Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Žiga Pušnik

Uporaba globokih konvolucijskih nevronskih mrež na jezikovnih problemih

diplomsko delo

univerzitetni študij prve stopnje računalništvo in informatika

Mentor: prof. dr. Marko Robnik-Šikonja

Ljubljana 2015

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Globoke konvolucijske nevronske mreže so se izkazale z zelo dobrim delovanjem na problemih procesiranja vizualne informacije. V zadnjem času se je pojavilo več poskusov njihove uporabe tudi na drugih področjih. Preučite pristope k procesiranju naravnega jezika s pomočjo konvolucije in globokih nevronskih mrež in jih uporabite na dveh jezikovnih problemih, klasifikaciji člankov v kategorije in postavljanju vejic v slovenskih besedilih. Predlagajte ustrezno arhitekturo globoke mreže, mrežo implementirajte s knjižnico Theano in delovanje mreže statistično ovrednotite.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Žiga Pušnik sem avtor diplomskega dela z naslovom:

Uporaba globokih konvolucijskih nevronskih mrež na jezikovnih problemih

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Marka Robnika-Šikonje,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 8. avgusta 2015 Podpis avtorja:

Zahvaljujem se mentorju prof. dr. Marku Robniku-Šikonji za strokovno pomoč in vodenje pri opravljanju diplomskega dela.

Zahvaljujem se svoji družini, ki me je vsa leta spodbujala pri izobraževanju.

Zahvaljujem se vsem, ki so mi pomagali na poti do izobrazbe.

Kazalo

Povzetek

Cilj diplomske naloge je preizkusiti učenje jezikovnih problemov s pomočjo globokih konvolucijskih nevronskih mrež. Konvolucijske nevronske mreže so bile razvite predvsem za področje umetnega zaznavanja in delujejo na podlagi konvolucije. Naučili smo jih, da so na podlagi kratkega povzetka besedila napovedale razred, h kateremu spada. Drugi problem, ki smo ga reševali je postavljanje vejic v slovenskem jeziku. Konvolucijsko nevronsko mrežo smo sprogramirali s programskim jezikom python. Uporabili smo knjižnjico Theano. Izhajali smo iz že obstoječih raziskav. Opišemo način, kako smo obdelali podatkovne množice, da so primerne za naš model. Opravili smo več poskusov. Primerjali smo lematizacijo in krnjenje ter predstavitev besedila z vektorizacijo in predstavitev z bitnim poljem. Zadovoljive rezultate smo dobili, če smo besedilo kvantizirali, kjer smo črke vektorizirali z 1 do m kodiranjem. Naši rezultati pri postavljanju vejic so primerljivi z rezultati drugih raziskav.

Ključne besede: strojno učenje, obdelava naravnega jezika, nevronska mreža, nevron, konvolucija, konvolucijska nevronska mreža, klasifikacija, klasifikacijski model, klasifikator, klasifikacijska točnost, jezik, besedilo, vejica, lema, krn, moment, gradient, gradientni spust, vzratno širjenje napake, učenje, stopnja učenja, podatkovna množica, jezikovni korpus, atribut.

Abstract

The thesis examines the learning of language problems with convolutional neural networks. Convolutional neural networks were developed for machine vision. We used them to classify short abstracts and to learn a comma placement in Slovenian language. We programmed our convolutional neural network in programming language python with Theano library. Our work is based on existing research. We describe adaptation of datasets to our model. Several experiments were conducted and we compared lemmatization versus stemming and vector representation of text versus byte array representation. The best results were obtained with text quantized with 1 to m encoding. Comma placing results are comparable with other machine learning approaches.

Keywords: machine learning, natural language processing, neural network, neuron, convolution, convolutional neural netvork, clasification, clasification model, clasificator, clasification accuracy, language, text, comma, lemma, stemm, momentum, gradient, gradient descent, backpropagation, learning rate, momentum rate, dataset, text corpus , attribute.

Poglavje 1

Uvod

Razvoj tehnologije, kot sredstva za izboljšanje kakovosti življenja, poganjajo tri stvari: človeška želja po raziskovanju, posnemanje vzorcev in rešitev iz narave ter uporaba že obstoječe tehnologije na različnih področjih. Kot primer je mogoče navesti osebne računalnike, ki jih uporabljamo že na mnogih področjih našega življenja. Tudi področje umetne inteligence je tovrsten primer, cilj pa je posnemati človeško razmišljanje. Tudi tukaj je mnogo principov povzeto iz narave, npr. genetski algoritmi ali nevronske mreže.

Procesiranje naravnega jezika (NLP, natural language processing) je del umetne inteligence že od nastanka le-te. Tako je Alan Turing leta 1950 v članku *Computing machinery and inteligence* [17] predlagal tako imenovan Turingov test. To je test, kjer udeleženec nekaj časa komunicira z entiteto v drugi sobi ter na koncu poda svoje mnenje, ali je komuniciral s človekom ali s strojem. V primeru, da sogovornik ne loči stroja od osebe, lahko smatramo takšen sistem inteligenten. Do danes ni še nobena naprava opravila tega testa, saj bi moral stroj vedeti skoraj vse o zunanjem svetu, tudi stvari, ki se nam ljudem zdijo samoumevne. Zavedamo se, da če vržemo kamen v zrak, bo priletel nazaj na tla (morda tudi nam na glavo), stroju pa moramo takšne stvari vnesti ali pa pustiti, da se preko interakcije z zunanjim svetom sam nauči takšnih zakonov, kar pa je dolgotrajen in nezanesljiv postopek. Slednje je eden izmed razlogov, da so se raziskovalci spočetka omejili na tako

imenovane mikro svetove, kjer imamo omejen prostor ter omejeno število nastopajočih objektov, npr. osnovnih geometrijskih likov. Primer je Terry Winograd s programom SHRDLU [20], ki nadzira robotsko roko in se odziva na ukaze, kot so: *"Prosim, prestavi rdečo kocko na zelen valj"*. Navkljub težavam je postal NLP del našega življenja, zavedno in nezavedno se poslužujemo uspehov raziskovalcev že desetletja. Tako lahko besedilo prevedemo z orodjem Google Translate iz jezika, ki ga ne razumemo, v jezik, ki nam je domač, ali pa se zatipkamo in nam brskalnik poda predlog, kaj smo želeli poiskati. Če nam je dolgčas, lahko poklepetamo s spletnim klepetalnikom, kot je ALICE Chatbot. V te namene so bile razviti tehnologije kot so POS (part of speech) označevanje, lematizacija, vektorska predstavitev besedil in primerni jezikovni korpusi. Na voljo je mnogo programskih knjižnjic, ki olajšajo procesiranje naravnega jezika, tudi v slovenskem jeziku.

Vzporedno je bilo za potrebe umetnega zaznavanja razvitih mnogo pristopov. Eden od njih so konvolucijske nevronske mreže, ki so pravzaprav nadgradnja preprostih nevronskih mrež. Uporabljajo se predvsem za prepoznavo človeških obrazov [11], razpoznavanje ročno pisanih številk (raziskovalna podatkovna baza MNIST [12]) in drugih objektov, ki jih tako ali drugače zaznavamo v vsakdanjem življenju.

V nalogi bomo obstoječ pristop strojnega učenja s konvolucijskimi nevronskimi mrežami, ki so sicer namenjene predvsem za umetno zaznavanje, prilagodili za procesiranje naravnega jezika. Tak pristop je opisan že v članku [21], kjer so na opisih entitet, pridobljenih iz DBpedie [13], učili konvolucijske nevronske mreže na predelanem besedilu. Črke so najprej kvantizirali in besede zamenjali s sinonimi. Rezultati, ki so jih dosegli, so pokazali, da so konvolucijske nevronske mreže primerne za učenje jezika. Tudi mi smo poskusili ta pristop na kratkih povzetkih besedil, z namenom, da jih klasificiramo v razrede kot so žival, rastlina, pisno delo, glasbeno delo, naravni dogodek in druge. Opravili smo tudi več poskusov in rezultate primerjali.

Zanima nas tudi, kako se konvolucijske nevronske mreže odrežejo pri problemih, kjer namesto surovih podatkov, kot so slike ali besedilo, za klasifika-

cijo uporabljamo atribute, ki so zgenerirani na podlagi pravil. Zato opravimo tudi nekaj poskusov na podatkovni množici postavljanja vejic v slovenskem jeziku.

V poglavju 2 so podrobneje opisane konvolucijske nevronske mreže in njihovo delovanje. Za nadzorovano učenje potrebujemo velike podatkovne množice. V poglavju 3 opišemo, kako so bile pridobljene ter kako so urejene. Ker se konvolucijske nevronske mreže uporabljajo pri umetnem zaznavanju, je treba temu ustrezno prilagoditi predprocesiranje besedil, zato v poglavju 4 opišemo, kakšni pristopi so so bili uporabljeni. V poglavju 5 so kritično ovrednoteni in komentirani rezultati testov. Na koncu v poglavju 6 zaključimo, povzamemo narejeno, analiziramo uporabljen pristop in podamo predloge za izboljšave in nadaljno delo.

Poglavje 2

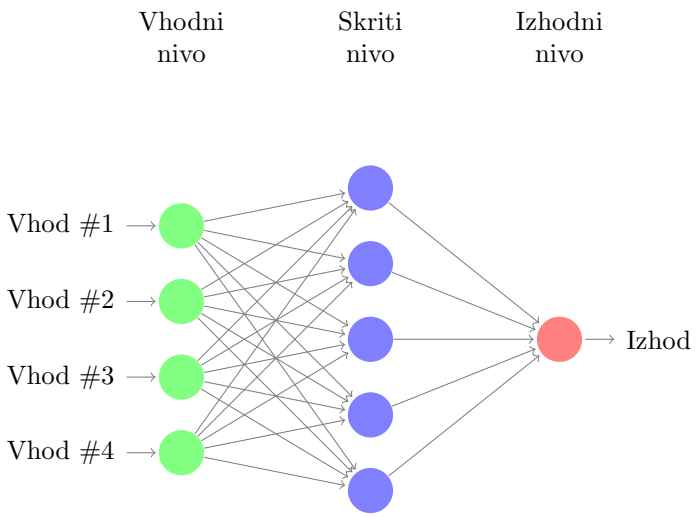
Konvolucijske nevronske mreže

Poglavje je namenjeno razumevanju konvolucijskih nevronskih mrež. Najprej v podpoglavju 2.1 opišemo ter z enačbami ponazorimo delovanje preprostih nevronskih mrež. Razložimo gradientni spust z vzratnim širjenjem napake z momentom. V podpoglavju 2.2 je opisana in grafično ponazorjena konvolucija kot matematična operacija ter zakaj je primerna za strojno učenje, razložimo pristop združevanja maksimalnih vrednosti in na koncu predstavimo konvolucijsko nevronsko mrežo.

