоређен алгоритам *PV-DM* са другим најбољим алгоритмима са којима није било поређења у [95].

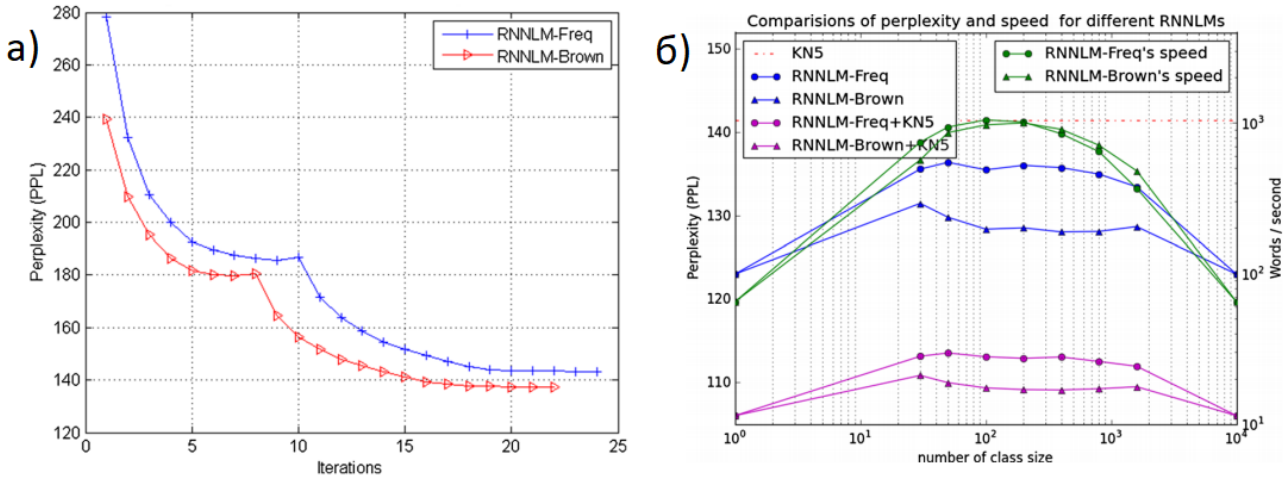
4.4.3. Кластеризација речи помоћу рекурентних неуралних мрежа

У раду [96] се проширује коришћење рекурентих неуралних мрежа при моделовању језика тако да се укључи кластеризовање речи у групе у моделовање језика. За кластеризацију се користи Браонов алгоритам (*Brown clustering*) за естимацију језичког модела базираног на класама. Ово је модел хијерархијске кластеризације речи који се често користи у процесуирању природног језика. Он оптимизује функцију дату у 4.4.3.1. где су *V*, *C* и *T* кардиналост речника, број кластера речи и величина корпуса текста, редом. Ова формула се може написати и као 4.4.3.2, где n_{w_i} представља број појављивања w_i у корпусу.

$$E(C) = -\frac{1}{|T|} logP(w_1 \dots w_{|T|}) = -\frac{1}{T} log \prod_{i=1}^{|T|} P(C(w_i)|C(w_{i-1})P(w_i|C(w_i))) \quad (4.4.3.1)$$

$$E(C) = -\frac{1}{|T|}\sum_{i=1}^{|T|} log \frac{n_{c_i,c_{i-1}}}{n_{c_{i-1}}} * \frac{n_{w_i}}{n_{c_i}} = -\sum_{c,c' \in C} P(c,c') log \frac{P(c,c')}{P(c)P(c')} - \sum_{w \in V} P(w) logP(w) \quad (4.4.3.2)$$

У раду [96] Браонова кластеризација замењује кластеризацију речи која је дата као оптимизација језичког модела, описаног у 3.6.3, развијеног помоћу рекурентних неуралних мрежа. Првенствени циљ те кластеризације био је убрзање, а док се у раду [96] фокусирају на што боље резултате. На слици 4.4.3.1. дато је поређење између резултата и брзине ових алгоритама. [96]



Слика 4.4.3. а) Поређење перформанси кластеризације речи између алгоритама описаних у 3.6.3. и 4.4.3 за различити број итерација (мање је боље). б) Поређење перформанси и брзине кластеризације речи помоћу н-грама са 5 речи, алгоритама описаних у 3.6.3 и 4.4.3 и њихових комбинација са н-грамима [96]

4.4.4. Кластеризација помоћу дугих краткотрајних меморија

До сада приказане кластеризације помоћу неуралних мрежа се ослањају на језички модел и чињеницу да се кластеризују документа, међутим неуралне мреже су у стању да раде генералну кластеризацију тако што се уче да производе резултате који оптимизују функцију грешке. У овом одељку биће дат такав пример помоћу дугих краткотрајних меморија описан у раду [97]. Тамо су дата два циља за ненадгледано учење: оптимизација добитка бинарних информација (*Binary Information Gain Optimization – BINGO*) и оптимизација непараметарске ентропије (*Nonparametric Entropy Optimization – NEO*). Помоћу ових оптимизационих функција дуга краткотрајна меморија учи да разликује различите секвенце и да их групише по њиховим одликама.

Оптимизација добитака бинарних информација кластеризује нелабелиране податке максимизацијом информација добијених из обсервација излаза мреже са једним скривеним слојем са активационом логистичком функцијом. Сваки неурон у излазу је један бит, а број класа је 2^{n_o}, где је n_o број неурона у излазном слоју. Тежине се ажурирају формулом датом у једначини 4.4.4.1, где је $f'(y_i)$ извод логистичке функције, x је улаз, y је стварни излаз, а o је добијени излаз. За више улаза користе се линеарне оцене другог реда, као што је дато једначином 4.4.4.2, где је $\bar{y}$ средња вредност свих излаза, а Q_y је аутокорелација. [97]

$$\Delta w_i = f'(y_i)x(y_i - o_i) \tag{4.4.4.1}$$

$$o = y + (Q_y - 2I)(y - \bar{y}) \tag{4.4.4.2}$$

Непараметарска ентропија има диференцијално правило за учење које оптимизује ентропију одређивањем густине кернела, тако да није потребно вршити додатне претпоставке о глаткости густине или функционалне форме. Одређивање густине непараметарског Парзеновог прозора (*Parzen window*) може се одредити формулом 4.4.4.3, где је T узорак тачака излаза, K_σ је кернелска функција, у овом случају Гаусова расподела са варијансом σ^2. Ширина кернела која најбоље одређује густину може се оценити максимизацијом логаритамске вероватноће дате једначином 4.4.4.4. Њени изводи представљени су једначином 4.4.4.5.

$$p(y) = \frac{1}{|T|}\sum_{y_j \in T} K_\sigma\,(y - y_j) \tag{4.4.4.3}$$

$$L = \sum_{y_i \in S} log \sum_{y_j \in T} K_\sigma(y_i - y_j) - |S|\log|T| \tag{4.4.4.4}$$

$$\frac{\partial L}{\partial \sigma} = \sum_{y_i} \frac{\sum_{y_j \in T} \frac{\partial}{\partial \sigma} K_\sigma(y_i - y_j)}{\sum_{y_j \in T} K_\sigma(y_i - y_j)} \tag{4.4.4.5}$$

На основу претходних једначина непараметарска оцена емпиријске ентропије (под условом да је дат оптимални облик кернела) неуралне мреже са излазом може се израчунати као у једначини 4.4.4.6 и минимизовати у односу на тежине у неуралној мрежи по формули 4.4.4.7. [97]

$$H(y) = -\frac{1}{|S|} \sum_{y_i \in S} log \sum_{y_j \in T} K_\sigma(y_i - y_j) + \log |T| \tag{4.4.4.6}$$

$$\frac{\partial}{\partial w} H(y) = -\frac{1}{|S|} \sum_{y_i \in S} \frac{\sum_{y_j \in T} \frac{\partial}{\partial w} K_\sigma(y_i - y_j)}{\sum_{y_j \in T} K_\sigma(y_i - y_j)} \tag{4.4.4.7}$$

Мала ентропија се постиже добром кластеризацијом података. [97] Оваква дефиниција може се користити за сигнале, као што је на пример говорни сигнал, међутим, за коришћење над текстом потребне су измене методе.

4.5. Кластеризација помоћу конволуционих неуралних мрежа

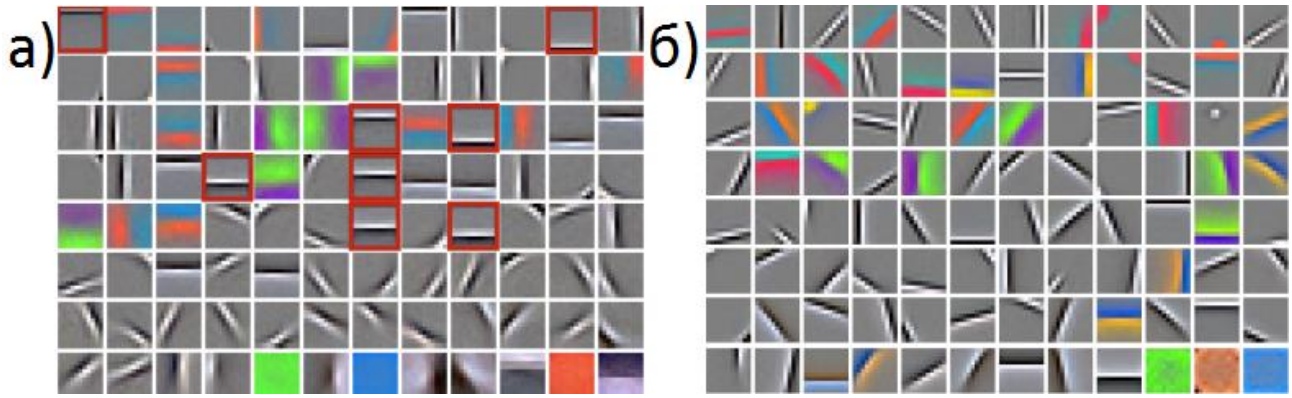
Поред рекурентних и обичних неуралних мрежа, могуће је користити конволуционе неуралне мреже за решавање ненадгледаних проблема, укључујући и кластеризацију. У раду [98] се креира конволуциона мрежа за одређивање *k* средњих вредности и уједно се оптимизује број корелисаних параметара коришћењем филтера чиме се повећава тачност алгоритма.

