Modeliranje reprezentacije riječi i njihovih međuovisnosti tehnikama strojnog učenja

Nikić, Patrick

Sveučilište J.J.Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni diplomski studij matematike

Patrick Nikić

Modeliranje reprezentacije riječi i njihovih međuovisnosti tehnikama strojnog učenja

Diplomski rad

Osijek, 2020.

Sveučilište J.J.Strossmayera u Osijeku
Odjel za matematiku
Sveučilišni diplomski studij matematike

Patrick Nikić

Modeliranje reprezentacije riječi i njihovih međuovisnosti tehnikama strojnog učenja

Diplomski rad

Voditelj: doc.dr.sc. Domagoj Ševerdija

Osijek, 2020.

Sadržaj

Uvod

Umjetne neuronske mreže su model računanja inspiriran radom bioloških neurona u procesu obrade informacija u ljudskom mozgu. Potaknule su velike pomake u znanosti i industriji, zahvaljujući zadivljujućim rezultatima u prepoznavanju govora, računalnom vidu i obradi teksta. Iako matematika upletena u neuronske mreže nije trivijalna stvar, čovjek može relativno lako steći operativno znanje o njihovoj strukturi i djelovanju.

U prvom poglavlju će biti objašnjeni osnovni pojmovi vezani uz neuronske mreže i optimizaciju. Ideja je bila dati kratak, ali što potpuniji uvid u proces definiranja neuronske mreže, učenja mreže te opisati tehnike i algoritme koje se koriste u pozadini.

U idućem poglavlju ćemo se fokusirati na dio strojnog učenja koji proučava interakciju između računala i ljudskog govora (jezika). Ljudski jezik je vrlo efektan u opisu našeg odnosa sa stvarima koje nas okružuju. Sa samo nekoliko riječi možemo prenijeti neku ideju ili zamisao, najčešće bez dvosmislenosti. Naš jezik je složen i implicitno vrlo strukturiran. Stoga, nije jednostavan zadatak za računalo naučiti prirodni jezik. Dio znanosti koji se time bavi se zove obrada prirodnog jezika (eng. *natural language processing (NLP)*). U obradi prirodnog jezika, cilj je naučiti računalo obradu i analizu velikih količina jezičnih podataka.

Kako bismo razumjeli jezične podatke, najprije moramo shvatiti riječi. Nije moguće usporediti dvije riječi ako ne poznajemo njihovo značenje. Zbog toga, svakoj riječi želimo pridružiti značenje koje ona ima u odnosu na druge riječi. Kako su ulazi za probleme u strojnom učenju isključivo brojevi (vektori brojeva), pojasnit ćemo kako se od se riječi mogu dobiti vektori koji obuhvaćaju raznolika značenja istih. Pomoću vektora riječi, možemo rješavati mnoge zadatke u obradi prirodnog jezika.

U trećem poglavlju slijedi jedna primjena vektora riječi: parsiranje međuovisnosti među riječima u rečenici. Zadatak je analizirati strukturu rečenice i prepoznati nadređene riječi u rečenici, te njima podređene riječi koje o njima ovise, mijenjaju ih ili nadopunjuju. Opisat ćemo neuronsku mrežu koja se koristi za treniranje parsera međuovisnosti. Mrežu smo trenirali i testirali na jednom hrvatskom korpusu te smo analizirali dobivene rezultate.

U praktičnom dijelu korišteni su vektori riječi izrađeni word2vec modelom (vidi [8]) iz hrvatskog CoNLL (The Conference on Computational Natural Language Learning) korpusa[1]. Slike u radu su izrađene koristeći alate GeoGebra[2] i GNU Image Manipulation Program[3].

Istaknuti sadržaj

U pravokutnicima poput ovoga je sadržaj koji daje dodatni kontekst ili informacije, a nije ključan za razumijevanje teksta.

[1] Vektori riječi mogu se besplatno preuzeti na `http://vectors.nlpl.eu/repository/` zahvaljujući Language Technology Group sa Sveučilišta u Oslu.

[2] Službena stranica: `http://www.geogebra.org/`

[3] Službena stranica: `http://gimp.org/`

1 Umjetne neuronske mreže

Ptice su nas potaknule na let, a možda je slonova surla nadahnuće za izradu robotskih ruku. Priroda je inspirirala mnoge ljudske izume. Tako je i arhitektura ljudskog mozga inspiracija za izradu pametnog stroja što je motiviralo izum *umjetnih neuronskih mreža (UNM)*[4]. Ipak, kao što ni avioni ne mašu krilima, tako ni umjetni neuroni ne rade poput bioloških.

UNM su sama srž za duboko učenje. One su snažan alat koji se može prilagoditi velikim količinama podataka u vrlo različitim područjima. Primjenjuju se u obradi slike (prepoznavanje lica, znakova, objekata), prepoznavanju govora, strojnom prevođenju tekstova, medicinskoj dijagnostici, vožnji autonomnih vozila itd.