2.1 Preproste umetne nevronske mreže

2.1.1 Opis

Preprosto umetno nevronsko mrežo (slika 2.1) lahko opišemo kot poln, usmerjen, n-delen acikličen graf, kjer je $n \geq 2$, število $n-2$ pa predstavlja število skritih nivojev v nevronski mreži. Vsak nivo, razen vhodnega in izhodnega, je sestavljen iz poljubnega števila nevronov. Vhodni sloj ima toliko nevronov kot je atributov, izhodni sloj pa vsebuje enako število nevronov, kot je vseh razredov pri klasifikaciji in enega pri regresiji. Pri regresiji poda izhodni nevron oceno regresijske spremenljivke. Pri klasifikaciji predstavlja vsak izhodni nevron svoj razred, njegovo vrednost pa si lahko predstavljamo kot verjetnost določenega razreda R_i pri danem primeru $\hat{X}$ (2.1). Usmerjene



Slika 2.1: Nevronska mreža z enim skritim nivojem, štirimi nevroni na vhodu in enim na izhodu.

povezave predstavljajo sinapse, kjer ima vsaka povezava svojo utež. Tukaj lahko najdemo analogijo, saj si lahko velikost uteži zamislimo kot debelino sinapse. Kadar v možganih podatek potuje po sinapsi, se signal ojača [8].

$$P(R = R_i \mid \hat{X}) \tag{2.1}$$

Vsak nevron sešteva utežene vrednosti svojih vhodov in jih normalizira z vnaprej podano aktivacijsko funkcijo. Kljub temu, da ta funkcija ni točno določena, se v praksi najpogosteje uporablja sigmoida (2.2), z definicijskim območjem realnih števil in zalogo vrednosti na odprtem intervalu $(0,\ 1)$. Sigmoida ima lepo lastnost (2.3). Primerne normalizacijske funkcije so še hiperbolični tangens in stopničasta funkcija.

$$S(t) = \frac{1}{1 + e^{-t}} \tag{2.2}$$

$$\frac{\mathrm{d}y}{\mathrm{d}t} S(t) = S(t)(1 - S(t)) \tag{2.3}$$

Pomembno vlogo ima tudi vrednost pristranskosti (bias value) b, to je vrednost, ki jo nevron doda že uteženi vsoti. Ta vrednost igra pomembno vlogo,

saj zamakne aktivacijsko funkcijo po definicijskem območju, kar se izkaže za ključni faktor pri učenju umetnih nevronskih mrež. Matematičen zapis izhoda iz nevrona i prikazuje enačba (2.4) in ponazarja slika 2.2.

Legenda:

X_i ... izhod i-tega nevrona

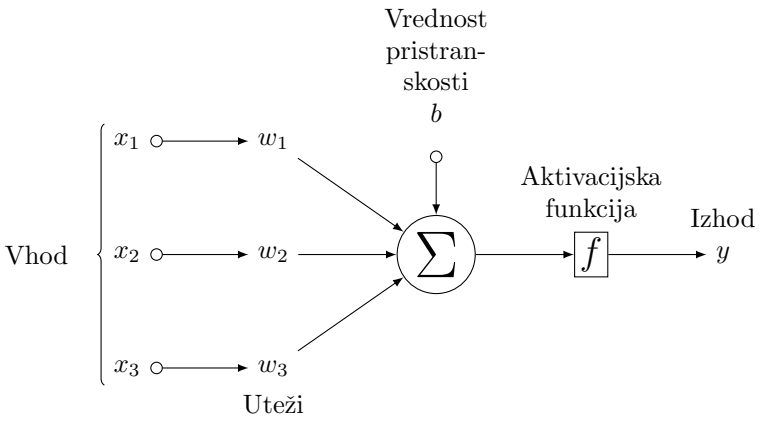
X_j ... izhod iz j-tega nevrona

W_{ji} ... utež, ki povezuje nevrona j in i

b_i ... pristranska vrednost nevrona i

in_i ... vhod v aktivacijsko funkcijo S

$\hat{y}$ ... klasifikacijski vektor, ki je 1, če $R = R_i$, drugače 0



Slika 2.2: Izhod enega nevrona.

$$X_i = S(in_i), \text{ kjer } in_i = \sum_j X_j W_{ji} + b_i \tag{2.4}$$

2.1.2 Gradientni spust

Ena izmed pristopov nadzorovanega učenja nevronskih mrež je gradientni spust (gradient descent), ki minimizira napako, oziroma maksimizira klasifikacijsko točnost. Gradientni spust je optimizacijski algoritem, primeren za različne vrste optimizacij pri matematičnem modeliranju in strojnem učenju. Pripelje nas do lokalnega minimuma. Gradient je vektor, ki kaže v smeri največjega vzpona funkcije f. Sestavljen je iz parcialnih odvodov vseh spremenljivk funkcije f (2.5). Ker kaže gradient v smeri največjega vzpona, gradientni spust napreduje v nasprotni smeri, v smeri največjega padanja funkcije f.

$$\nabla f = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3}, ... \frac{\partial f}{\partial x_n}) \tag{2.5}$$

Eden izmed načinov kako lahko zapišemo funkcije napake C je (2.6), ter nato z verižnim pravilom za odvajanje izpeljemo enačbo $\frac{\partial C}{\partial \hat{W}}$ (2.7). Vsako utež W_{ij} nato posodobimo tako, da ji prištejemo ΔW_{ij} (2.9) in Δb_i. To nam omogoča, da posodobimo uteži samo na zadnjem sloju nevronske mreže. Izkazalo se je, da so nevronske mreže brez skritih nivojev sposobne reševati samo linearne probleme, zato je potrebno dodati skrite nivoje. Če spremenimo izhod nevrona na skritem nivoju, ta vpliva na izhod vseh nevronov na vseh sledečih nivojih, kar pri nevronski mreži z dvema nivojema ni mogoče. Vpeljava skritih nivojev zmanjša število potrebnih nevronov za klasifikacijo, tudi za eksponenten faktor [4]. Zaradi naštetih razlogov je bila predlagana metoda vzratnega širjenja napake (backpropagation).

Legenda:

C ... funkcija napake

$f(\hat{X})$ ... izhod zadnjega nivoja nevronske mreže

m ... število primerov, ki jih zavzamemo v enem obhodu

α ... stopnja učenja (največrat realno število med 0 in 1)

y_i ... 1, če $R = R_i$, drugače 0

$$C = \frac{1}{2m} \sum_{m} (\hat{y} - f(\hat{X}))^2 \tag{2.6}$$

$$\Delta W_{ij} = \alpha \Delta_i X_j, \text{ kjer } \Delta_i = (y_i - S(in_i))S'(in_i) \tag{2.7}$$

$$\Delta b_i = \alpha \Delta_i \tag{2.8}$$

$$W_{ij} = W_{ij} + \Delta W_{ij} \tag{2.9}$$

Vzratno širjenje napake

Omenili smo že, da vpeljava novih skritih nivojev zmanjšuje potrebno število nevronov in nam omogoča, da rešujemo tudi nelinearne probleme. S tem namenom so se začele uporabljati tako imenovane globoke arhitekture. Zato je potrebno znati naučiti tudi uteži skritih nivojev. To nam omogoča vzratno širjenje napake, kjer napako posredujemo nazaj iz zadnjih nivojev proti prvim (2.10).

$$\Delta_j = S'(in_j) \sum_{i} W_{ij} \Delta_i \tag{2.10}$$

Gradientni spust z momentom

Ker se mnogokrat zgodi, da se nevronska mreža pri učenju ujame v lokalni minimum, se uporablja moment. To se zgodi, kadar ima funkcija napake lokalni minimum, v katerega se gradient ujame in iterira v neskončnost po dveh pobočjih funkcije, kar prikazuje slika 2.3. Moment je nadgradnja gradientnega spusta in je rekurzivna formula (2.11), s katero priredimo gradientni spust tako, da vsebuje majhen delež λ (stopnja momenta) vseh prejšnjih gradientov. Če je funkcija napake dovolj strma, ustvari to preko nekaj iteracij velik moment, saj se ta s seštevanjem veča. Ko pridemo do pobočja in gradient obrne smer, zaradi velikega momenta še vedno napredujemo v prvotni smeri. Kot bi se s sankami spustili po griču in prevozili izboklino. Tako

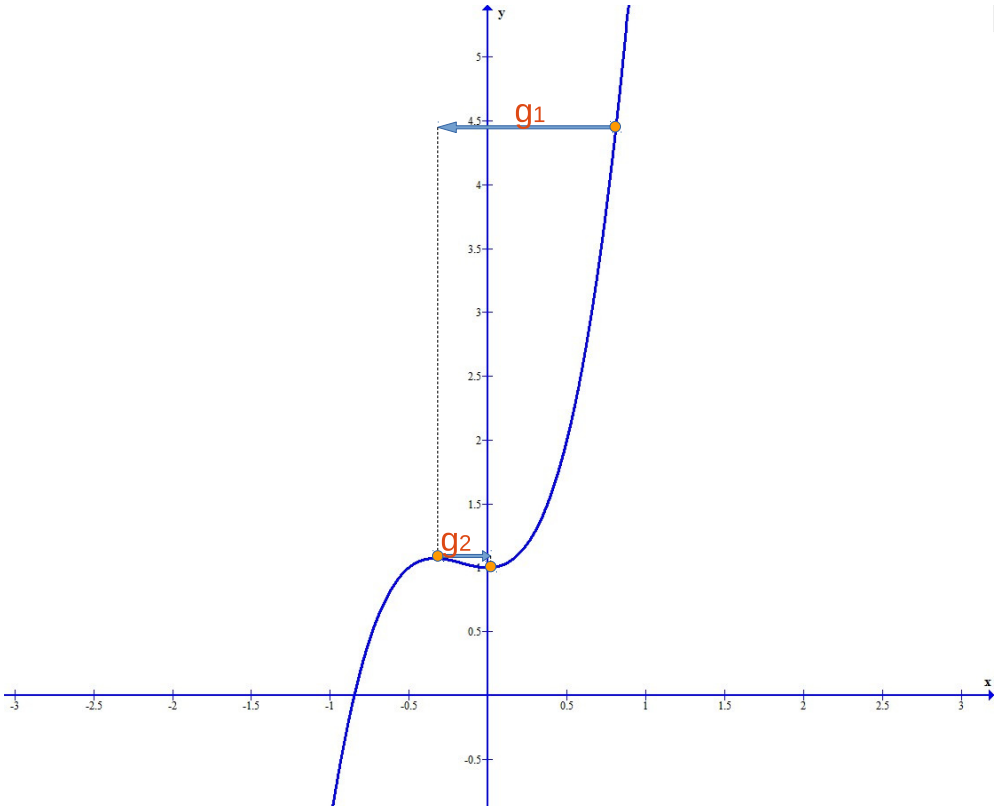
predstavlja hitrost naleta oziroma pospešek delež momenta, ki mu ponavadi določimo vrednost na odprtem intervalu (0,1). Če je delež enak 0, govorimo o navadnem gradientnem spustu, drugače pa moment i skozi iteracije pada eksponentno, zato se s spuščanjem v globino rekurzivne formule gradienti porazgubijo, vendar metoda omogoča, da se včasih rešimo iz lokalnega minimuma. Slika 2.4 ponazarja ravno takšen scenarij.