Први корак алгоритма је да се науче филтри користећи *k* средњих вредности. Идеја је да се учење филтера уради на принципу алгоритма *k* средњих вредности, тако да филтери представљају центроиде. У контексту [98], улази $w^{(i)}$ су случајно добијени парчићи слике и центроиди су филтери који се користе да се кодирају слике. На овај начин се учи речник $D \in R^{nxk}$ из улазних вектора $w^{(i)} \in R^n$ за *i=1,2,…,m*. У алгоритму се речник може представити формулама 4.5.1 до 4.5.3, где је $s^{(i)} \in R^k$ кодирани вектор везан за улаз $w^{(i)}$ и $D^{(j)}$ је *j*-та колона речника. Матрице $W \in R^{nxm}$ и $S \in R^{nxm}$ имају колоне $w^{(i)}$ и $s^{(i)}$, редом. [98]

$$s_j^{(i)} = \begin{cases} D^{(j)^T} w^{(i)}, \text{ако је } j = arg\max_l |D^{(l)^T} w^{(i)}| \\ 0, \text{иначе} \end{cases} \tag{4.5.1}$$

$$D = WS^T + D \tag{4.5.2}$$

$$D^{(j)} = \frac{D^{(J)}}{||D^{(j)}||_2} \tag{4.5.3}$$



Слика. 4.5.1. а) Филтери издвојени методом к-средњих вредности; б) Филтери издвојени конволуцијом средњих вредности. [98]

Описана шема учи центроиде сваког кластера на нивоу парчића, а с друге стране филтери се користе за конволуцију код конволуционих неуралних мрежа. Као што се може видети на слици 4.5.1. а), многи центроиди имају исту орјентацију и представљају померену верзију другог центорида у простору. Оваква особина није добра за конволуцију јер ће произвести исте излазе, тј. откриће исте особине слике. Овај проблем се може решити променом начина на који се узимају узорци (парчићи) слике. Уместо парчића исте величине, овај алгоритам користи велике прозоре као улазе на основу којих се одлучује које парчиће из тих прозора треба издвојити за кластеризацију. [98]

Прозори се бирају тако да су два пута већи од величине филтера и случајно су изабрани у односу на улазну слику. Центроиди алгоритма *k* средњих вредности конволуирају цео прозор како би израчунали матрицу сличности сваке локације. Парче које даје највећу активацију из прозора је најсличиније особинама центроида, зато се оно користи за одговарајући центроид. Одговарајуће учење се може обавити корацима 4.5.4 до 4.5.6, где су x и y информације о горњем левом углу улазног парчета. [98]

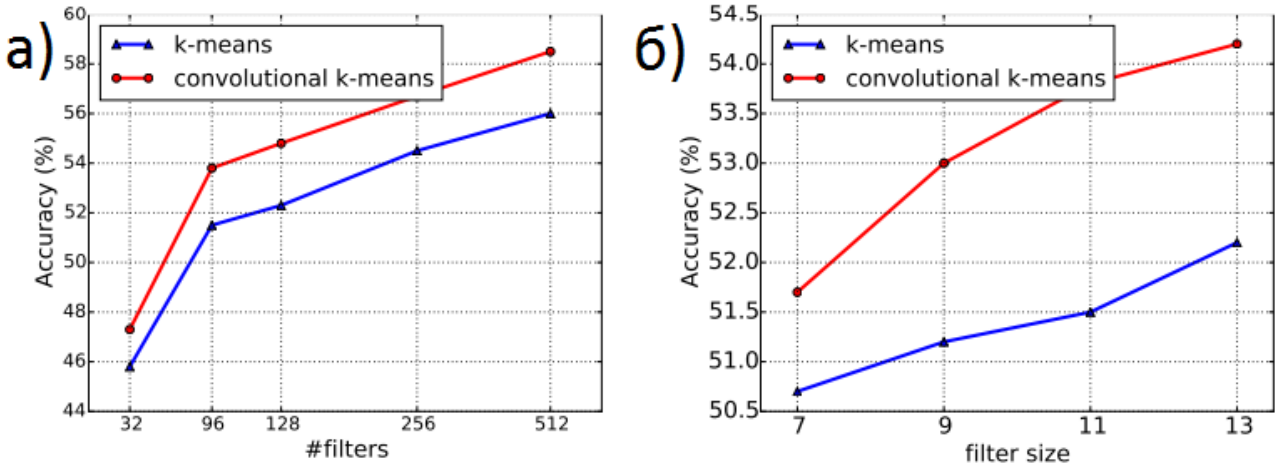
$$s_j^{(i)} = \begin{cases} D^{(j)^T} w_{(x,y)}^{(i)}, \text{ако је} (j,x,y) = \arg\max_{(l,m,n)} |D^{(l)^T} w_{(m,n)}^{(i)}| \\ 0, \text{ иначе} \end{cases} \tag{4.5.4}$$

$$D = W_{(x,y)} S^T + D \tag{4.5.5}$$

$$D^{(j)} = \frac{D^{(j)}}{||D^{(j)}||_2} \tag{4.5.6}$$

Када су корелисани филтери уклоњени, има места за додатне филтере који могу да науче нове информације (слика 4.5.1. б)). Овако добијени алгоритам има бољу тачност у односу на почетни без обзира на величину и број филтера (слика 4.5.2). Међутим, што је број и величина филтера већи то је тачност боља за оба алгоритма, али је и време тренирања дуже.

Овај алгоритам се може модификовати да ради са текстом, при чему ће се уместо дводимензионе слике користити једнодимензиони сигнал текста и филтер ће учити карактеристике текста. На крају, треба рећи да се овај слој даље комбинује са другим слојевима који могу бити ненадгледани или надгледани, а резултати дати у [98] тестирањем различитих комбинација показују да је тачност повећана за 5% коришћењем два ненадгледана слоја.



Слика 4.5.2. а) Тачност алгоритама у зависности од броја филтера; б) Тачност алгоритама у зависности од величине филтера [98]

Док је у [98] представљено коришћење конволуционих мрежа и ненадгледаног учења за уобичајне задатке конволуционе мреже (тј. обраду слике), у [99] се конволуционе мреже користе за класификацију релација. Као улаз у ове мреже узимају се све речи из текста, које се трансформишу у векторе. Затим се издвајају лексичке одлике, док се одлике реченица уче конволуцијом. На крају се ова два нивоа одлика спајају како би се добио коначни изглед вектора одлика који се убацује у софтмакс класификатор на основу ког се одређује веза између две именице. У овом раду се конволуционе неуралне мреже користе за надгледано учење, али је аутор овог мастер рада одлучио да га укратко представи овде због тога што се класификација лако може променити у кластеризацију изменом алгоритма тако да оптимизује функцију грешке кластеризације.

У раду [63] који је укратко описан у одељку 3.5.3. конволуционе неуралне мреже се користе за кластеризацију кратких текстова. Алгоритам такође користи спој методе *k* средњих вредности и конволуционих мрежа као у раду [98].

5. КЛАСТЕРИЗАЦИЈА НАУЧНИХ РАДОВА

Као што је речено у уводу циљ овог рада је да се кластеризују научни радови из различитих области. Аутор намерно користи само методе машинског учења које могу да формирају резултате без лабелирања јер сматра да је за широку примену решења немогуће лабелирати толико велике количине текста. Како би се направио такав систем, потребно је креирати језички модел, а затим груписати документа у зависности од њихове сличности у језичком моделу. Због тога су у поглављу 3. приказани алгоритми за креирање језичког модела, а у поглављу 4. алгоритми за кластерисање. У поглављу пет ће они бити искомбиновани уз одређене доприносе аутора ради добијања коначног резултата и упоређивања перформанси.

5.1. Подаци

Подаци су прикупљени са сајта *arxiv.org* на ком се објављују научни радови који могу бити прочитани бесплатно и одмах по завршетку писања за разлику од конференција и часописа код којих процес публиковања траје месецима и приступ радовима се често наплаћује. Радови су на овом сајту груписани у шест главних категорија: физика, математика, рачунарске науке, квантитативна биологија, квантитативне финансије и статистика. Свака од ових категорија садржи једну или више поткатегорија, а свака од поткатегорија садржи теме. Свака категорија, поткатегорија и тема има свој код.

Овај вебсајт има свој *API (Application programming interface)* који се може користити за добијање информација. Због ауторских права, не може се добити цео текст рада, већ само наслов, апстракт и додатне информације, тако да се о сваком раду чува:

- Наслов рада (пример: "*A nonextensive approach to Bose-Einstein condensation of trapped interacting boson gas*");
- Идентификација рада на *arxiv.org* (пример: *0802.1248*);
- Апстракт рада (неколико параграфа који описују суштину рада, углавном дужине до пола стране);
- Поткатегорија и тема одвојене тачком (пример: *cond-mat.stat-mech* (поткатегорија кондензована материја, а тема статистичка механика));
- *doi* линк (јединствена идентификација рада, пример: 10.1007/s10909-007-9596-2);
- Датум креирања (у формату „YYYY-MM-DD", пример 2008-02-24);
- Датум ажурирања (у формату „YYYY-MM-DD", пример 2008-02-24, постоји само уколико је рад ажуриран, тј. објављена је нова верзија).
- Листа аутора (садржи име па презиме аутора, име је понекад дато само првим словом; може бити један или више аутора)

Сајт *arxiv.org* прикупља радове од 2007 године до данас, тако да су преузети радови креирани од 1. августа 2007. закључно са 31. јулом 2017. године. Треба напоменути да је датум креирања заправо датум постављања рада на arxiv.org по први пут. Уколико се ради о старим радовима (који су настали пре arxiv.org), онда се може десити да су они додати на сајт и

деценијама касније, те ће њихово време креирања бити свакако између 2007. и 2017. године, а њихов стварни датум објављивања и настанка се не може добити из података који се прикупљају преко *API*-ја.

Аутор овог мастер рада је проширио библиотеку [100] како би добио информације о свим доступним радовима на вебсајту за горе наведени период. За екстракцију је коришћен период креирања документа на сајту и поткатегорије. Прикупљено је укупно 4.2 GB података, што су информације за 3.799.944 радова. За прикупљање података коришћен је *python 3.4* и библиотеке csv, *datetime*, *time*, *urllib* и *xml*.

5.2. Техничка спецификација

Сви алгоритми имплементирани су на програмском језику *python* најчешће користећи и проширујући библиотеке које већ постоје јер су многе од њих оптимизоване за рад са одговарајућим алгоритмима. Одговарајуће библиотеке ће бити напоменуте и детаљније објашњене у наредним потпоглављима.

Сви програми су извршени на *Windows 10* оперативном систему, са *64GB RAM* меморије и процесором *i7-6700* брзине *3.4GHz* користећи *2.7* или *3.4* 64-битну верзију програмског језика *python*. Алгоритми су извршавани често паралелно при чему је коришћено и до 100% процесора и 99% меморије што је утицало на време извршавања појединачних програма, тако да нису мерена времена извршавања алгоритама.