Ovo poglavlje daje općeniti uvod u neuronske mreže i uvid u cijeli proces učenja mreže te završava s nekoliko praktičnih savjeta i tehnika koje se često koriste u praksi.

1.1 Neuron

Umjetni neuron je osnovna građevna jedinica u neuronskim mrežama. Riječ je o funkciji koja prima n ulaza i vraća jednu izlaznu vrijednost (aktivacija). Za svaku ulaznu vrijednost x_i, neuron ima pripadajuću težinu w_i, te ima i jedan dodatni parametar b (eng. *bias*). Najprije računa težinsku sumu ulaznih podataka i na nju primjeni neku aktivacijsku funkciju f.

$$z = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b \tag{1.1}$$

$$a = f(z) \tag{1.2}$$

Umjetni neuron možemo shvatiti kao metodu kojom ćemo odvagnuti (w_i) informacije (x_i) za donošenje odluke (a). Parametar b nam kazuje s kolikom lakoćom neuron vraća 1. S velikom vrijednosti b, neuron će lako vratiti 1, dok će za negativnu vrijednost b, neuron teže vratiti 1.

Aktivacijske funkcije

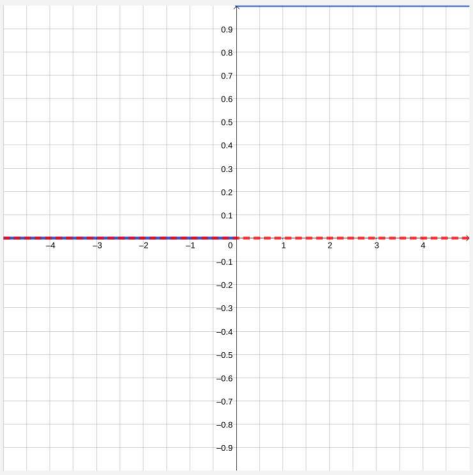
Pri izgradnji različitih neuronskih mreža, neke aktivacijske funkcije daju bolje rezultate od drugih. Ovdje ćemo navesti neke koje se najčešće koriste.

Heaviside step funkcija je dana s:

$$H(z) = \begin{cases} 0, & z < 0 \\ 1, & z \geq 0 \end{cases}$$

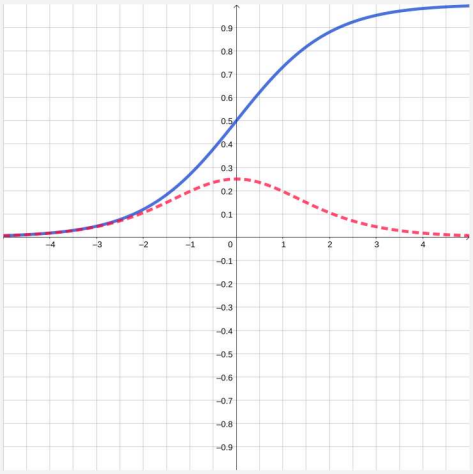
$$H'(z) = 0$$

Funkcija je vrlo jednostavna i koristi se ako želimo binarni izlaz (0 ili 1). Korisna je najviše zbog teorijskih analiza i u eksperimentalnim mrežama. Njen ključni problem je što je po dijelovima konstantna pa se s njenom derivacijom ništa ne može. Naime, optimizacijske metode poput gradijentnog spusta trebaju ne-nul derivaciju kako bi pronašle minimum / maksimum.



Slika 1.1: Step funkcija i derivacija

[4] eng. *artificial neural network (ANN)*



Slika 1.2: Sigmoid funkcija i derivacija

Sigmoid je čest izbor aktivacijske funkcije, naročito kod klasifikacije, jer vraća vrijednost između 0 i 1. Dana je s:

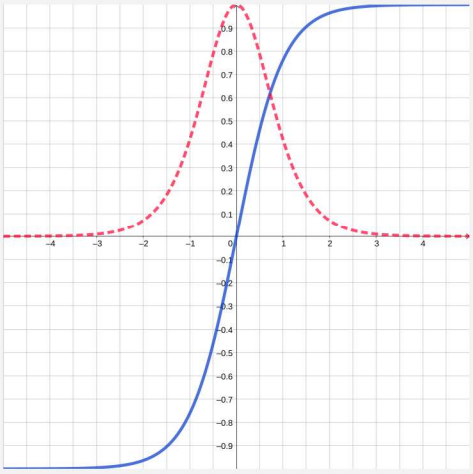
$$\sigma(z) = \frac{1}{1+\exp(-z)}$$
$$\sigma'(z) = \frac{-\exp(-z)}{1+\exp(-z)} = \sigma(z)(1-\sigma(z))$$

U praksi, ova funkcija se susreće s problemom nestajućeg gradijenta. Za velike vrijednosti $|z|$ će gradijent biti blizu nule, ažuriranje parametara u mreži tijekom učenja će biti neznatno i učenje jako sporo. To se nastoji izbjeći u dubokom učenju jer su izračuni ionako skupi i dugotrajni. Također, treba paziti kod inicijalizacije težina sigmoid neurona, jer će prevelike težine odmah zasititi neuron i mreža će jedva što naučiti. Dodatna neželjenost jest što izlaz ove funkcije nije centriran oko nule, što može stvarati poteškoće kasnijim slojevima u mreži.