Legenda:

Θ ... matrika, ki vsebuje vse uteži vseh nevronov na poljubnem nivoju

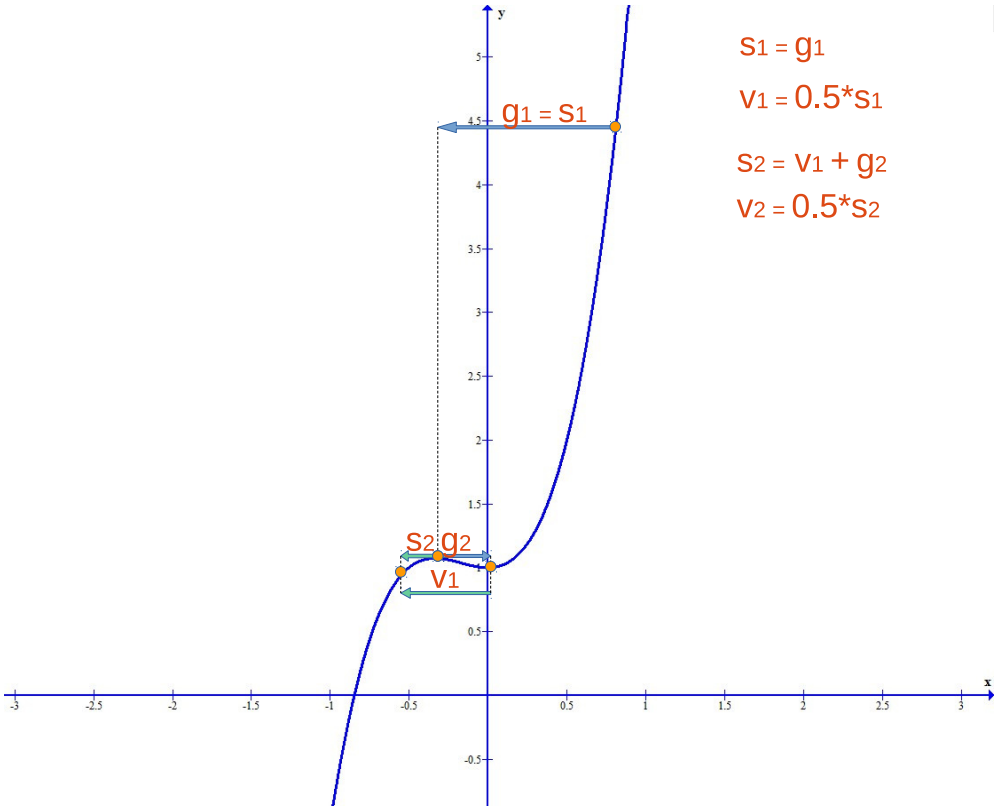
v_i ... moment v v i-ti iteraciji

λ ... vrednost, s katero pomnožimo moment v



Slika 2.3: Slika prikazuje gradientni spust brez momenta na polinomu $f(x) = 4x^3 + 2x^2 + 1$, kjer v dveh iteracijah obtičimo v lokalnem minimumu.

$$v_{i+1} = \lambda v_i + \alpha \Delta_i \hat{X}; \Theta = \Theta + v_{i+1} \tag{2.11}$$



Slika 2.4: Slika prikazuje gradientni spust z uporabo momenta na polinomu $f(x) = 4x^3 + 2x^2 + 1$, kjer se iz lokalnega minimuma uspemo rešiti. g_i predstavlja gradient v točki i, s_i dejanski spust in v_i moment v točki i. Stopnja momenta je enaka eni polovici.

2.2 Konvolucijske nevronske mreže

2.2.1 Konvolucija kot matematična operacija

Konvolucija je matematična operacija med dvema funkcijama $I(x)$ (vidno polje) in $g(x)$ (filter), kjer je rezultat $I_g(x)$ prav tako funkcija, ki si jo lahko predstavljamo kot transformiranko funkcije $I(x)$ s filtrom $g(x)$. Konvolucija z jedrom $g(x)$ in sliko $I(x)$ je definirana s spodnjo enačbo (2.12), oziroma v diskretnem prostoru z (2.13).

$$I_g(x) = g(x) * I(x) = \int_{-\infty}^{+\infty} g(u)I(x-u)\mathrm{d}u \tag{2.12}$$

$$I_g(x) = g(x) * I(x) = \sum_{u=-\infty}^{+\infty} g(u)I(x-u) \tag{2.13}$$

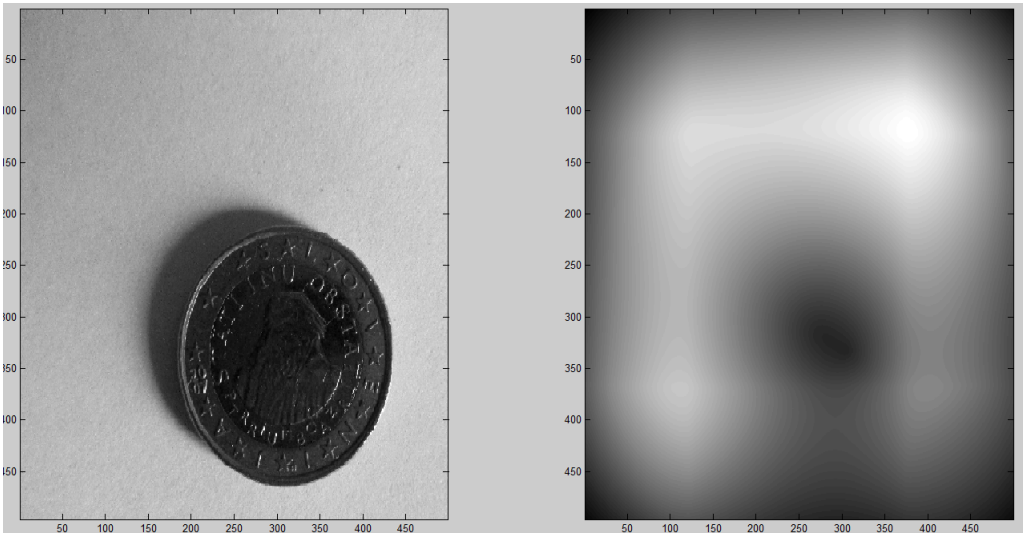
Konvolucija je nepogrešljiv element pri umetnem zaznavanju, saj lahko sliko konvoliramo z raznimi filtri, ki delujejo kot npr. meglenje ali detektor robov. Slika 2.5 prikazuje originalno sliko (lena.png) uporabljeno v članku [14] konvolirano z Gaussovim filtrom velikosti 10x10 in standardnim odklonom velikosti 5 ter ekstrapolirane robove, ki so bili pridobljeni s filtrom:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Konvolucija je primerna tudi za detekcijo. S filtrom podobnim objektom lahko detektiramo preproste objekte. Slika 2.6 prikazuje originalno sliko z objektom, ki ga želimo detektirati, ter rezultat kovolucije s primernim filtrom. Filter je okrogle oblike, z enicami v krogu, vse ostalo so ničle. Opazimo lahko, da je na poziciji kovanca zgostitev črnih točk, to pomeni, da je filter pri kovancu posredoval maksimalne vrednosti. Če določimo neko pragovno vrednost, lahko izluščimo približno pozicijo kovanca, oziroma ga detektiramo.

Ker je gradientni spust z vzratnim širjenjem napake z uporabo momenta počasna metoda in se z vpeljavo konvolucije v nevronsko mrežo časovna

Slika 2.5: Iz leve proti desni: originalna slika, zamegljena slika z Gaussovim filtrom ter slika z ekstrapoliranimi robovi.



Slika 2.6: Slika kovanca konvolirana s primernim filtrom.

kompleksnost le poveča, se poslužujemo metode združevanja maksimalnih vrednosti (max pooling), ki zmanjšuje število parametrov in s tem poveča hitrost algoritma.

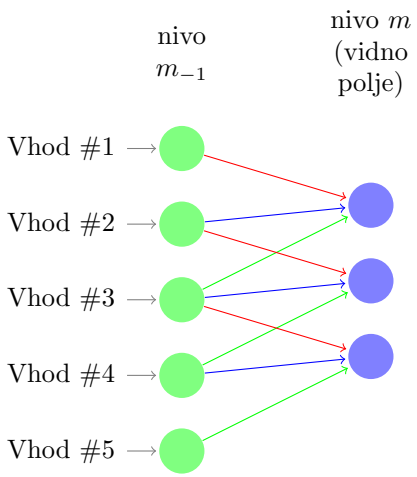
2.2.2 Združevanje maksimalnih vrednosti

Združevanje maksimalnih vrednosti (max pooling) je tehnika, s katero na preprost način zmanjšamo število parametrov. To storimo tako, da z vnaprej določeno velikostjo okna drsimo po vidnem polju s korakom velikosti okna. Na tak način zagotovimo, da ne pride do prekrivanja. Pri vsakem koraku ohranimo le največjo vrednost, ostale vrednosti pa zanemarimo. Maksimalne vrednosti združimo v vidno polje (feature map), ki je vhod v naslednje konvolucijske nivoje. Tako na učinkovit način zmanjšamo število parametrov ter zagotovimo na premik invarianten model. Primerneje bi bilo, da združujemo povprečne vrednosti, vendar se tega ne poslužujemo zaradi računske zahtevnosti, ki bi jo povečali brez opaznih rezultatov.