5.3. Претпроцесирање

У поглављу два описане су многе технике претпроцесирања текста и речено је да се са новијим алгоритмима веома мали број њих заправо користи. Пошто су у овом раду коришћени алгоритми са најбољим резултатима, а то су уједно и најновији алгоритми, претпроцесирање текста је минимално и било је исто за све алгоритме. Коришћени су следећи кораци:

- Чишћење од интерпункције, при чему је знак за нови ред замењен празнином, а знаци тачка, зарез, двотачка, отворена и затворена заграда су избачени. Други знакови интерпункције као што су наводници, упитници, узвичници, цртице, апострофи нису избачени јер се сматра да доприносе контексту. Ово су текстови из научних радова и не очекују се такви знаци осим ако нису битни за садржај.
- Коришћење само малих слова: У научним радовима се велика слова користе за скраћенице и за почетна слова у реченицама. За почетна слова у реченицама је битно да се промене у мала, а скраћенице написане малим словима не губе на значају, тако да је аутор одлучио да сва слова претвори у мала.
- Токенизација: аутор је поделио сав текст у токене, који су издвојени на основу празнина између њих. Треба приметити да ће речи повезане цртицом означавати једну реч, што је било логично за аутора овог рада јер такве речи имају смисао само ако се користе заједно, а ако се користе одвојено могу имати скроз другачији смисао.

Одређивање основе речи није коришћено јер се сматра да ће алгоритми бити у стању да речима са сличном основом доделе сличне векторе, тј. да схвате семантичку блискост између речи са истом основом. Избацивање честих речи није коришћено јер се у алгоритмима користи константа којом се честе речи узоркују мање од ређих речи.

5.4. Креирање језичког модела

За креирање језичког модела аутор је изабрао три алгоритма која ће упоређивати у овом поглављу. Сва три алгоритма су настала у претходних пар година и дају боље резултате од њихових претходника за краће време. У овом раду на основу вектора добијених помоћу њих су креирани кластери радова и коначни резултати су упоређивани. Како би се упоредили само алгоритми за креирање језичких модела, испитивано је колико добро препознају релације и семантичке сличности између речи. Коришћени су алгоритми: *word2vec* (потпоглаље 3.7), *gloVe* (потпоглавље 3.8) и *doc2vec* (потпоглавље 4.4).

За алгоритме *word2vec* и *doc2vec* коришћена је имплементација дата у библиотеци *Gensim* [101], а за алгоритам *gloVe* коришћена је библиотека [102]. Ово су библиотеке за језик *python* које су имплементиране на сличан начин и оптимизоване су тако што су рачунања имплементирана у језику *C++* и искомпајлирана тако да се могу користити у језику *python*. На овај начин убрзан је рад алгоритма. Међутим, аутор је имао значајне проблеме да одговрајуће библиотеке подеси и користи из више разлога.

Због ограничења које има језик python у својој 32-битној верзији, он може да користи само *2 GB RAM* меморије без обзира на могућности компјутера, те је аутор био принуђен да за све алгоритме креирања језичког модела користи 64-битну верзију *python 2.7*. Такође, пошто је део кода био написан на језику C++ био је потребан *Visual C++ Build Tools* алат за компајлирање библиотека тако да раде са 64-битним улазима. Као основу за коришћене библиотеке, било је потребно инсталирати *numpy, scipy* и *cython* библиотеке.

Нажалост, многе библиотеке постоје само у 32-битној верзији, те треба водити рачуна која се библиотека користи и треба бити спреман на самосатално компајлирање библиотеке. Такође, треба водити рачуна о томе да ли је библиотека доступна за *python 2* или за *python 3*. Ове ствари најчешће нису специфициране, већ се могу открити тек кад се појави грешка приликом инсталирања библиотеке или покретања програма.

Сви алгоритми су користили исте параметре: 4 нити, величина контекста 10 (тј. удаљеност од централне речи максимално 5 за *word2vec* алгоритме), свака реч се представља вектором са 100 димензија, коефицијент учења је 0.025 и број епоха кроз које алгоритам пролази је 5. Код свих алгоритама се креира модел на основу постављених параметара и улаза. Сви модели су тренирани са три различита улаза: 1) само наслови радова, 2) само апстракти, 3) наслови радова и апстракти. Уколико се нека реч појављује мање од два пута избачена је из корпуса.

Код *word2vec* алгоритма, улази су речи, а трениране су четири верзије алгоритма:

- Изостављање н-грама са хијерархијском софтмакс функцијом
- Изостављање н-грама са негативним узорковањем
- Континуална врећа речи са хијерархијском софтмакс функцијом
- Континуална врећа речи са негативним узорковањем

Код *GloVe* алгоритма улази су такође речи, а коначно добијени вектори су прерађени тако да имају исту струкутуру као вектори добијени претходним алгоритмом. Алгоритам се састоји из креирања корпуса, речника и матрице заједничког понављања, а затим тренирања модела на основу дате матрице и параметара.

За оба претходна алгоритма, по добијању вектора речи израчунати су вектори наслова, апстракта и оба (наслова и апстракта заједно) као сума вектора речи који их чине.

doc2vec алгоритам се базира на *word2vec* алгоритму, осим што су улази параграфи и излази су вектори параграфа, а не вектори речи. У овом случају параграфи су наслови, тј. апстракти или оба. Коришћене су две верзије алгоритма:

- *PV-DM* (описан у одељку 4.4.1)
- *PV-DBOW* (описан у одељку 4.4.2)

Алгоритми су поређени по релацијама које успевају да открију и простору вектора које одређују. Детаље погледати у потпоглављу 5.6.

5.5. Одређивање сличности између радова

Аутор је креирао код за одређивање сличности између радова на основу само наслова, само апстракта или оба заједно, а користећи векторе добијене за њих у претходном кораку. Над таквим скупом вектора аутор је имплементирао два алгоритма кластеризације: 1) алгоритам *k* средњих вредности и 2) спектрална кластеризација. Аутор се одлучио за ова два алгоритма зато што су други представљени алгоритми временски захтевнији, ослањају се на ове алгоритме, имају сличне резултате или једноставно нису пригодни за коришћење у овом случају.

Оба алгоритма се могу наћи у библиотеци *scikit-learn* [88] чија је имплементација и овде коришћена, а која се ослања на библиотеке *numpy*, *scipy* и *setuptools*. Пошто је укупан број поткатегорија на сајту *arxiv.org* 18, а број тема је 148, ово је број кластера који је дефинисан код оба алгоритма, тј. за сваки од језичких модела креирани урађена је кластеризација у 18 и у 148 група.

Код алгоритма *k*-средњих вредности центри почетних кластера се иницијализују на паметан начин тако да се убрза конвергирање, а због велике вероватноће да алгоритам заврши у локалном минимуму, алгоритам се покреће 10 пута и бира се најбољи резултат. Максимални број итерација (после којих се алгоритам зауставља чак иако не конвергира) је 300, а граница грешке испод које се извршавање зауставља је 10^{-4}. Овај алгоритам користи 4 паралелне нити.

Код спектралног кластеровања се на крају за одређивање група на основу вектора сопствених вредности, користи алгоритам *k*-средњих вредности чији су параметри исти као код горе наведеног алгоритма. За рачунање сличности између елемената користи се *rbf* кернел.

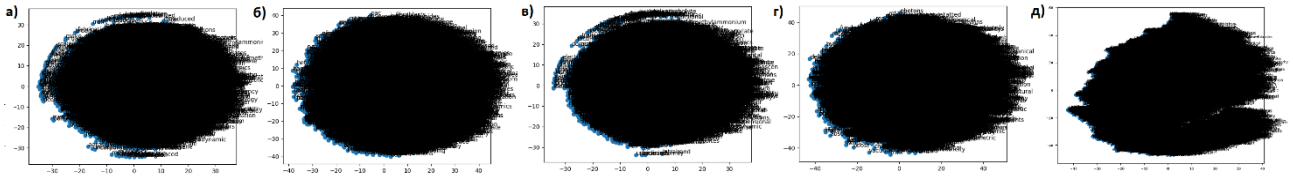
Након кластеризације, за одређивање поклапања између стварних поткатегорија и тема и добијених кластера, коришћен је нормализовани заједнички скор, описан у одељку 4.1.2. Резултати су дати у потпоглављу 5.6.

5.6. Резултати

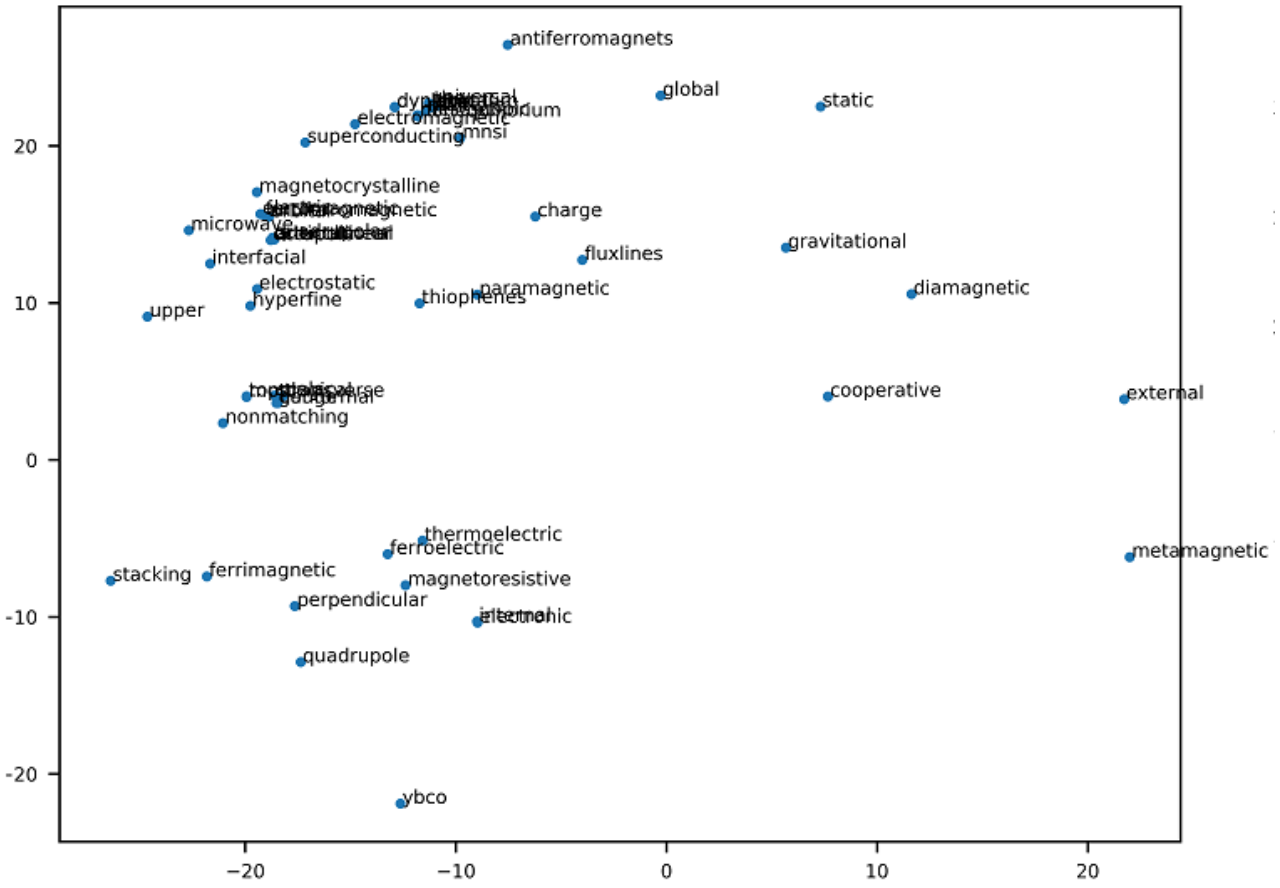
Улазни подаци садрже информације о 3.799.944 радова, при чему се за добијање резултата користе наслови радова, апстракти радова, комбинација ова два текста и лабеле којима је рад означен на *arxiv.org*. Над њима су прво извршени алгоритми за векторизацију речи (*word2vec, glove*) и параграфа (*doc2vec*) који су за сваку реч одредили вектор у 100 димензија. Језички модел који је креиран на овај начин оцењује се на основу тога колико има смисла за човека и колико се разумевање језика човека поклапа са разумевањем модела. На слици 5.6.1. приказани су генерисани речници у дводимензионом простору. Да би се добила