2.2.3 Opis

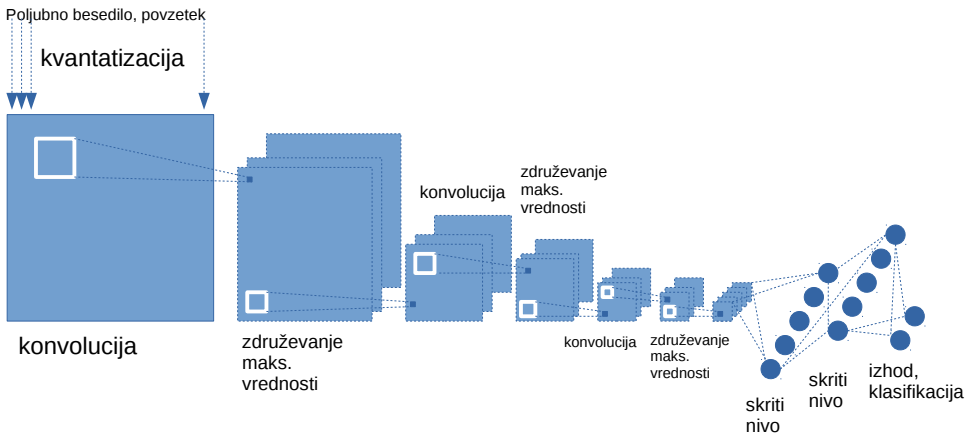
Če konvolirana vidna polja povežemo v kaskado in na koncu dodamo polno povezane nivoje nevronov govorimo o konvolucijski nevronski mreži. Konvolucijske nevronske mreže, se uporabljajo predvsem na področju umetnega zaznavanja, za kar so bile tudi razvite. Delujejo na principu konvolucije, kjer z različnimi filtri konvoliramo sliko. Omenili smo že, da lahko s pomočjo kovolucije sliko transformiramo ali pa zaznavamo objekte, zaradi česar so kovolucijske nevronske mreže primerne za strojno učenje. Z njihovo pomočjo lahko uspešno klasificiramo predmete na slikah, prepoznavamo osebe, ločimo med moškim in ženskim spolom, itd.

Pri konvolucijski nevronski mreži si lahko predstavljamo filter kot množico uteži, ki se ponavlja čez celotno vidno polje. Enodimenzionalni primer ponazarja slika 2.7. Tudi na konvolucijskih nivojih lahko uporabimo pristope nadzorovanega strojnega učenja, kot je gradientni spust, vzratno širjenje napake in moment, le z razliko, da je zaradi deljenih uteži deljena tudi ΔW_{ij} in jo prištejemo, kjerkoli se ta utež pojavi.



Slika 2.7: Nivo m_{-1} konvoliran s filtrom velikosti 3, rezultat je nivo m.

Celotna zgradba konvoluvijske nevronske mreže je prikazana na sliki 2.8.



Slika 2.8: Struktura konvolucijske nevronske mreže za besedilne podatke.

Poglavje 3

Podatkovne množice

V tem poglavju opišemo naše podatkovne množice. Za primerjavo rezultatov sta kot v članku [21] uporabljeni DBpedia in ontologija kot vir podatkov za klasifikacijo. Najprej opišemo projekt DBpedia, razložimo koncept ontologije, opišemo sestavo podatkovnih množic ter kako smo jih pridobili. Dotaknemo se tudi postavljanja vejic v slovenskem jeziku in opišemo uporabljeni učni korpus.

3.1 DBpedia

DBpedia je projekt, kjer je iz Wikipedie pridobljena semantično strukturirana podatkovna množica, ki omogoča napredne poizvedbe in je tako velik korak k semantičnemu spletu [13]. Podatkovna množica je dosegljiva v RDF (Resource Description Framework) formatu. RDF format je format iz družine World Wide Web Consortium (W3C) specifikacij, načrtovan kot metapodatkovni model [19]. Uporablja se predvsem v medmrežnih komunikacijah, kjer prenašamo strukturirane in hierarhično soodvisne podatke. Njegove tri glavne komponente so subjekt (identifikator vira), predikat (ime lastnosti) in objekt (vrednost lastnosti). Po RDF podatkovnih množicah lahko poizvedujemo z RDF poizvedovalnim jezikom SPARQL (SPARQL Protocol and RDF Query Language), ki zaradi svoje preprostosti in podobnosti z poizvedoval-

nim jezikom SQL (Structured Query Language) omogoča hitro in preprosto poizvedovanje. Ker so podatki pridobljeni iz strani Wikipedie ter semantično strukturirani, je količina informacij velika, zato je DBpedia kakovosten vir za pridobivanje in generiranje podatkovnih množic, ki so namenjene znanstvenim raziskavam, predvsem pri obdelavah naravnega jezika in strojnem učenju.

3.1.1 Ontologija

„Ontologija je filozofska disciplina, ki obravnava osnovo, vzroke in najsplošnejše lastnosti stvarnosti in načela, ki obravnavajo osnovo, vzroke in najsplošnejše lastnosti česa.“ Povzeto po (Slovar slovenskega knjižnega jezika, 1975).

V informatiki so ontologije formalne predstavitve množic in konceptov med njimi, ki vsebujejo objekte, razrede, atribute in relacije [18]. Projekt DBpedia je z vpeljavo podatkovne množice Ontology omogočil poizvedovanje objektov glede na razrede in podrazrede, ki jim pripadajo po hierarhični zgradbi.

3.1.2 Opis generirane podatkovne množice Ontologija

Z poizvedovalnim jezikom SPAQRQL so pridobljene entitete naslednjih devetnajst neprekrivajočih se razredov ontologije: Animal, Plant, City, WrittenWork, Film, MusicalWork, NaturalPlace, Building, Artist, Athlete, EducationalInstitution, Company, CelestialBody, Device, MeanOfTransportation, Infrastructure, TimePeriod, SocietalEvent in Village. Poleg imena entitete je bil pridobljen tudi povzetek v angleškem jeziku. Ta dva atributa sta združena in tvorita opis entitete, ki je klasificaran v določen razred. Vsi opisi razreda so shranjeni v posebni datoteki z imenom razreda, kar omogoča nadaljno delo s podatkovno množico. Število primerov posameznega razreda ponazarja tabela 3.1. Zaradi prevelike časovne zahtevnosti ne moremo uporabiti vseh primerov, zato uporabimo le naslednjih 10 razredov: Athlete, Animal,

MusicalWork, Village, Artist, Film, Infrastructure, Building, Company in NaturalPlace. Pri vsakem razredu naključno izberemo 3.000 primerov za učno množico in po 1.000 primerov za validacijsko in testno množico. Tako je učna množica velika 30.000 primerov, validacijska in testna množica pa sta veliki vsaka po 10.000 primerov. Tako kot se spreminja število primerov posameznega razreda, se spreminja tudi dolžina opisa posamezne entitete.

Tabela 3.1: Število primerov posameznih razredov, urejeno padajoče.

Razred	Število primerov
Athlete	268115
Animal	187587
MusicalWork	173612
Village	159993
Artist	95507
Film	86488
Infrastructure	71857
Building	67795
Company	63061
NaturalPlace	60120
WrittenWork	55173
Plant	50585
EducationalInstitution	50451
MeanOfTransportation	47472
SocietalEvent	46643
CelestialBody	22360
City	20891
Device	7695
TimePeriod	2073

Atributi so pridobljeni s spodnjo SPARQL poizvedbo:

```
PREFIX ont: <http://dbpedia.org/ontology/>
PREFIX dbpedia2: <http://dbpedia.org/property/>

SELECT ?entity ?abstract WHERE {
?entity a ont:<<ime razreda>>.
?entity ont:abstract ?abstract.
FILTER langMatches(lang(?abstract), 'en')
```

3.2 Vejice v slovenskem jeziku

Vejica je enodelno levostično ločilo, s katerim med seboj ločujemo stavke, pišemo vrinjene stavke ter naštevamo. Pravila za postavljanje vejic so navedena v Slovenskem pravopisu [16], da pa gre za kompleksen problem kaže tudi to, da Slovenski pravopis obravnava rabo vejic kar na sedmih straneh. S problemom postavljanja vejic se soočamo že od drugega razreda osnovne šole dalje, pa vendar nas ima večina večje ali manjše zadrege pri postavljanju vejic v slovenskem jeziku. Eden izmed razlogov te zadrege je, da lahko z nepravilno rabo vejice spremenimo pomen povedi, kar ima lahko usodne posledice. „*Rešiti ne obesiti*“ je primer dvoumne rabe vejice. Če vejica stoji pred členkom ne, je oseba prosta, če pa je vejica za členkom, je oseba obsojena na vešala.

Za postavljanje vejic v slovenskem jeziku obstajata dve orodji, to sta Besana [1] in LanguageTool [2].

Na to temo je bilo opravljeno že nekaj raziskav, tako je Anja Kranjc pripravila diplomsko nalogo z naslovom *Postavljanje vejic v slovenščini s pomočjo strojnega učenja* [10], kjer je z različnimi algoritmi strojnega učenja klasificirala vejice v slovenskem jeziku. Izhajala je iz pristopa Petra Holozana, ki je v članku *Kako dobro programi popravljajo vejice v slovenščini* [9] za klasifikacijo vejic v slovenskem jeziku uporabil strojno učenje. Tako je

generirala nove atribute iz že obstoječega korpusa Šolar1 in Šolar2 ter jih analizirala z algoritmom ReliefF [15].

Dosedaj poskusi niso bili opravljeni na konvolucijskih nevronskih mrežah, zato nas zanima, kako se naš model obnese pri tej nalogi z podatkovno množico Šolar2 in enakimi atributi.

3.2.1 Šolar2

Šolar2 je korpus namenjen pregledu pravilnosti pisanja vejic in ga lahko uporabimo za klasifikacijo vejic v slovenskem jeziku. Izhaja iz korpusa Šolar1, kjer so zbrana in urejena besedila, ki so nastala v okviru osnovnošolskega pouka. Šolar2 je posodobljena verzija, kjer je primerov z vejico veliko več, veliko stavkov pa je Peter Holozan ročno pregledal in popravil postavitev vejic. Korpus je oblikoskladenjsko označen, lematiziran z označevalnikom Obeliks [7] in skladenjsko razčlenjen s skladenjskim razčlenjevalnikom [6]. Vsebuje 728.927 primerov, od tega je 65.356 primerov z vejicami in 663.571 brez.