ова слика, вектори су из 100 димензија пребачени у две димензије, тако да се изгуби што мања количина информација. То је урађено помоћу репрезентације *t* дистрибуираних случајних комшија (*TSNE – t-distributed stochastic neighbor embedding*), а затим су дводимензиони вектори приказани на слици заједно са речима које представљају. Речници су приказани само за моделе добијене од наслова радова, јер аутор није могао да креира остале слике на свом рачунару због временске и просторне комплексности *TSNE* алгоритма и величине података.

Речници настали од наслова радова имају 76899 речи, речници од апстракта имају 393764 речи, а речници креирани од наслова и апстракта имају 409210 речи. Овде треба навести да се под речју сматра токен добијен после претпроцесирања које је описано у потпоглављу 5.3. Многи токени су формуле или сложенице настале од више речи. Пошто су радови писани у LaTex програму, формуле су специфициране одговарајућом синтаксом.



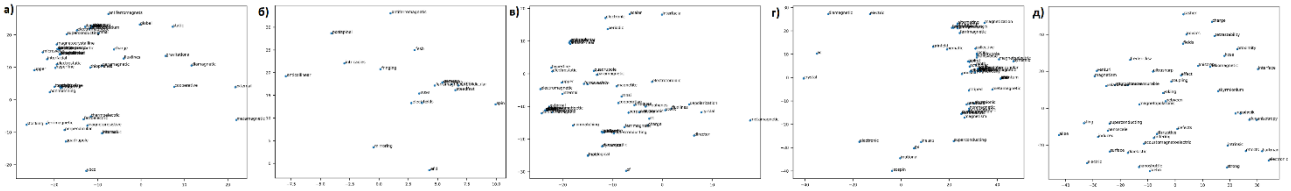
Слика 5.6.1. Речници представљени у 2D простору креирани: а) *word2vec*, изостављање н-грама, хијерархијски софтмакс; б) *word2vec*, изостављање н-грама, негативно узорковање; в) *word2vec*, континуална вреће речи, хијерархијски софтмакс; г) *word2vec*, континуална вреће речи, негативно узорковање; д) *GloVe*



Слика 5.6.2. Најсличнијих 100 речи за реч *magnetic* на основу језичког модела креираног *word2vec* алгоритмом са изостављањем н-грама и хијерархијским софтмаксом

Нажалост, због великог броја речи у речнику на сликама 5.6.1а) до 5.6.1.д), није могуће одредити које се речи налазе где, колико су блиске и какав је језички модел генерално. Међутим, могуће је видети да се опсег вредности разликује за различите алгоритме. За алгоритме са хијерархијским софтмаксом опсег вредности вектора је између -30 и 30, за алгоритме са негативним узорковањем између -40 и 40, а за *GloVe* између -50 и 60.

Како би се оценила могућност алгоритама да разумеју сличности између речи аутор је изабрао неколико речи и одредио 2D приказ за 100 речи (из којих су избачене формуле) које су најсличније тим речима. На слици 5.6.2. приказане су речи најсличније речи *magnetic* на основу језичког модела креирног алгоритмом са изостављањем н-грама и хијерархијским софтмаксом. На слици 5.6.3. приказане су слике за речи најсличније речи *magnetic* за све алгоритме. Може се приметити да простори речи изгледају веома различито међу собом. Ако су две речи сличне (близу) под једним моделом, не значи да ће оне бити једнако близу код других модела. Такође, 100 најсличнијих речи за реч *magnetic* нису исте за ове алгоритме, мада се скупови преклапају.

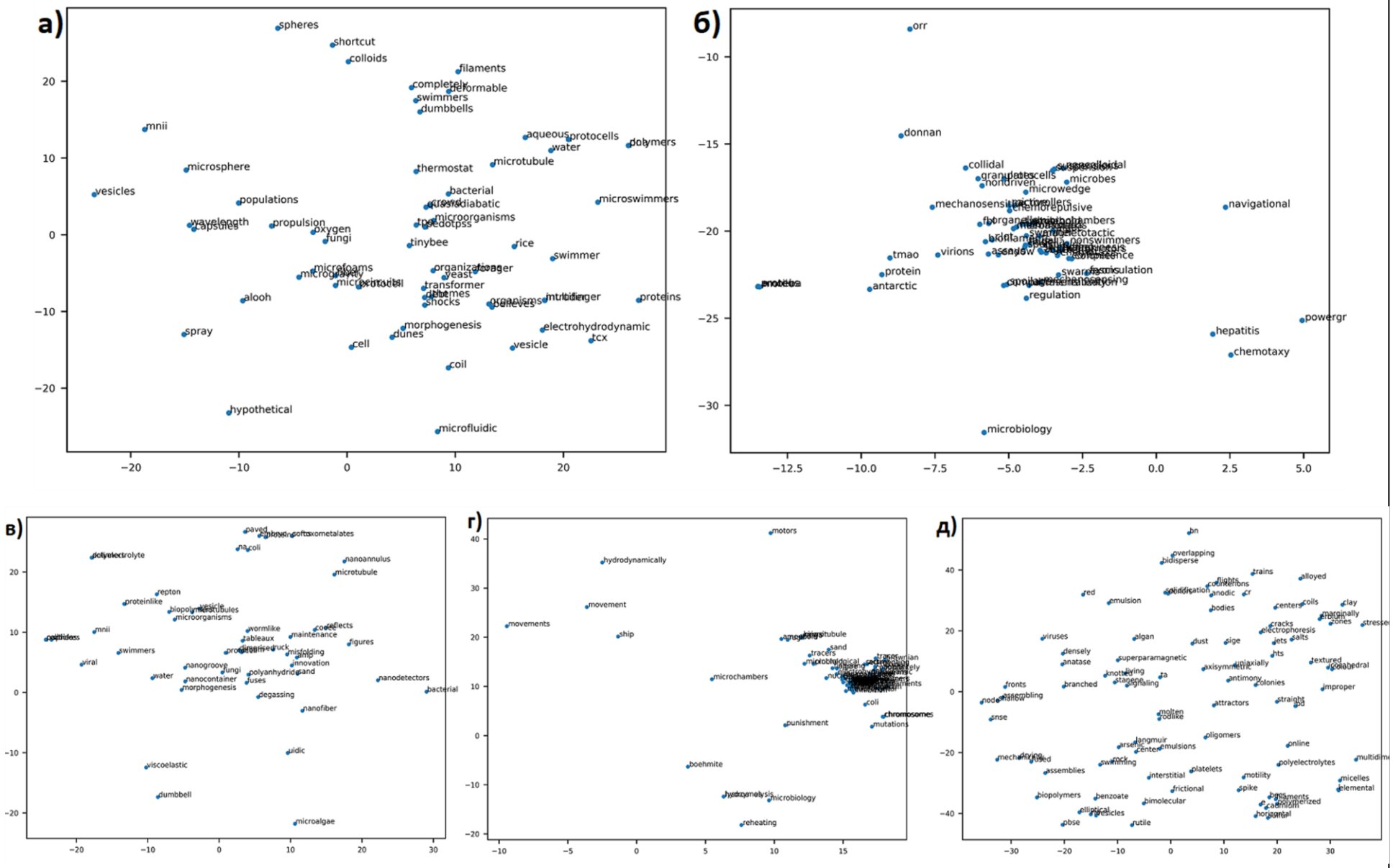


Слика 5.6.3. Најсличнијих 100 речи за реч magnetic на основу језичких модела креираних: а) *word2vec*, изостављање н-грама, хијерархијски софтмакс; б) *word2vec*, изостављање н-грама, негативно узорковање; в) *word2vec*, континуална врећа речи, хијерархијски софтмакс; г) *word2vec*, континуална врећа речи, негативно узорковање; д) *GloVe*

На слици 5.6.4. приказане су најсличније речи за реч *bacteria*, на којима се могу уочити сличне карактеристике као и на слици 5.6.3. Речи које су изабране од стране *word2vec* варијанти алгоритма су међу собом сличне и постоје видљива преклапања, а понекад су чак исте речи приказане на истом месту у простору или се две речи односе на исти начин једна према другој у два различита језичка модела. Међутим, речи које су изабране од стране *GloVe* алгоритма деле врло мали подскуп речи са *word2vec* алгоритмима.

Ни на једној од наведених слика није јасно да су неке речи близу јер су веома сличне као што је дато у описима алгоритама. Ово се може објаснити чињеницом да се овде ради са много мањом количином података и чињеницом да су речи изабране да буду сличне одређеној речи која није приказана на слици. Такође, може се закључити да су речи приказане на сликама у горе описаним радовима пажљиво биране на основу анализе сличности између њих тако да се произведу слике које показују добре резултате. Због тога ће се овај текст у наставку бавити такође проналажењем сличних речи и разумевањем перформанси алгоритма за одређивање сличности.

Према мишљењу аутора, речи које су изабране као најсличније за речи *magnetic* и *bacteria* су најбоље изабране *GloVe* алгоритмом, што ће показују и коефицијенти сличности дати на слици 5.6.5. На оснвоу тих коефицијената и речи које се приказују као сличне може се закључити да *GloVe* алгоритам може дати врло добре резултате и за мању количину података. С друге стране, може се такође применити да је овај алгоритам прилично нестабилан ако се упоређују избори дати за модел креиран над апстрактима и за модел креиран над апстрактима и насловима. Ови скупови су често веома различити за *GloVe*, док word2vec алгоритми дају приближно сличне резултате у овом случају.



Слика 5.6.4. Најсличнијих 100 речи за реч *bacteria* на основу језичких модела креираних: а) *word2vec*, изостављање н-грама, хијерархијски софтмакс; б) *word2vec*, изостављање н-грама, негативно узорковање; в) *word2vec*, континуална врећа речи, хијерархијски софтмакс; г) *word2vec*, континуална врећа речи, негативно узорковање;

		Наслов	Апстракт	Оба
particles	SgHs	particle, 0.771; colloids: 0.727; electrons: 0.639; fermions: 0.627; rods:0.600; bosons: 0.600	beads: 0.785; colloids: 0.777; swimmers: 0.756; rods: 0.749; bodies: 0.748; monomers: 0.747	colloids: 0.778; objects: 9.740; particle: 0.739; swimmers: 0.737; bodies: 0.723; monomers: 0.718
	SgNs	self-propelled:0.752; particle: 0.726; quasi-two-dimensions:0.721; yolk-shell: 0.710; macromolecules/ interacting: 0.700; amphiphatic: 0.699	particle: 0.835; self-propelled: 0.742; colloids: 0.718; shperes: 0.716; colloidal: 0.704; gold-capped: 0.701	particle: 0.830; self-propelled: 0.740; colloids: 0.733; shperes: 0.723; janus: 0.714; swimmers: 0.712
	CbHs	particle: 0.770; colloids: 0.735; electrons: 0.637; spheres: 0.631; rods: 0.623; fermions: 0.622	beads: 0.774; colloids: 0.767; swimmers: 0.745; grains: 0.745; bodies: 0.743; objects: 0.741	colloids: 0.771; swimmers: 0.746; objects: 0.740; particle: 0.738; bodies: 0.732; monomers: 0.719
	CbNs	particle: 0.736; colloids: 0.648; spheres: 0.635; objects: 0.590; rods: 0.574; dumbbells: 0.563	particle: 0.753; spheres: 0.731; colloids: 0.724; balls: 0.702; monomers: 0.684; beads: 0.680	particle: 0.760; spheres: 0.716; colloids: 0.696; dumbbells: 0.679; beads: 0.679; monomers: 0.668
	GloVe	atoms: 0.980; confined: 0.977; media: 0.969; condensate: 0.968; bose-einstain: 0.968; periodically: 0.964	species: 0.950; $n_{s{\\\\cal: 0.940; bicone-shaped: 0.931; n^30: 0.924; pre-coated: 0.923; particle: 0.921	$n_{s{\\\\cal: 0.951; n^30: 0.951; nonpointlike: 0.925; arbitrary-spin: 0.924; pre-coated: 0.923; supermobile: 0.922
collide	SgHs	+b: 0.472; kronecker's: 0.465; moderately-doped: 0.452; fe_{106}te: 0.430; antiholons: 0.427; sr$_2$fe{/it: 0.424	annihilate: 0.619; move: 0.619; join: 0.583; colliding: 0.568; interact: 0.566; touch: 0.551;	interact: 0.645; move: 0.609; react: 0.585; annihilate: 0.568; repel: 0.563; interacts: 0.539
	SgNs	never: 0.650; o'connell's: 0.649; tumble?: 0.637; kronecker's: 0.634; tied-down: 0.623; o'connell: 0.623	reconnect: 0.686; repel: 0.665; $1_{1}$and: 0.645; colliding: 0.634; $\\\\floor{\\\\frac{n}{4}}\\\\rfloor$: 0.623; annihilate: 0.616	hop: 0.646; annihilate: 0.630; push:0.626; colliding: 0.619; repel: 0.615; $1_{1}$and: 0.610
	CbHs	kronecker's: 0.526; +b: 0.460; sr1-xcax3ru207: 0.456; pseudo-half-metalicity: 0.448; mexican-hat-type: 0.444; flagella-driven: 0.443	move: 0.633; annihilate: 0.611; interact: 0.604; coalesce: 0.580; join: 0.576; colliding: 0.559;	interact: 0.661; move: 0.623; rotate: 0.599; annihilate: 0.574; repel: 0.570; accumulate: 0.559
	CbNs	never: 0.577; auction: 0.561; compressive: 0.554; merges: 0.498; infere: 0.490; entropy-reducing: 0.486	interact: 0.662; join: 0.652; colliding: 0.649; repel: 0.646; annihilate: 0.643; coalesce: 0.607	annihilate: 0.668; join: 0.665; repel: 0.651; interact: 0.645; colliding: 0.629; move: 0.628
	GloVe	steered: 0.946; mimic: 0.946; double-sigmoid: 0.944; infinite-range: 0.942; magnon-polaritons: 0.941; inter-vortex: 0.939	preferentially: 0.971; annihilate: 0.968; coalesce: 0.965; prefer: 0.959; slide: 0.957; repel: 0.956	coalesce: 0.963; preferentially: 0.960; annihilate: 0.956; slide: 0.954; stay: 0.950; adsorb: 0.947
magnetic	SgHs	ferromagnetic: 0.653; antiferromagnetic: 0.609; electric: 0.598; local: 0.515; topological: 0.502; transverse: 0.49	electric: 0.779; coercive: 0.698; high-magnetic: 0.686; oersted: 0.676; pseudo-magnetic: 0.660; stray: 0.627	electric: 0.772; coercive: 0.691; high-magnetic: 0.690; oersted: 0.632; stray: 0.618; electromagnetic: 0.608
	SgNs	field: 0.643; co$_2$fe$_{04}$mn$_{6}$si: 0.604; cobalt-graphene: 0.604; anisotrophy: 0.600; fe_{025}tas_2:0.599; piezo-control: 0.598	magnetization: 0.691; field: 0.685; fields: 0.677; electric: 0.670; electrotarnsport: 0.663; site-doping: 0.651	magnetization: 0.722; fields: 0.693; field: 0.688; high-topological-number: 0.680; electric: 0.676; anisotropy: 0.674
	CbHs	ferromagnetic: 0.660; antiferromagnetic: 0.603; electric: 0.596; local: 0.540; orbital: 0.506; topological: 0.492	electric: 0.786; coercive: 0.689; oersted:0.689; high-magnetic: 0.679; pseudo-magnetic: 0.643; stray:0.623	electric: 0.774; high-magnetic: 0.699; coercive: 0.691; oersted: 0.650; overhauser: 0.618; stray: 0.617
	CbNs	ferromagnetic: 0.658; antiferromagnetic: 0.615; electric: 0.552; spin: 0.517; cycloidal: 0.467; of: 0.460	electric: 0.697; ferromagnetic: 0.637; high-magnetic: 0.622; z-magnetic: 0.597; ferrimagnetic: 0.577; coercive: 0.574	electric: 0.678; ferromagnetic: 0.644; high-magnetic: 0.641; coercive: 0.580; antiferromagnetic: 0.580; ferrimagnetic: 0.562
	GloVe	effect: 0.963; hartman: 0.962; anisotropy: 0.962; ettingshause: 0.952; pt: 0.948; hose: 0.946	motion-inducing: 0.953; 7-t: 0.953; in-planed: 0.952; transverse-conical: 0.951; $1/31/30$: 0.949; $n_\\\\|=1$: 0.948	motion-inducing: 0.954; synthetic: 0.949; 7-t: 0.949; inverse-radial: 0.945; soft-axis: 0.944; polarity-tunable: 0.944
statistics	SgHs	statistics"; 0.466; thermostatistics: 0.459; distributions: 0.430; resources: 0.412; process: 0.412; densities: 0.411	distributions: 0.594; trajectories: 0.565; ensembles: 0.552; statistics-the: 0.506; indistinguishability: 0.496; noninteracting: 0.494	distributions: 0.601; statistic: 0.553; ensembles: 0.541; distribution: 0.523; statistics-the: 0.522; trajectories: 0.521
	SgNs	counting: 0.696; rank-ordering: 0.655; six-qubit: 0.654; single--molecule: 0.652; non--extensive: 0.649; legitimate: 0.646	counting: 0.686; distributions: 0.659; distribution; 0.643; statistical: 0.624; poissonian: 0.617; extreme-value: 0.611	counting: 0.699; distributions: 0.675; distribution: 0.667; statistical: 0.626; wigner-dyson: 0.622; randomly-generated: 0.621
	CbHs	distributions: 0.463; thermostatistics: 0.433; yoctocalorimetry: 0.423; extractable: 0.407; variance: 0.404; nonmarkovian: 0.402	distributions: 0.572; trajectories: 0.553; ensembles: 0.534; statistics-the: 0.488; indistinguishability: 0.487; weights: 0.462	distributions: 0.602; statistic: 0.582; distribution: 0.534; pdfs: 0.528; trajectories: 0.519; ensembles: 0.510
	CbNs	distributions: 0.501; trajectories: 0.494; thermostatics: 0.465; statisticsoperator: 0.463; integrability: 0.458; correlators: 0.454	distributions: 0.537; log-poisson: 0.534; wilczeck's: 0.529; indistinguishability: 0.522; full-counting: 0.516; poissonian: 0.513	distributions: 0.573; statistic: 0.551; wilczeck's: 0.504; indistinguishability: 0.500; danube: 0.495; bhp: 0.492;
	GloVe	one-parameter: 0.971; stationary: 0.971; gumbel: 0.971; steady-state: 0.969; work: 0.968; weibull: 0.968	boltzmann-planck: 0.946; quons: 0.941; tw-gue: 0.940; switching-time: 0.940; n^{-2}: 0.937	passage-time: 0.936; bose-einstain: 0.933; 2d-kpz: 0.930; fractionary: 0.926; --wavefunction: 0.926; ifds: 0.9253
bacteria	SgHs	microorganisms: 0.478; vesicle: 0.439; nucleation-and-growth: 0.424; morphogenesis: 0.424; quasiadiabetic: 0.417; microorganisms": 0.412	microorganisms: 0.720; swimmers: 0.690; flagella: 0.670; bacterial: 0.668; macromolecules: 0.663; microswimmers: 0.643	microorganisms: 0.814; swimmers: 0.714; bacterial: 0.707; organisms: 0.702; microswimmers: 0.666; proteins: 0.654
	SgNs	twitching: 0.675; microchambers; 0.671; superfluid: 0.645; motile: 0.636; non-motile: 0.619; spoilage; 0.618	motile: 0.852; bacterial: 0.852; microorganisms: 0.830; motility: 0.806; microswimmers: 0.805; organisms: 0.800;	motile: 0.848; bacterial: 0.843; organisms: 0.824; microorganisms: 0.821; swimming: 0.819; motility: 0.817
	CbHs	eyring-fluids: 0.471; transversely-corrugated: 0.449; dna: 0.440; water: 0.440; nanogroove: 0.437; microtubule: 0.435	microorganisms: 0.748; swimmers: 0.673; bacterial: 0.665; macromolecules; 0.662; colloids: 0.657; flagella: 0.653;	microorganisms: 0.808; swimmers: 0.747; organisms; 0.708; bacterial: 0.703; microswimmers: 0.693; proteins: 0.666;
	CbNs	motile: 0.709; microorganisms: 0.668; motility: 0.665; culture: 0.595; swimming: 0.594; magnetotactic: 0.593;	microorganisms: 0.808; motile: 0.763; microswimmers: 0.736; bacterial: 0.730; organisms: 0.727; sperm: 0.725	microorganisms: 0.814; algae: 0.776; motile: 0.772; microswimmers: 0.752; motility: 0.750; bacterial: 0.744
	GloVe	emulsions: 0.996; filaments: 0.995; clay: 0.994; snse:0.994; becs: 0.994; vesicles: 0.994;	macromolecules: 0.980; motile: 0.972; beings: 0.972; microorganisms: 0.968; biomolecules: 0.966; fibers: 0.965	motile: 0.969; macromolecules: 0.965; oocytes: 0.963; cytoskeleton: 0.961; tissues: 0.961; colonies: 0.957
cancer	SgHs	ketene: 0.441; cancerous: 0.573; vivo: 0.561; tissues: 0.560; apoptosis: 0.541; motility: 0.522	tumor: 0.614; cancerous: 0.573; vivo: 0.561; tissues: 0.560; apoptosis: 0.541; lamin-related: 0.524;	tumor; 0.658; fuel: 0.561; tissue: 0.545; antigen-presenting: 0.522; plant: 0.522; human: 0.520
	SgNs	prostate: 0.660; five-color: 0.625; vivo: 0.618; insoluble: 0.599; transporters: 0.593; delivery: 0.583	disease: 0.726; therapy: 0.720; neurodegeneration: 0.717; tumorigenesis: 0.716; drug: 0.713; tumor: 0.710	tumor: 0.776; disease: 0.723; drug: 0.714; delivery: 0.709; metastasis: 0.705; regulation: 0.705
	CbHs	ketene: 0.451; stereo-selectivity: 0.428; crawling: 0.427; biological: 0.403; heptamers: 0.392; signalised: 0.391	tumor: 0.628; apoptosis: 0.591; vivo: 0.578; plant: 0.576; cancererous: 0.549; blood: 0.546	tumor: 0.680; fuel: 0.564; cancerous: 0.553; apoptosis: 0.542; tissue: 0.527; plant: 0.524;
	CbNs	insoluble: 0.601; signaling: 0.577; drug: 0.562; delivery: 0.557; cracker: 0.553; vivo: 0.553	embryonic: 0.709; tumor: 0.692; mammalian: 0.691; metabolism: 0.675; apoptosis: 0.660; tissue: 0.656	tumor: 0.693; culture: 0.684; apoptosis: 0.682; embryonic: 0.673; mammalian: 0.669; tissue: 0.659
	GloVe	uptake: 0.995; lysozoyme: 0.994; wishart-laguerre: 0.993; concrete: 0.993; hydrated: 0.993; electromigration: 0.993;	tissue: 0.977; cathodes: 0.971; tandem: 0.969; microelectronics: 0.967; sequencing: 0.964; li-ion: 0.964	tissue: 0.970; smart: 0.966; urban: 0.965; tandem: 0.964; battery: 0.962; wireless: 0.959; interconnect: 0.959