Z označevalnikom so bile pridobljene MSD oznake oziroma oblikoskladenjske oznake besed, kjer prva črka oznake predstavlja besedno vrsto, ostale pa njene lastnosti. Za generiranje atributov je uporabljeno okoliško okno in sicer trenutna beseda, pet besed pred trenutno besedo in pet besed za trenutno besedo. Vsega skupaj je torej 11 MSD oznak. Pri podatkovni množici Šolar2-MSD11 so uporabljene samo prve črke oznak, ki ponazarjajo besedno vrsto, pri množici Šolar2-MSD99 pa je iz vsake oznake ustvarjenih 9 atributov, kolikor je dolžina najdaljše MSD oznake, ta pripada zaimku. Tako iz enajstih atributov nastane 99 novih. 45 atributov je bilo pridobljenih z orodjem za postavljanje vejic LanguageTool. Za vsako besedo znotraj okna pa množica vsebuje tudi tri atribute za povezave trenutne besede. Podatkovna množica Šolar2-MSD11 vsebuje 90 atributov, množica Šolar2-MSD99 pa vsebuje vsega skupaj 187 atributov vključno z razredom (je-vejica, ni-vejice). Vsi atributi so podrobneje opisani v [10].

Poglavje 4

Realizacija

V tem poglavju razložimo opravljene poskuse. Najprej v razdelku *Predprocesiranje* 4.1 opišemo predprocesiranje tako za klasifikacijo ontologije kot za postavljanje vejic. V podpoglavju 4.2 je argumentirana izbira knjižnjice Theano za strojno učenje, na koncu pa je predstavljen klasifikacijski model z izbranimi parametri.

4.1 Predprocesiranje

Kjub temu, da lahko naš model sprejme golo besedilo in ga bolj ali manj ustrezno klasificira, je zaradi boljše klasifikacijske točnosti opravljenega tudi nekaj predprocesiranja. Uporabljene tehnike so opisane v naslednjih podpoglavjih.

4.1.1 DBpedia - ontologija

Vsa predprocesiranja so bila opravljeno s knjižnjico NLTK (Natural Language Toolkit), ki uporablja tudi svoje besedilne korpuse. Pri obdelavi naravnega jezika sta pogosto uporabljena dva postopka pridobivanja osnovne oblike besed. To sta lematizacija in krnjenje. Lematizacija oziroma geslenje je postopek, kjer besedi priredimo osnovno slovarsko obliko oziroma geslo. Na takšen način obdelano besedilo je primerno za vektorsko predstavitev be-

sedil (bag of words, bow). Preprostejši, a hitrejši postopek je krnjenje, kjer besedi odrežemo končnico z namero, da dobimo njen koren. Tako je lema besede *boljši* enaka *dober* in njen krn *bolj*. Na podatkovni množici Ontologije sta za primerjavo uporabljeni obe tehniki, kot tudi osnovno besedilo.

Za kvantizacijo besedila je uporabljena vektorizacija, kjer vsako črko zakodiramo v vektor s kodiranjem 1 do m. Gre za kodiranje, kjer z m biti zakodiramo podatek velik $log_2 m$ bitov. Tako recimo 0 v 1 do 3 kodiranju predstavlja vektor 001, vrednost 1 predstavlja vektor 010 in vrednost 2 predstavlja vektor 100. Kvantizacija je postopek, kjer poljubnemu številu priredimo diskretno vrednost. V našem primeru jo uporabimo za dvig dimenzije, kar omogoči večje konvolucijske filtre pri klasifikacijskem modelu.

Konkretno m = 45 predstavlja število uporabljenih znakov. Ti so: 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '.', '!', '?', ',', ';', '-', "'", '(' in ')'. Vsi ostali znaki vključno s presledki so zakodirani kot ničti vektorji. Ti vektorji tvorijo dvodimenzionalno vidno polje, ki služi kot vhod v naš model.

Za primerjavo naučimo konvolucijsko mrežo tudi na surovih podatkih, kjer je besedilo predstavljeno kot bitno polje in so uporabljeni samo znaki v ASCII kodiranju. Ker imamo enodimenzionalno predstavitev, so uporabljeni filtri primerne oblike.

Za primer vzemimo 1 do 5 kodiranje, kjer imamo na voljo le znake: 'a', 'b', 'k', 's' in 'u'. Z 1 do 5 kodiranjem kvantizirajmo besedo 'abakus'. Rezultat je matrika s petimi vrsticami (število znakov) in šestimi stolpci (dolžina niza):

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Konvolirajmo to matriko s filtrom:

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & -1 & -1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Rezultat je matrika s tremi stolpci in vrsticami:

$$\begin{bmatrix} 0 & 3 & 1 \\ 2 & -1 & -1 \\ 0 & 1 & 3 \end{bmatrix}$$

Če pa še z oknom s tremi vrsticami in stoplci posredujemo največjo vrednost dobimo:

$$\begin{bmatrix} 3 \end{bmatrix}$$

Opravimo še iste operacije kar na golih podatkih. Znaki so kodirani s kodno abecedo ASCII in predstavljeni kot bitno polje. Niz 'abakus' predstavlja naslednji vektor celih števil:

$$\begin{bmatrix} 97 & 98 & 97 & 107 & 117 & 115 \end{bmatrix}$$

Konvolirajmo še ta vektor s filtrom:

$$\begin{bmatrix} 1 & 0 & 2 & 0 \end{bmatrix}$$

Rezultat je vektor:

$$\begin{bmatrix} 291 & 312 & 331 \end{bmatrix}$$

Tudi tukaj lahko posredujemo maksimalno vrednost, le da ima okno le eno vrstico in tri stolpce. Rezultat je tako:

$$\begin{bmatrix} 331 \end{bmatrix}$$

Opazimo naslednje, ker so podatki v enodimenzionalni predstavitvi, sta takše oblike tudi filter in okno za združevanje maksimalne vrednosti, medtem ko so filtri pri kvantiziranih podatkih dvodimenzionalni. Ravno takšne tranformacije pa opravlja naša konvolucijska nevronska mreža, ki smo jo sprogramirali s knjižnjico Theano 4.2. S konvolucijskimi nevronskimi mrežami je mogoče natrenirati tudi podatkovne množice s klasičnimi atributi, če te na primeren način obdelamo. Obdelavo podatkovne mnižice Šolar2 opišemo v naslednjem razdelku 4.1.2.

4.1.2 Šolar2

Da bo podatkovna množica Šolar2 primerna za naš klasifikacijski model, moramo opraviti nekaj predprocesiranja. Konvolucijsko mrežo bi lahko trenirali kar na surovih podatkih, vendar bi s tem vnesli nepotrebne parametre in neskladnost atributov. To bi se zgodilo, ker lahko MSD kode zavzamejo različno število znakov. Tudi vejice, ki v datoteki ločujejo atribute, ne prinesejo nobene informacije, le to, da je po vejici naslednji atribut. Vejic ne potrebujemo, ker podatkovne množice Šolar2-MSD11 in Šolar2-MSD99 predelamo tako, da je vsak atribut velik le en znak, in jih zato odstranimo. Do dvoumnosti bi prišlo, če bi ne bilo mogoče zamenjati dvoznakovnih atributov z enim samim znakom.

V podatkovni množici Šolar2-MSD11 zamenjamo MSD kodi 'Vd' in 'Vp' z znakoma 'A' in 'B', ki v množici ne nastopata. Oznaki 'Vd' in 'Vp' opredeljujeta veznik, črki d in p pa povesta ali je veznik priredni oziroma podredni. Iz datoteke odstranimo tudi vse vejice in klasifikacijski razred z definicijskim območjem (ni-vejice, je-vejica), nadomestimo z vrednostima 0 in 1. Vsak primer z razredom tako vsebuje 90 atributov. Na koncu vsako klasifikacijo kvantiziramo z 1 do 18 kodiranjem. Uporabimo vse znake, ki se lahko pojavijo v podatkovni množici. Ti so: 'D','Z','S','G','A','B','P','L','Y','R','K','O','N','M', '*', '0', '1' in '-'. Vhod v klasifikacijski model je tako matrika z 18 vrsticami in 89 stolpci.

Podobno transformacijo opravimo tudi na podatkovni množici Šolar2-

MSD99, le da nam pri tej množici ni potrebno zamenjati vrednosti atributov, saj so že enoznakovni. Podatkovna množica vsebuje 178 atributov. Tudi tukaj opravimo kvantizacijo, le da uporabimo 1 do 35 kodiranje, saj se lahko v podatkovni množici pojavijo naslednji znaki: 'L', 'O', 'K', 'R', 'G', 'D', 'S', 'N', 'M', 'Z', 'P', 'V', 'Y', 'n', 'o', '0', '1', 'd', 'p', 'b', 'v', 't', '*', 'c', 's', 'a', 'e', 'i','g', 'z', 'l', 'r', 'm', 'k' in '-'. Vhod v klasifikacijski model je tako matrika z 35 vrsticami in 177 stolpci.

Pri obeh podatkovnih množicah naključno izberemo 100.000 negativnih primerov, kjer vejice ni in 10.000 pozitivnih, kjer vejica je. Učna množica ima tako 60.000 pozitivnih primerov in 6.000 negativnih in tako zavzema 66.000 primerov, medtem ko testna in validacijska množica zavzemata vsaka po 22.000 primerov od katerih je 2.000 pozitivnih. Na tak način obdržimo razmerje med pozitivnimi in negativnimi primeri, ki je 1:10.