Слика 5.6.5. Најсличније речи за речи *particles, collide, magnetic, statistics, bacteria, cancer* добијене креирањем *word2vec* и *GloVe* модела над насловима, апстрактима и оба

На слици 5.6.5 се може приметити да су коефицијенти сличности добијени помоћу наслова често много мањи од коефицијената добијених са већом количином података код свих варијанти *word2vec* алгоритма. С друге стране, коефицијенти добијени од стране *GloVe* алгоритма не зависе од количине података што такође доприноси закључку да је *GloVe* алгоритам у стању да добро ради са много мањом количином података.

У радовима у којима је описан *word2vec* речено је такође да је алгоритам у стању да одреди релације између речи и то у више различитих нивоа. Аутор је тестирао могућност алгоритма да одреди релације на основу семантичког значења (као што је дат пример у једначини 3.7.1) и ни један од тест примера није био задовољавајући те је закључак да је за одређивање такве врсте односа потребна много већа количина података. Такође, научни радови не дају пун опсег различитих могућности за релације између речи што је условило лоше перформансе у одређивању релација речи.

Други тип релација које би алгоритам требао да открије су релације између речи у једнини и множини или релације између глагола у различитим временима. Ова тестирања дала су боље резултате, при чему је алгоритам у свим варијантама био у стању да схвати релацију једнине и множине и нађе одговарајућу реч међу прве три најсличније речи (слика 5.6.6). С друге стране, алгоритам није схватио релацију између различитих времена и уместо тога су резултати најчешће укључивали синониме глагола за који је требало да се одреди друго глаголско време.

		Наслов	Апстракт	Оба
V(particles) - V(particle) + V(fields)	SgHs	field: 0.698; dipoles: 0.564	moments: 0.679; field: 0.658	moments: 0.673; field: 0.657
	SgNs	"creating: 0.543; external: 0.517	field: 0.648, order23456: 0.5794	field: 0.655; medium-driven: 0.599
	CbHs	field: 0.701; dipoles: 0.554	moments: 0.680; field: 0.652	moments: 0.660; field: 0.650
	CbNs	filed: 0.633; dipoles: 0.513	field: 0.624; dipoles: 0.558	field: 0.624, field;: 0.580
V(joining) - V(joined) + V(moving)	SgHs	multivortex: 0.435; bosons: 0.416; lines: 0.394	diffusing: 0.551; rotating: 0.513; propagating: 0.480	rotating: 0.579; curved: 0.523; staright: 0.504
	SgNs	fluxonics: 0.452; propagating: 0.445; gyromode: 0.445	move: 0.601; moves: 0.569; rotating: 0.567	move: 0.600; rotating: 0.569; movement: 0.564
	CbHs	antiresonant: 0.444; pinning: 0.429; lines: 0.416	diffusing: 0.559; curved: 0.546; rotating: 0.532	rotating: 0.531; bouncing: 0.517; straight: 0.503
	CbNs	centuries: 0.412; cavity: 0.399; by-product: 0.386	diffusing: 0.531; propagating: 0.530; move: 0.528	move: 0.558; drags: 0.537; movement: 0.537

Слика 5.6.6. Одређивање релација између речи помоћу *word2vec* алгоритма

На основу извршених тестирања за креирање језичког модела може се закључити да је за мање количине података боље креирати језичке моделе помоћу *GloVe* алгоритма, а у случају великих података боље је користити *word2vec*. Такође, одређене особине и могућности *word2vec* алгоритма не могу се добро испољити над овако формалним подацима. Због тога је препорука да се у случају рада са оваквим текстовима ради ригорозније претпроцесирање како би се побољшале перформансе алгортима. На основу извршених тестирања аутор је схватио да остављање одређених знакова интерпункције допринело лошијим, уместо бољим перформансама алгоритма. Додатно аутор препоручује уклањање свих формула пре рада са оваквим текстовима.

На крају, идеја целог рада је била да се документи кластеризују на основу вектора којим се они могу представити. Аутор је креирао скрипте за одређивање вектора код наслова, апстракта и наслова и апстракта заједно, међутим, ове операције су се испоставиле као

временски и меморијски веома захтевне и коришћена компјутерска конфигурација није могла да подржи веће фајлове као што су апстракти и наслови и апстракти заједно. Због тога су креирани вектори само за наслове помоћу *doc2vec* алгоритма за одређивање вектора параграфа. Такође, креирани су вектори наслова као сума вектора речи унутар наслова, при чему су вектори речи били одређени *word2vec* и *GloVe* алгоритмом. Нажалост, димензоналност модела *GloVe* алгоритма је била исувише велика и рачунар није био у стању да креира векторе наслова на основу њих.

За овако креиране векторе наслова имплементирана је кластеризација помоћу метода *k* средњих вредности и спектралне кластеризације, при чему је у оба случаја кластеризација вршена за 18 (број поткатегорија) и 148 (број тема) кластера. Рачунар није био у стању да изведе спектралну кластеризацију због недовољне меморије, док су резултати методе *k* средњих вредности дати у табели 5.6.1 за различите *doc2vec* и *word2vec* алгоритме.

Најбоље резултате при одређивању 18 кластера дала је метода *PVDBOW*, а при одређивању 148 кластера метода *SgNs*. Иако су у радовима у којима су ови алгоритми описани перформансе биле значајно боље код алгоритама изостављања н-грама, у овом раду не постоје велике разлике у перформансама између варијанти алгоритма *word2vec*. С друге стране, алгоритам *PVDBOW* је дао много боље резултате при кластеровању од алгоритма *PVDM* што није било очекивано на основу резултата оригиналног рада.

Табела 5.6.1. Нормализовани заједнички скор при одређивању 18 и 148 кластера за методе: *word2vec*, изостављање н-грама, хијерархијски софтмакс; *word2vec*, изостављање н-грама, негативно узорковање; *word2vec*, континуална врећа речи, хијерархијски софтмакс; *word2vec*, континуална врећа речи, негативно узорковање; *doc2vec*, дистрибутивни меморијски модел вектора параграфа; *doc2vec*, дистрибуирана врећа речи

	NMI (18 clusters)	NMI (148 clusters)
SgHs	0.16794	0.25234
SgNs	0.17130	0.25444
CbowHs	0.17556	0.25429
CbowNs	0.15404	0.25151
PVDM	0.00720	0.03258
PVDBOW	0.18536	0.20338

6. Закључак

У овом раду је дат опис најкоришћенијих метода за моделовање језика и кластеризацију, с нагласком на новије методе и методе које користе нелабелиране податке како би дале коначне резултате. За већину описаних метода дата је статистичка основа, осим тамо где то није било могуће. Представљене су и неке методе које се више користе над другим типовима података, а не над текстом, али су и за текст дале добре резултате и аутор овог рада сматра да имају потенцијала у овој области.

У поглављу пет су коришћени најбољи од представљених алгоритама да се групишу радови. Ови експерименти имали су за циљ да упореде различите методе независно од поређења која су дали аутори над својим подацима. Ни једна од ових метода није евалуирана над подацима са сајта *arxiv.org* до сада (колико је познато аутору овог мастер рада), те су они добра основа за објективно поређење свих метода одједном. Аутор је користио најбоље имплементације које је нашао и притом обратио пажњу да алгоритми имплементирани на сличан начин. Такође, коришћен је исти софтвер, хардвер, исте параметри где је могуће и додаване су еквивалентне измене/подешавања метода.

Други циљ експеримената јесте да се разуме колико су данашњи најбољи алгоритми у стању да кластеризују радове према њиховој семантичкој сличности и колико се кластери добијени на овај начин поклапају с категоризацијом сајта *arxiv.org* коју су дали истраживачи у одговарајућим пољима. Треба имати у виду да је учење машина и људи веома различито и да постоји пуно основа по којима се текстови могу груписати (нпр. тема, врста експеримента, објекат експериментисања, методологија...). Нажалост, кластеризацију добијену на овај начин није могуће упоредити са кластеризацијом коју би урадио обичан човек јер такви експерименти нису рађени са људима.

Резултати су показали да перформансе језичких модела приликом кластеровања значајно разликују од перформанси приликом надгледаног учења које су извршене у оригиналним радовима. При евалуацији језичких модела може се видети да *GloVe* алгоритам може радити веома добро и са мањом количином података, док word2vec алгоритам губи свој квалитет значајно како се количина података смањује. С друге стране, *GloVe* алгоритам није стабилан у односу на мале промене улазних података.

Треба напоменути да је потпуно исте конфигурације алгоритама могуће применити и над српским језиком, али је потребно прикупити довољно велики корпус речи. У неком даљем раду би се то могло пробати са текстовима прикупљеним из разних интернет новинских чланака. Такође, могуће је тестирати и упоређивати брзине алгоритама што није рађено у овом раду. За то аутор предлаже рад са имплементацијама само на језику C++ јер ће то дати најбрже резултате и омогућити више тестирања. У том случају могуће је такође истестирати понашање алгоритама приликом промене параметара.

У овом раду приказани су битни алгоритми за обраду језика и одређене су перформансе таквих модела при ненадгледаном учењу. Добијени резултати су показали да постоји могућност за даље испитивање и унапређење система развијеног од стране аутора. Овај систем се такође може испитати и на другим језицима.

Литература

[1] V. Batanović, B. Furlan, and B. Nikolić, "A software system for determining the semantic similarity of short texts in Serbian", *Telecommunications Forum (TELFOR), 19.* (str. 1249-1252). IEEE, *Nov. 2011*.

[2] POS Tagging And Token Classification By Using Bangla Tokenizer [Online]. Available: https://www.slideshare.net/sujitdas750331/token-classification (13.07.2017)

[3] T. Krilavičius, Ž. Medelis, J. Kapočiūtė-Dzikienė and T. Žalandauskas, "News media analysis using focused crawl and natural language processing: case of Lithuanian news websites. *Information and Software Technologies*, 48-61, 2012.

[4] Overall structure of a compiler [Online]. Available: https://engineering.purdue.edu/~milind/ece468/2015fall/lecture-01.pdf, str. 29, (14.07.2017)

[5] All About Stop Words for Text Minining and Information Retrieval [Online]. Available: http://text-analytics101.rxnlp.com/2014/10/all-about-stop-words-for-text-mining.html. (15.07.2017)

[6] Removing stop words with NLTK in Python [Online]. Available: http://www.geeksforgeeks.org/removing-stop-words-nltk-python/ (15.07.2017)

[7] C. D. Manning, P. Raghavan, and H. Schutze, "Introduction To Information Retrieval", *Cambridge University Press*, Cambridge, England, 2008.

[8] A. Schofield, M. Magnusson, D. Mimno, "Pulling Out the Stops: Rethinking Stopword Removal for Topic Models", *EACL*, 432, 2017.

[9] C. J. van Rijsbergen, S.E. Robertson and M.F. Porter, „New models in probabilistic information retrieval", *London: British Library*, (British Library Research and Development Report, no. 5587), 1980.

[10] M.F. Porter, An algorithm for suffix stripping, *Program*, **14**(3) pp 130−137, 1980.

[11] Stemming [Online]. Available: https://en.wikipedia.org/wiki/Stemming (16.07.2017)

[12] S. Vijayarani, M. J. Ilamathi, and M. Nithya, "Preprocessing techniques for text mining-an overview", *International Journal of Computer Science & Communication Networks*, *5*(1), 7-16, 2015.

[13] Milošević N., "Mašinska analiza sentimenta rečenica na srpskom jeziku", *master rad na Elektrotehničkom fakultetu Univerziteta u Beogradu, Katedra za računarsku tehniku i informatiku*, 2012.

[14] J. Plisson, N. Lavrac, D. Mladenic, "A rule based approach to word lemmatization", *Proceedings C of the 7th International Multi-Conference Information Society IS*, 2004.

[15] V. Balakrishnan, E. Lloyd-Yemoh, "Stemming and lemmatization: a comparison of retrieval performances", *Lecture Notes on Software Engineering* 2.3: 262, 2014.

[16] Lemmatizing with NLTK [Online]. Available: https://pythonprogramming.net/lemmatizing-nltk-tutorial/ (16.07.2017)

[17] Stanford CoreNLP – Natural language software [Online]. Available: https://stanfordnlp.github.io/CoreNLP/

[18] OpenNLP Manual [Online]. Available: http://opennlp.apache.org/docs/1.8.1/manual/opennlp.html (16.07.2017)

[19] Document Clustering with Python [Online]. Available: http://brandonrose.org/clustering#Stopwords,-stemming,-and-tokenizing (17.07.2017)

[20] Part of Speech Tagger [Online]. Available: http://opennlp.apache.org/docs/1.8.1/manual/opennlp.html#tools.postagger (17.07.2017)

[21] A. Ratnaparkhi, "A maximum entropy model for part-of-speech tagging", *Proceedings of the conference on empirical methods in natural language processing*, Vol. 1, 1996.

[22] Advanced Natural Language Processing [Online]. Available: http://cs.nyu.edu/courses/spring04/G22.2591-001/lecture3.html (18.07.2017)

[23] Chunker [Online]. Available: http://opennlp.apache.org/docs/1.8.1/manual/opennlp.html#tools.chunker (18.07.2017)

[24] L. A. Ramshaw, and M. P. Mitchell, "Text chunking using transformation-based learning." *Natural language processing using very large corpora*. Springer Netherlands, 157-176, 1999.

[25] T. Kudo, and Y. Matsumoto, "Chunking with support vector machines." *Proceedings of the second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies*. Association for Computational Linguistics, 2001.

[26] NLP Programming Tutorial 8 – Phrase Structure Parsing [Online]. Available: http://www.phontron.com/slides/nlp-programming-en-10-parsing.pdf (20.07.2017)

[27] The Standford Parser: A statistical parser [Online]. Available: https://nlp.stanford.edu/software/lex-parser.shtml (20.07.2017)

[28] G. Salton, and C. Buckley, "Term-weighting approaches in automatic text retrieval", *Information processing & management*, *24*(5), 513-523, 1988.

[29] NLTK API [Online]. Available: http://www.nltk.org/api/ (23.07.2017)

[30] TFIDFCalculator [Online]. Available: https://gist.github.com/guenodz/d5add59b31114a3a3c66 (24.07.2017)

[31] Bag of Words & TF-IDF [Online]. Available: https://deeplearning4j.org/bagofwords-tf-idf (24.07.2017)

[32] TFIDF EXPLAINED USING APACHE MAHOUT [Online]. Available: http://technobium.com/tfidf-explained-using-apache-mahout/ (25.07.2017)

[33] E. M. Riseman and A. R. Hanson, „A contextual postprocessing system for error correction using binary n-grams", *IEEE Transactions on Computers*, *100*(5), 480-493, 1974.

[34] T. Abou-Assaleh, N. Cercone, V. Keselj and R. Sweidan, „N-gram-based detection of new malicious code", In *Computer Software and Applications Conference, 2004, COMPSAC 2004, Proceedings of the 28th Annual International* (Vol. 2, pp. 41-42), IEEE, 2004.

[35] X. Wang, A. McCallum and X. Wei, „Topical n-grams: Phrase and topic discovery, with an application to information retrieval", In *Data Mining, 2007, ICDM 2007, Seventh IEEE International Conference on* (pp. 697-702), IEEE, 2007.

[36] A. Barrón-Cedeño and P. Rosso, „On automatic plagiarism detection based on n-grams comparison“, *Advances in Information Retrieval*, 696-700, 2009.

[37] V. Keselj, F. Peng, N. Cercone and C. Thomas, „N-gram-based author profiles for authorship attribution“, In *Proceedings of the conference pacific association for computational linguistics, PACLING* (Vol. 3, pp. 255-264), 2003.

[38] W. B. Cavnar and J. M. Trenkle, „N-gram-based text categorization“. *Ann Arbor MI, 48113*(2), 161-175, 1994.

[39] S. Jiampojamarn, “Automatic Biological Term Annotation Using n-gram and Classification Models”, *Dalhousie University, Halifax, Nova Scotia,* 2005.

[40] H. Liu, “Biological Information Extraction Using Patterns of Characters, Tag Sequences and Subgraphs”, *Dalhousie University, Halifax, Nova Scotia,* 2010.

[41] D. Jurafsky and J. H. Martin, “Speech and Language Processing” [Online]. Available: https://web.stanford.edu/~jurafsky/slp3/ (29.07.2017)

[42] Wikipedia, “N-gram” [Online]. Available: https://en.wikipedia.org/wiki/N-gram (29.07.2017)

[43] D. Migol, “Apache OpenNLP N-gram model creation example” [Online]. Available: http://www.denismigol.com/posts/291/apache-opennlp-n-gram-model-creation-example (30.07.2017)

[44] Wikipedia, “Google Ngram Viewer” [Online], Available: https://en.wikipedia.org/wiki/Google_Ngram_Viewer (30.07.2017)

[45] R. Albright, „Taming Text with the SVD“, *SAS Institute Inc., Cary, NC*, 2004.

[46] H. H. Su, „Singular value and singular value decomposition“ [Online], Available: http://www.math.jhu.edu/~hhaosu/Teaching/0203SLA/Notes-Ch8_3.pdf (31.07.2017)

[47] L. N. Trefethen and D. Bau III, “Numerical Linear Algebra (*Vol. 50)*”, *SIAM REVIEW 40*, 1997.