4.2 Knjižnjica Theano

Konvolucijsko nevronsko mrežo smo sprogramirali s pomočjo knjižnjice Theano. Theano je knjižnjica za programski jezik python in je primerna za strojno učenje. Ena izmed funkcionalnosti je modul tensor, ki ga je mogoče uvoziti v python z izrazom *import theano.tensor as T*. Čeprav knjižnjica Theano podpira vse python objekte, ima Theano svoje podatkovne tipe, ki so namenjeni predvsem za simbolične matrične izraze. Izraz $x = T.fmatrix()$ je klic, ki inicializira spremenljivko tipa tensor. Tip spremenljivke x je znan, ker *x.type* kaže na *T.fmatrix*. Ker je *theano.function* vmesnik za *theano.compiler*, le-ta zgradi objekt iz simboličnega grafa. Tako prevajalnik optimizira definiran izraz, ne da bi pokvaril njegov semantični pomen [5]. Theano za računanje odvodov uporablja simbolično odvajanje [3], kar je prednost, saj to poenostavi program. Gradient $\frac{\partial f}{\partial x}$ dobimo s klicem $grad = T.grad(f,x)$, kjer je f simboličen izraz (funkcija) in x spremenljivka, po kateri odvajamo. Zaradi te lastnosti je za aktivacijsko funkcijo primernejši hiperbolični tanges kot sigmoida, saj je hitrejši. Theano funkcionalnost so tudi skupne spremen-

ljivke, ki so dostopne vsem Theano funkcijam v programu. Na takšen način Theano optimizira dostop do spremenljivk, saj so te spravljene v predpomnilniku. Tako deklarirane spremenljivke nam tudi olajšajo kodiranje, saj je koda krajša in preprostejša. Spodnja izvorna koda ponazarja sestavo konvolucijske nevronske mreže. Prikazana sta konstruktorja konvolucijskega in nevronskega nivoja, koda je tudi primerno komentirana.

```
#definicija razreda skriti nivo
class HiddenLayer(object):
 #konstruktor
 def __init__(self, rng, input, n_in, n_out, W=None, b=None,
  activation=T.tanh):

  #skriti nivo navadne nevronske mreze,
  #nevroni so polnopovezani

  #matrika utezi W je oblike
  #(stevilo vhodov, stevilo izhodov)
  #vsak stolpec predstavlja vse utezi vhodnega nevrona
  #do vseh izhodov

  #vektor pristranskosti je oblike (stevilo izhodov)

  #rng je numpy.random.RandomState
  #generator nakljucnih stevil
  #izhod skritega nivoja je tanh(dot(input,W) + b)

  self.input = input

  # W je inicializirana z nakljucnimi vrednostmi po
  #enakomerni porazdelitvi od
  #sqrt(-6./(n_in+n_hidden)) do sqrt(6./(n_in+n_hidden))

  if W is None:
   W_values = numpy.asarray(
    rng.uniform(
     low=-numpy.sqrt(6. / (n_in + n_out)),
```

```
            high=numpy.sqrt(6. / (n_in + n_out)),
            size=(n_in, n_out)
            ),
          dtype=theano.config.floatX)
        #inicializacija skupne spremenljivke W
        W = theano.shared(value=W_values, name='W', borrow=True)

        if b is None:
         #inicializacija nictega vektorja b
         b_values = numpy.zeros((n_out,),
          dtype=theano.config.floatX)
         b = theano.shared(value=b_values, name='b', borrow=True)

        self.W = W
        self.b = b

        #deklaracija izhoda nivoja
        lin_output = T.dot(input, self.W) + self.b
        self.output = (
         lin_output if activation is None
         else activation(lin_output)
        )
        #parametri modela, potrebni za theano funkcije
        self.params = [self.W, self.b]

#definicija razreda konvolucijski nivo
#z zdruzevanjem maks. vrednosti
class LeNetConvPoolLayer(object):
 #konstruktor
 def __init__(self, rng, input, filter_shape,
  image_shape, poolsize=(2, 2)):

  #parameter poolsize
  #velikost okna za zdruzevanje vrednosti
  #oblike (vrstice, stolpci)

  #parameter input
  #vhod v nivo, tipa theano.tensor.dtensor4
```

```
#mora veljati, drugace napaka
assert image_shape[1] == filter_shape[1]
self.input = input

#stevilo vhodov v skriti nivo je
#enako sirina filtra*visina filtra*
#velikost prve dimenzije vidnega polja, ta je
#enaka stevilu filtrov prejsnjega konv. nivoja
fan_in = numpy.prod(filter_shape[1:])
fan_out = (filter_shape[0] * numpy.prod(filter_shape[2:])
 numpy.prod(poolsize))
#inicializacija utezi
W_bound = numpy.sqrt(6. / (fan_in + fan_out))
self.W = theano.shared(
 numpy.asarray(
   rng.uniform(low=-W_bound, high=W_bound, size=filter_shape),
   dtype=theano.config.floatX
 ), borrow=True)

#vektor pristranskih vrednosti
#ena vrednost na filter
#velikost je enaka stevilu filtrov
b_values = numpy.zeros((filter_shape[0],),
 dtype=theano.config.floatX)
self.b = theano.shared(value=b_values, borrow=True)

#konvoliraj vhodno vidno polje z filtri
conv_out = conv.conv2d(
 input=input,
 filters=self.W,
 filter_shape=filter_shape,
 image_shape=image_shape
)

#zdruzi maksimalne vrednosti
pooled_out = downsample.max_pool_2d(
 input=conv_out,
```

```
    ds=poolsize,
    ignore_border=True
   )

  #dodaj vrednost pristranskosti in normaliziraj
  self.output = T.tanh(pooled_out +
    self.b.dimshuffle('x', 0, 'x', 'x'))

  self.params = [self.W, self.b]
```

4.3 Klasifikacijski model

Ker je klasifikacijski model pri konvolucijskih nevronskih mrežah odvisen od oblike podatkov, smo ustvarili več klasifikacijskih modelov. Dva modela smo ustvarili za množici Šolar2-MSD11 in Šolar2-MSD99, ter dva modela za podatkovno množico pridobljene z DBpedije.

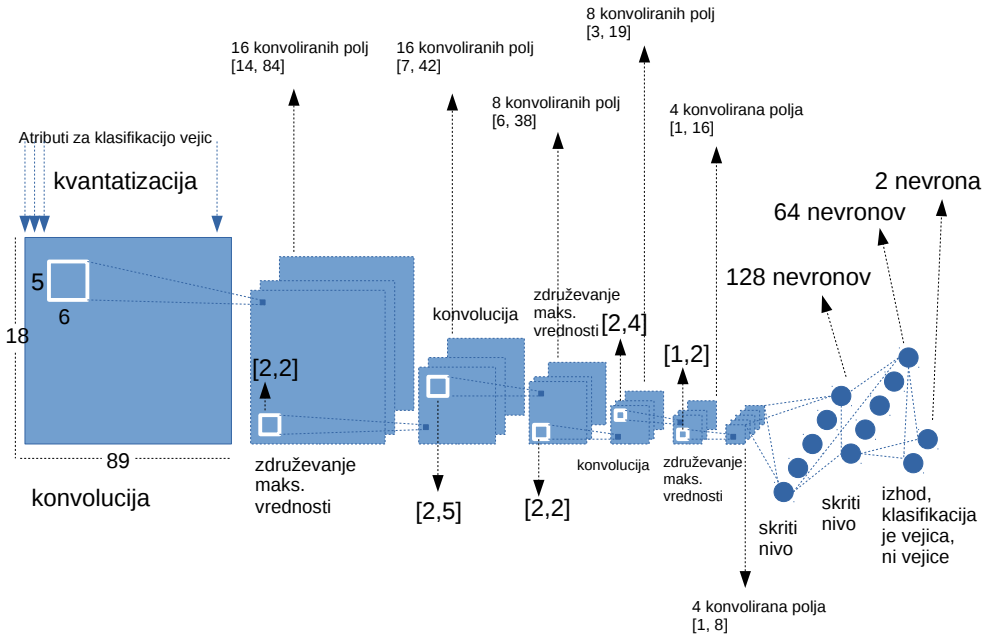
Za poskuse na množici pridobljeni iz DBpedije je naša nevronska mreža sestavljena iz štirih kovolucijskih slojev in treh polnopovezanih nevronskih slojev. Ustvarjena sta bila dva modela, kjer je eden primeren za vektorizacijo, temu primerne so velikosti konvolucijskih filtrov, in drugi za učenje na navadnem besedilu, tj. bitno polje. Oba modela sestavlja dvaintrideset filtrov na prvem in drugem konvolucijskem nivoju in šestnajst na tretjem in četrtem. Izhod iz zadnjega konvucijskega nivoja je transformiran tako, da je kompatibilen z vhodom prvega polnopovezanega nevronskega sloja. Izhod prvega navadnega nevronskega nivoja je velikosti 128, izhod drugega navadnega nivoja je 64 in tretjega 10, kolikor je razredov. Maksimalne vrednosti zdužujemo z oknom velikosti [2,2], tj. matrika z dvema vrsticama in dvema stolpcema. Filtri prvega konvolucijskega nivoja pri modelu za kvantizirane podatke so oblike [6,7], drugega [7,7], tretjega [4,4] in četrtega [2,4]. Tudi filtri modela primernega za podatke kodirane v ASCII abecedi so enake oblike, le da imajo ti filtri le eno vrstico in enako število stolpcev.

Pri postavljanju vejic smo zmanjšali število nivojev nevronske mreže. Ta

vsebuje tri konvolucijske in tri polnopovezane nevronske nivoje. Prvi konvolucijski nivo vsebuje 16 filtrov, drugi 8 in tretji 4. Izhod prvega navadnega nevronskega nivoja je velikosti 128 in drugega 64. Ker imamo binarni klasifikator (je vejica, ni vejice) ima zato izhodni nivo le dva nevrona. Tudi tukaj združujemo maksimalne vrednosti z oknom velikosti [2,2]. Model primeren za učenje na podatkovni množici Šolar2-MSD11 ima filtre prvega nivoja oblike [5,6], drugega [2,5] in tretjega [2,4]. Model primeren za trening na podatkovni množici Šolar2-MSD99 pa ima filtre prvega nivoja oblike [6,6], drugega [6,7] in tretjega [4,5]. Slika 4.1 prikazuje klasifikacijski model uporabljen za trening na podatkovni množici Šolar2-MSD11.

Takšno obliko filtrov izberemo, ker želimo dovolj velike filtre, da lahko ti zavzamejo pričakovano število znakov v besedah, ki se v učni množici pojavljajo z visoko frekvenco. Z večanjem filtrov povečujemo čas procesiranja, zato iščemo kompromis. Ker konvolirana vidna polja še pomanjšamo z metodo združevanja maksimalnih vrednosti, morajo biti filtri taki, da bo velikost izhoda iz konvolucije sodo število. Le tako lahko združimo maksimalne vrednosti z oknom oblike [2,2].