[48] D. H. Chau, “Text Analytics (Text Mining)” [Online], Available: http://poloclub.gatech.edu/cse6242/2014spring/lectures/CSE6242-20140403-TextLsiVisApp.pdf (31.07.2017)

[49] B. Tang, M. Shepherd, E. Milios and M. I .Heywood, „Comparing and combining dimension reduction techniques for efficient text clustering“, In *Proceeding of SIAM International Workshop on Feature Selection for Data Mining* (pp. 17-26), April 2005.

[50] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, „Indexing by latent semantic analysis“, *Journal of the American society for information science, 41*(6), 391, 1990.

[51] T. K. Landauer, P. W. Foltz and D. Laham, „An introduction to latent semantic analysis“, *Discourse processes“, 25*(2-3), 259-284, 1998.

[52] T. Hofmann, „Probabilistic latent semantic indexing“, In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval* (pp. 50-57). ACM, August 1999.

[53] M. Hasan, and Y. Masumoto, „Document clustering: before and after the singular value decomposition“, *Sapporo, Japan, Information Processing Society of Japan (IPSJ-TR: 99-NL-134.) pp*, 47-55, 1999.

[54] “An Introduction to Deep Learning” [Online], Available: https://blog.algorithmia.com/introduction-to-deep-learning-2016/ (1.8.2017)

[55] M. Mazur, “A Step by Step Backpropagation Example” [Online], Available: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/ (1.8.2017)

[56] Y. Bengio, R. Ducharme, P. Vincent and C. Jauvin, “A neural probabilistic language model“, *Journal of machine learning research, 3*(Feb), 1137-1155, 2003.

[57] H. Schwenk, “Continuous space language models“, *Computer Speech & Language*, *21*(3), 492-518, 2007

[58] A. Karpathy, „Convolutional Neural Networks“ [Online], Available: http://cs231n.github.io/convolutional-networks/ (3.8.2017)

[59] H. Fukui, T. Yamashita, Y. Yamauchi, H. Fujiyoshi, and H. Murase, „Pedestrian detection based on deep convolutional neural network with ensemble inference network“, In *Intelligent Vehicles Symposium (IV), 2015 IEEE*(pp. 223-228), IEEE, June 2015.

[60] A. Prakash, “One by One [1x1] Convolution – counter-intuitively useful” [Online], Available: http://iamaaditya.github.io/2016/03/one-by-one-convolution/ (3.8.2017)

[61] R. K . Srivastava, K. Greff and J. Schmidhuber,“Highway networks“, *International Conference for Machine Learning, ICML*, 2015.

[62] P. Blunsom, E. Grefenstette and N. Kalchbrenner, „A convolutional neural network for modelling sentences“, In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, 2014.

[63] J. Xu, W. Peng, T. Guanhua, X. Bo, Z. Jun, W. Fangyuan and H. Hongwei, „Short text clustering via convolutional neural networks“, 2015.

[64] N. Q. Pham, G. Kruszewski and G. Boleda, „Convolutional Neural Network Language Models“, In *EMNLP* (pp. 1153-1162), 2016.

[65] P. Roelants, “How to implement a recurrent neural network Part 1” [Online], Available: http://peterroelants.github.io/posts/rnn_implementation_part01/ (5.8.2017)

[66] T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur, „Recurrent neural network based language model“, In *Interspeech*(Vol. 2, p. 3), 2010.

[67] T. Mikolov, S. Kombrink, L. Burget, J. Černocký and S. Khudanpur, „Extensions of recurrent neural network language model“, In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on* (pp. 5528-5531), IEEE, 2011.

[68] T. Mikolov and G. Zweig, „Context dependent recurrent neural network language model“, *SLT, 12*, 234-239, 2012.

[69] W. De Mulder, S. Bethard and M. F. Moens, „A survey on the application of recurrent neural networks to statistical language modeling“, *Computer Speech & Language*, *30*(1), 61-98, 2015.

[70] “A Beginner’s Guide to Recurrent Networks and LSTMs” [Online], Available: https://deeplearning4j.org/lstm (8.8.2017)

[71] A. Mallya, "LSTM Forward and Backward Pass" [Online], Available: http://arunmallya.github.io/writeups/nn/lstm/index.html#/ (9.8.2017)

[72] M. Sundermeyer, R. Schlüter and H. Ney, „LSTM neural networks for language modeling", In *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.

[73] I. Sutskever, O. Vinyals, and Q. V. Le, „Sequence to sequence learning with neural networks", In *Advances in neural information processing systems*(pp. 3104-3112), 2014.

[74] T. Mikolov, K. Chen, G. Corrado and J. Dean, „Efficient estimation of word representations in vector space", *ICLR Workshop*, 2013.

[75] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado and J. Dean, „Distributed representations of words and phrases and their compositionality", In *Advances in neural information processing systems* (pp. 3111-3119), 2013.

[76] J. Pennington, R. Socher and C. D. Manning, „Glove: Global vectors for word representation", In *EMNLP* (Vol. 14, pp. 1532-1543), 2014.

[77] R. G. Gomez-Ol, "Deep Learning frameworks: a review before finishing 2016" [Online], Available: https://medium.com/@ricardo.guerrero/deep-learning-frameworks-a-review-before-finishing-2016-5b3ab4010b06 (12.8.2017)

[78] J. Misiti, "Awesome Machine Learning" [Online], Available: https://github.com/josephmisiti/awesome-machine-learning (12.8.2017)

[79] A. Vaswani, D. Chiang and V. Fossum, "Neural Probabilistic Language Model Toolkit" [Online], Available: https://nlg.isi.edu/software/nplm/ (13.8.2017)

[80] N. Q. Pham, "Convolutional Neural Network Language Models" [Online], Available: https://github.com/quanpn90/NCE_CNNLM (13.8.2017)

[81] D. Britz, "Implementing a CNN for Text Classification in TensorFlow" [Online], Available: http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/ (13.8.2017)

[82] D. Britz, „Recurrent Neural Networks Tutorial, Part 2 – Implementing a RNN with Python, Numpy and Theano" [Online], Available: http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-2-implementing-a-language-model-rnn-with-python-numpy-and-theano/ (14.8.2017)

[83] S. Enarvi, "TheanoLM" [Online], Available: https://github.com/senarvi/theanolm (14.8.2017)

[84] "K-Means Clustering" [Online], Available: https://lazyprogrammer.me/k-means-clustering/ (15.8.2017)

[85] T. Hardes, "Clustering" [Online], Avaialble: http://thardes.de/big-data-englisch/clustering/ (15.8.2017)

[86] K. Chen, "K-means Clustering" [Online], Available: https://studentnet.cs.manchester.ac.uk/ugt/COMP24111/materials/slides/K-means.pdf (16.8.2017)

[87] "K-means" [Online], Available: https://nlp.stanford.edu/IR-book/html/htmledition/k-means-1.html (16.8.2017)

[88] „Clustering" [Online], Available: http://scikit-learn.org/stable/modules/clustering.html (17.8.2017)

[89] "Evaluation of clustering" [Online], Available: https://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-clustering-1.html (17.8.2017)

[90] "Hierarchical Clustering Algorithms" [Online], Available: https://home.deib.polimi.it/matteucc/Clustering/tutorial_html/hierarchical.html (18.8.2017)

[91] "Hierarchical Clustering" [Online], Available: http://www.statisticshowto.com/hierarchical-clustering/ (18.8.2017)

[92] „Hierarchical Clustering Java" [Online], Available: https://github.com/lbehnke/hierarchical-clustering-java (18.8.2017)

[93] U. von Luxburg, "A Tutorial on Spectral Clustering", *Max Planck Institute for Biological Cybernetics*, 2006.

[94] A. Y. Ng, M. I. Jordan, Y. Weiss, "On spectral clustering: Analysis and an algorithm", *In Advances in neural information processing systems (pp. 849 – 856)*, 2002.

[95] Q. Le, and T. Mikolov, „Distributed representations of sentences and documents", In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)* (pp. 1188-1196), 2014.

[96] Y. Shi, W. Q. Zhang, J. Liu and M. T. Johnson, „RNN language model with word clustering and class-based output layer", *EURASIP Journal on Audio, Speech, and Music Processing*, *2013*(1), 22, 2013.

[97] M. Klapper-Rybicka, N. N. Schraudolph and J. Schmidhuber, „Unsupervised learning in LSTM recurrent neural networks", In *International Conference on Artificial Neural Networks* (pp. 684-691). Springer, Berlin, Heidelberg, Avgust 2001.

[98] A. Dundar, J. Jin, and E. Culurciello, „Convolutional clustering for unsupervised learning". *arXiv preprint arXiv:1511.06241*, 2015.

[99] D. Zeng, K. Liu, S. Lai, G. Zhou, and J. Zhao, „Relation Classification via Convolutional Deep Neural Network", In *COLING* (pp. 2335-2344), August 2014

[100] M. Sadjadi, "Arxivscraper" [Online], Available: https://github.com/Mahdisadjadi/arxivscraper (23.8.2017)

[101] R. Rehurek, „Gensim API Reference" [Online], Available: https://radimrehurek.com/gensim/apiref.html (25.8.2017)

[102] M. Kula, "Glove Python" [Online], Available: https://github.com/maciejkula/glove-python (27.8.2017)

Списак скраћеница

TF-IDF	*Term Frequency–Inverse Document Frequency*
SVD	*Singular Value Decomposition*
word2vec	*Word to vector*
GloVe	*Global Vectors for Word Representation*
NLTK	*Natural Language Toolkit*
CoreNLP	*Core Natural Language Processing*
OpenNLP	*Open Natural Language Processing*
NER	*Named Entity Recognition*
MI	*The Mutual Information Method*
TBRS	*Term Based Random Sampling*
NLP	*Natural Language Processing*
IR	*Information Retreival*
PCA	*Principal Component Analysis*
LSI	*Latent Semantic Indexing*
NN	*Neural networks*
ReLu	*Rectified Linear Unit*
CNN	*Convolutional Neural Network*
RNN	*Recurrent Neuraln Network*
LSTM	*Long short-term memory*
LDA	*Latent Dirichlet Allocation*
CBOW	*Continuous Bag of Words*
NCE	*Noise Contrastive Estimation*
NMI	*Normalized Mutual Information*
FMI	*Fowlkes-Mallows Index*
PV-DM	*Distributed Memory Model of Paragraph Vectors*
PV-DBOW	*Distributed Bag of Words version of Paragraph Vector*
BINGO	*Binary Information Gain Optimization*
NEO	*Nonparametric Entropy Optimization*
API	*Application Programming Interface*
TSNE	*t-distributed stochastic neighbor embedding*

Списак слика

Списак табела