Takšno obliko modela smo izbrali zato, ker želimo čim globje arhitekture, saj z vpeljavo novih nivojev zmanjšamo potrebno število nevronov oziroma povečamo klasifikacijsko točnost [4]. Hkrati pa je potrebno, da model natreniramo v nekem sprejemljivem času. Zmanjšali smo število nivojev pri problemu postavljanja vejic, saj gre za binaren klasifikator. Število filtrov zmanjšujemo s faktorjem dve zaradi abstrakcije, ki jo želimo priučiti v našo konvolucijsko nevronsko mrežo. Zamislimo si naslednji primer. Gledamo sliko otroka, ki se igra z žogo. Samoumevno je, da je na sliki otrok, ki ga igra z žogo razveseljuje. Na takšen način bi tudi drugi osebi to sliko opisali. Ne bi se spuščali v senčenje, strukturo ozadja ali kompozicijo elementov. Gre za primer abstrakcije, ki smo je ljudje zmnožni, saj iz velike množice podatkov izluščimo bistveno informacijo. Prav tako konvolucijska nevronska mreža preko konvolucijskih nivojev združuje oziroma sešteva podatke, s čimer je vsak nivo abstraktnejši. Filtri na prvih konvolucijskih nivojih zdru-



Slika 4.1: Model globoke konvolucijske nevronske mreže primeren za podatkovno množico Šolar2-MSD11.

žujejo črke v besede, pri čemer posredujejo večjo vrednost pri besedah, ki se v učni množici večkrat pojavijo. Za primer vzemimo besedo „plant“, ta se mnogokrat pojavi v razredu Ontologije Plant. Filtri na višjem nivoju združujejo besede v stavke, tako bi lahko filtri posredovali velike vrednosti, če bi prejšnji nivoji že konvolirali besedno zvezo „plant grows“, filtri na zadnjih nivojih pa na podoben način združujejo besedne zveze v povedi. Za izbiro prostih parametrov smo se zgledovali predvsem po pristopih, ki so že uporabljeni v umetnem zaznavanju, saj za klasifikacijo teksta s konvolucijskimi nevronskimi mrežami še ni bilo opravljenih toliko raziskav.

4.4 Poskusi

Opravili smo 8 testov na učni množici DBpedije s 30.000 primeri. Ovrednotenje je bilo opravljeno na 10.000 primerih, prav tako testiranje. Število znakov kot vhod v model je 150. Besedilo je enkrat vektorizirano in drugič predstavljeno kot bitno polje, kot smo opisali v razdelku 4. Zanima nas tudi primerjava krnjenja in lematizaciije. Ker je pri konvolucijski nevronski mreži še mnogo drugih parametrov, kot sta *stopnja učenja* in *stopnja momenta*, je vsak test pognan dvakrat in sicer *stopnja učenja = 0.08, stopnja momenta = 0.004* ter *stopnja učenja = 0.13, stopnja momenta = 0.04*.

Opravili smo tudi dva testa na korpusu Šolar2. Enega na podatkovni množici Šolar2-MSD11 in drugega na podatkovni množici Šolar2-MSD99. Za stopnjo učenja smo izbrali vrednost 0.13 in za stopnjo momenta vrednost 0.04.

Čez učno množico iteriramo stokrat. Pri vsaki iteraciji ovrednotimo klasifikacijsko točnost na validacijski množici, ter si zapomnimo najboljši klasifikator. Na koncu ta klasifikator poženemo na testni množici, kar nam poda oceno za klasifikacijsko točnost klasifikatorja.

Poglavje 5

Rezultati

V tem poglavju opišemo in ovrednotimo rezultate testov. Najprej so v razdelku 5.1 predstvaljeni rezultati testov, ki smo jih opravili nad podatkovno množico DBpedie, nato v 5.2 predstavimo še rezultate za podatkovno množico Šolar2.

Opišimo še mere, ki jih uporabljamo za ocenjevanje kakovosti klasifikacijskih modelov. Klasifikacijska točnost (classification accuracy) je točnost, s katero klasifikator napoveduje oziroma klasificira primere v razrede (5.1). Točnost (precision) je vrednost, ki nam pove kolikšen delež klasifikacij za razred je pravilnih (5.2). Priklic (recall) je vrednost, ki izmed vseh pozitivnih primerov razreda, predstavlja delež pravilnih klasifikacij (5.3). Zamislimo si naslednji scenarij. Smo v knjižnici in čakamo, da nam knjižničar prinese vse knjige o programiranju, ki so na voljo. Ker knjižničar ni vešč programiranja, nekaj knjig izpusti. Med tistimi, ki jih prinese, so tudi takšne, ki niso s pravega področja. Točnost nam pove delež bistvenih knjig izmed vseh prinešenih. Priklic nam pove delež bistvenih knjig izmed vseh knjig o programiranju. Mero F_1 si lahko predstavljamo kot uteženo razmerje med točnostjo in priklicom (5.4).

Legenda:

CA ... klasifikacijska točnost

P ... točnost

R ... priklic

F_1 ... mera F_1

TP ... število pravilno napovedanih primerov pozitivnega razreda

TN ... številno pravilno napovedanih primerov negativnega razreda

FN ... število napačno napovedanih primerov pozitivnega razreda

FP ... število napačno napovedanih primerov negativnega razreda

$$CA = \frac{TP + TN}{TP + FN + FP + TN} \tag{5.1}$$

$$P = \frac{TP}{TP + FP} \tag{5.2}$$

$$R = \frac{TP}{TP + FN} \tag{5.3}$$

$$F_1 = \frac{2 * P * R}{P + R} \tag{5.4}$$

5.1 Rezultati DBpedije

Poglejmo si rezultate poskusov, ki so bili opravljeni na podatkovni množici pridobljeni z DBpedie.

Za stopnja učenja = 0.08 in stopnja momenta = 0.004 so v tabeli 5.1 prikazane točnosti, ki jih dobimo. Opazimo, da rezultati niso najboljši. Sklepamo lahko, da zaradi nizke stopnje momenta konvolucijska nevronska mreža, s takšno izbiro stopnje učenja, obtiči v lokalnem minimumu. Če bi klasifikator vedno napovedoval samo en razred izmed vseh desetih, bi dosegel 10% klasifikacijsko točnost. Naš model je nekoliko boljši, a vseeno neuporaben. Na točnost vpliva tudi način, kako podatke obdelamo, preden jih model sprejme. Model dosega nekoliko boljše rezultate, če se uči na kvantiziranih podatkih,

kjer je vsak znak vektoriziran z 1 do m kodiranjem. To pomeni, da dvodimenzionalna konvolucija prekaša enodimenzionalno, če imata oba modela enako globino konvolucijskih in skritih nivojev. Razlog za to bi lahko bilo večje število uteži pri modelu naučenem na kvantiziranih podatkih na konvolucijskih nivojih. Če ima model, ki se uči na besedilu predstavljenem kot bitno polje, filter velikosti n, mora biti filter modela, ki se uči na kvantiziranih podatkih velikosti n^2, kar pomeni večje število uteži. Kljub slabi klasifikacijski točnosti si poglejmo še primerjavo lematizacije s krnjenjem. Če učimo klasifikacijski model na surovem besedilu, sta si rezultata podobna, nekoliko bolje se odreže klasifikator, kjer besede samo krnimo, pri kvantiziranih podatkih pa klasifikator pri lematizirani učni množici dosega še nekoliko boljši rezultat.

Poglejmo še klasifikacijsko točnost klasifikatorja, kjer smo izbrali nekoliko večjo vrednost za stopnjo učenja, ta je 0.13 in veliko večjo stopnjo momenta, ki je 0.04. Podatki so prikazani v tabeli 5.2. Pri takšni izbiri parametrov so rezultati boljši. Vidimo, da se klasifikacijska točnost spremeni predvsem po zaslugi večje stopnje momenta. Tudi tukaj dobimo slabše rezultate, če klasifikacijski model učimo na besedilu predstavljenem kot bitno polje, kar potrjuje našo domnevo, da dvodimenzionalna konvolucija prekaša enodimenzionalno, saj je število uteži pri dvodimenzionalni konvoluciji eksponentno večje. Obetavnejša je klasifikacijska točnost pri kvantiziranih podatkih: 82.90% točnost dobimo s krnjenjem in 80.90% z lematizacijo. Sklepali bi lahko, da lematizacija prekaša krnjenje, vendar temu ni tako. Čeprav lematizacija priredi različnim izpeljankam isto lemo in s tem zmanjšuje dvoumnost v podatkih, gre pri krnjenju za to, da skrajšamo besedo in ohranimo samo krn. Če so besede krajše, jih lahko model, ki sprejema vnaprej določeno velikost podatkov, zajame več.

Poglejmo si še matriko zmot (confusion matrix) za klasifikacijski model, ki se je odrezal najbolje. V tabeli 5.3 opazimo, da klasifikacijski model v vse razrede klasificira enakomerno, saj so odstotki po diagonali veliko večjih od ostalih. K temu je pripomoglo to, da smo imeli uteženo število primerov. V

vsakem razredu jih je 10.000.

Če primerjamo klasifikacijsko točnost z rezultati, ki so navedeni v [21], opazimo, da so rezultati naših poskusov nekoliko slabši. Vendar pa je to razumljivo, saj vsebuje učna množica v [21] 400.000 primerov. Poleg tega konvolucijske nevronske mreže v [21] sestojijo iz več konvolucijskih nivojev, oziroma so globje.

Tabela 5.1: Klasifikacijska točnost za stopnja učenja = 0.08 in stopnja momenta = 0.004.

klasifikacijska točnost (%)	krnjenje	lematizacija
bitno polje	15.08%	13.83%
kvantizacija	32.14%	49.51%

Tabela 5.2: Klasifikacijska točnost za stopnja učenja = 0.13 in stopnja momenta = 0.04.

klasifikacijska točnost (%)	krnjenje	lematizacija
bitno polje	20.79%	46.33%
kvantizacija	82.90%	80.90%

5.2 Rezultati množice Šolar2

Poglejmo si še, kako dobro zna naš klasifikacijski model postavljati vejice v slovenskem jeziku. Rezultate prikazuje tabela 5.4.

Model, ki se je učil postavljati vejice na podatkovni množici Šolar2-MSD11 doseže 95.43% klasifikacijsko točnost. Če pogledamo točnost za primer, ko vejica nastopa, je ta dokaj visoka, znaša 91.55%. Priklic znaša 54.77%. To pomeni, da je naš klasifikator dokaj dober v postavljanju vejic. Če postavi vejico je verjetnost 91.55%, da tam vejica stoji, vendar veliko vejic

Tabela 5.3: Matrika zmot za vektorizacijo s krnjenjem na podatkovni množici DBpedije. Podatki so zaokroženi na dve decimalni mesti.

%	*Athlete*	*Animal*	*MusicalWork*	*Village*	*Artist*	*Film*	*Infrastructure*	*Building*	*Company*	*NaturalPlace*
Athlete	7.25	0.18	0.19	0.15	1.19	0.16	0.08	0.1	0.28	0.11
Animal	0.11	7.38	0.10	0.09	0.15	0.20	0.23	0.43	0.30	0.21
MusicalWork	0.22	0.35	6.85	0.09	0.21	0.65	0.18	0.51	0.42	0.30
Village	0.14	0.12	0.09	8.93	0.11	0.12	0.11	0.20	0.11	0.32
Artist	0.9	0.31	0.07	0.14	6.05	0.25	0.19	0.35	0.85	0.49
Film	0.11	0.33	0.22	0.12	0.22	8.32	0.20	0.32	0.23	0.18
Infrastructure	0.11	0.18	0.19	0.15	0.1	0.22	7.75	0.91	0.34	0.65
Building	0.07	0.32	0.21	0.29	0.24	0.19	0.48	6.73	0.78	0.86
Company	0.25	0.66	0.20	0.18	0.46	0.19	0.38	1.01	6.76	0.57
NaturalPlace	0.09	0.16	0.18	0.27	0.22	0.14	0.22	0.47	0.24	7.80

zgreši, skoraj polovico. Temu primerna je tudi F_1 ocena, ki je 68.54%. Poglejmo še primer, ko vejica ne nastopa. Tukaj so točnost, priklic in mera F_1 pričakovano višji. Ker je v podatkovni množici Šolar2-MSD11 desetkrat več primerov brez vejic, kot tistih kjer vejica nastopa, sta zato točnost v primeru, da vejica nastopa in klasifikacijska točnost podobni. Visoka sta tudi priklic in mera F_1. Točnost znaša 95.66%, priklic 99.50% ter mera F_1 97.54%.

Podobne rezultate smo dobili, kjer smo model učili na podatkovni množici Šolar2-MSD99. Osredotočimo se na primer ko vejica nastopa. Točnost v primeru vejice je nižja od točnosti, ki jo dosega model, ki se je učil na množici Šolar2-MSD11, vendar ima ta klasifikator zato večji priklic. To pomeni, da postavlja klasifikator, naučen na podatkovni množici Šolar2-MSD99, vejice s točnostjo 81.36%. Delež vejic, ki jih klasifikator ne zgreši znaša 65.59%.

Rezultati so primerljivi z diplomsko nalogo [10], kljub temu, da konvolucijske nevronske mreže niso namenjene podatkovnim množicam z atributi, saj filter drsi po vidnem polju in posreduje vrednosti, ki jih pridobi s kon-

volucijo. To pomeni, da obravnavajo konvolucijski nivoji vse atribute enakovredno. Konvolucijska nevronska mreža, ki smo jo učili na takšni množici, dosega solidno klasifikacijsko točnost. S tem smo se prepričali, da jih je mogoče uporabiti v ta namen. Dobro klasifikacijsko točnost dobimo, ker je konvolucijska nevronska mreža s pomočjo konvolucije sposobna detektirati vzorce, ki se v učni množici večkrat pojavijo. Konkretno gre za zaporedje vrednosti atributov, ki nakazujejo, da vejica stoji. Drugačno vlogo imajo polnopovezani skriti nivoji, katerih uteži niso deljene in ima zato vsaka utež svoj pomen. To pomeni, da ti nivoji razlikujejo ali se je takšna detekcija zgodila nad trenutno besedo ali nad katero drugo besedo v okoliškem oknu ter temu ustrezno posredujejo vrednosti izhodnemu nivoju, ki s 95% natančnostjo napove ali v besedilu stoji vejica ali ne.

Tabela 5.4: Klasifikacijska točnost s točnostjo, priklicom ter mero F_1 za primer ko vejica je in ko vejice ni.

	klas.	je vejica			ni vejice		
	točnost	točnost	priklic	F_1	točnost	priklic	F_1
Šolar2-MSD11	95.43%	91.55%	54.77%	68.54%	95.66%	99.50%	97.54%
Šolar2-MSD99	95.54%	81.36%	65.59%	72.63%	96.64%	98.51%	97.57%

Poglavje 6

Zaključek

Preizkusili smo, kako se konvolucijske nevronske mreže izkažejo pri učenju jezikovnih problemov za razumevanje besedila in za atributni opis. Opisali smo arhitekturo in delovanje konvolucijskih nevronskih mrež, kako se učijo in kako napovedujejo. Osredotočili smo se predvsem na konvolucijo, zakaj je koristna in kako jo uporabiti v nevronski mreži. Opisali smo podatkovno množico pridobljeno iz DBpedie in kako smo jo pridobili ter podatkovno množico Šolar2 in iz te množice izpeljani množici Šolar2-MSD11 in Šolar2-MSD99. Čeprav lahko konvolucijske nevronske mreže učimo na golem besedilu, smo opravili tudi nekaj predprocesiranja. Za podatkovno množico pridobljeno z DBpedie smo besedilo kvantizirali tako, da smo znake vektorizirali z 1 do m kodiranjem. Tekst smo za primerjavo pustili tudi v surovi obliki in na njem naučili konvolucijsko nevronsko mrežo. Zanimala nas je primerjava lematizacije in krnjenja, zato smo besede lematizirali in krnili. Prilagodili smo podatkovni množici Šolar2-MSD11 in Šolar2-MSD99, da sta bili primerni za naš model. Zanimalo nas je, kako na klasifikacijsko točnost vpliva izbira drugačne vrednosti stopnje učenja in stopnje momenta.

Ugotovili smo, da lahko konvolucijske nevronske mreže uspešno natreniramo na besedilnih korpusih. Čeprav so bile razvite za umetno zaznavanje, so primerne tudi za strojno učenje na področju jezika. Ugotovili smo, da enodimenzionalna konvolucija ni primerna za predstavitev besedila, saj se je bil

naš klasifikator nezmožen naučiti filtrov, da bi ti lahko detektirali besede in besedne zveze. Pokazali smo, da so konvolucijske nevronske mreže primerne za učenje postavljanja vejic in da so se sposobne učiti na atributih.

V nadaljnem delu bi bilo potrebno opraviti več poizkusov na še nekoliko globjih konvolucijskih mrežah, da bi lahko trdili, da enodimenzionalna konvolucija ni primerna (pričakujemo, da bi bila). Velik problem konvolucijskih nevronskih mrež je veliko število parametrov, ki jih moramo ustrezno nastaviti, in ki lahko že pri majhnih spremembah močno spremenijo uspešnost učenja. Ker je s konvolucijskimi nevronskimi mrežami opravljeno le malo raziskav s področja obdelave naravnega jezika in mnogo na področju umetnega zaznavanja, smo se po izbiri vrednosti za parametre zgledovali po slednjih. Poleg tega je potrebno konvolucijske nevronske mreže dolgo časa učiti, saj s tem, ko nalagamo nivo na nivo, vedno težje učimo nižje nivoje. Trdimo lahko, da so konvolucijske nevronske mreže dokaj časovno zahtevne. Zanimivo bi bilo raziskati, kako se izkaže učenje mrež, če prej nenadzorovano z Restricted Boltzman Machine nastavimo filtre prvih konvolucijskih nivojev in ali to pripomore k zmanjšanju času učenja in klasifikacijski točnosti. Če so se konvolucijske nevronske mreže izkazale v obdelavi naravnega jezika, bi bilo zanimivo videti tudi, kako se izkažejo v glasbi.

Literatura

[1] Besana - slovnični pregledovalnik. `http://besana.amebis.si/`. Dostop: 01-08-2015.

[2] Skupnost LanguageTool. Orodje LanguageTool. `http://community.languagetool.org`. Dostop: 01-08-2015.

[3] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, in Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning, NIPS 2012 Workshop, 2012.

[4] Yoshua Bengio. Learning deep architectures for AI. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[5] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, in Yoshua Bengio. Theano: a CPU and GPU Math Expression Compiler. V *Proceedings of the Python for Scientific Computing Conference (SciPy)*, Junij 2010.

[6] Kaja Dobrovoljc, Simon Krek, in Jan Rupnik. Skladenjski razčlenjevalnik za slovenščino. V *Zbornik Osme konference Jezikovne tehnologije, Institut Jožef Stefan, Ljubljana*, 2012.

[7] Miha Grcar, Simon Krek, in Kaja Dobrovoljc. Obeliks: statisticni oblikoskladenjski oznacevalnik in lematizator za slovenski jezik. V *Zbornik Osme konference Jezikovne tehnologije, Ljubljana, Slovenia*, 2012.

[8] Donald O Hebb. The organization of behavior; a neuropsychological theory. 1949.

[9] Peter Holozan. Kako dobro programi popravljajo vejice v slovenščini. V *Zbornik Osme konference Jezikovne tehnologije, Ljubljana, Slovenia*, 2012.

[10] Anja Krajnc. Postavljanje vejic v slovenščini s pomočjo strojnega učenja. Diplomsko delo, Fakulteta za računalništvo in informatiko, 2015.

[11] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, in Andrew D Back. Face recognition: A convolutional neural-network approach. *IEEE Transactions on Neural Networks*, 8(1):98–113, 1997.

[12] Yann LeCun, Corinna Cortes, in Christopher JC Burges. The MNIST database of handwritten digits, 1998.

[13] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, et al. DBpedia-a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*, 5:1–29, 2014.

[14] Jonathan Robinson in Vojislav Kecman. Combining support vector machine learning with the discrete cosine transform in image compression. *Neural Networks, IEEE Transactions on*, 14(4):950–958, 2003.

[15] Marko Robnik-Šikonja in Igor Kononenko. Theoretical and empirical analysis of ReliefF and RReliefF. *Machine learning*, 53(1-2):23–69, 2003.

[16] Jože Toporišič. *Slovenski pravopis*. Slovenska akademija znanosti in umetnosti, 2001.

[17] Alan M Turing. Computing machinery and intelligence. *Mind*, strani 433–460, 1950.

[18] Wikipedia. Ontologija (informatika) — Wikipedia, The Free Encyclopedia. `https://sl.wikipedia.org/wiki/Ontologija_(informatika)`, 2015. Dostop: 30-7-2015.

[19] Wikipedia. Resource Description Framework — Wikipedia, The Free Encyclopedia. `https://en.wikipedia.org/wiki/Resource_Description_Framework`, 2015. Dostop 30-7-2015.

[20] Terry Winograd. What does it mean to understand language? *Cognitive science*, 4(3):209–241, 1980.

[21] Xiang Zhang in Yann LeCun. Text Understanding from Scratch. *CoRR*, abs/1502.01710, 2015. Dostopno na `http://arxiv.org/abs/1502.01710`.