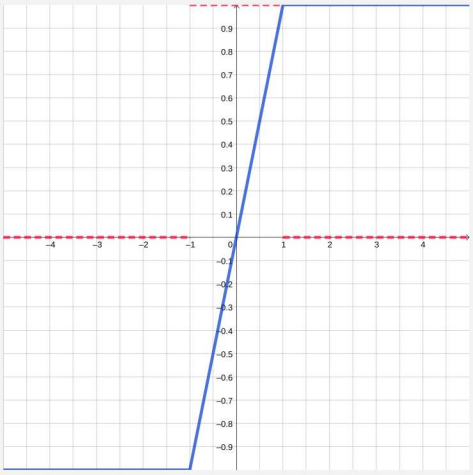
Tanh je skalirana inačica sigmoida, dana je s:

$$\tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)} = 2\sigma(2z) - 1$$
$$\tanh'(z) = 1 - \left(\frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}\right)^2 = 1 - \tanh^2(z)$$

Ova funkcija se, također, susreće s problemom nestajućeg gradijenta. Glavna razlika je što vraća vrijednost između -1 i 1, pa su vrijednosti izlaza tog sloja mreže centrirane oko nule. To doprinosi bržoj konvergenciji učenja pa tanh često brže konvergira od sigmoid funkcije. Zbog toga, u praksi se tanh uvijek preferira nad sigmoidom.



Slika 1.3: Tanh funkcija i derivacija



Slika 1.4: Hardtanh funkcija i derivacija

Hard tanh aktivacijska funkcija je dana s:

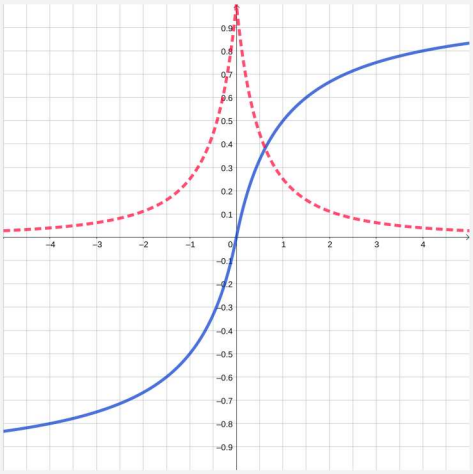
$$\text{hardtanh}(z) = \begin{cases} -1, & z < -1 \\ z, & -1 \le z \le 1 \\ 1, & z > 1 \end{cases}$$
$$\text{hardtanh}'(z) = \begin{cases} 1, & -1 \le z \le 1 \\ 0, & \text{inače} \end{cases}$$

Ova funkcija se nekada preferira u odnosu na tanh jer ju je jeftinije računati. Ipak, ne daje doprinose učenju za vrijednosti $|z| > 1$.

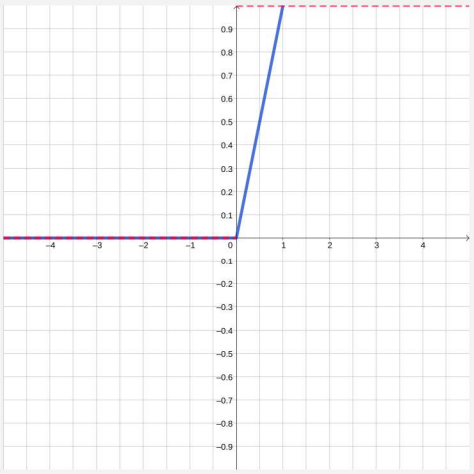
Softsign je funkcija slična tanh, a dana je s:

$$\text{softsign}(z) = \frac{z}{1+|z|}$$
$$\text{softsign}'(z) = \frac{\text{sgn}(z)}{(1+z)^2}$$

gdje je sgn(z) funkcija predznaka koja vraća ± 1, u ovisnosti o predznaku od z. Ova funkcija se svojoj asimptoti približava polinomijalno, za razliku od tanh koja to čini eksponencijalno. Zbog većih vrijednosti gradijenta za veće vrijednosti $|z|$, ova funkcija može brže konvergirati. Ipak, nije toliko popularna kao i druge aktivacijske funkcije.



Slika 1.5: Softsign funkcija i derivacija



Slika 1.6: ReLU funkcija i derivacija

ReLU (Rectified Linear Unit) funkcija je dana s:

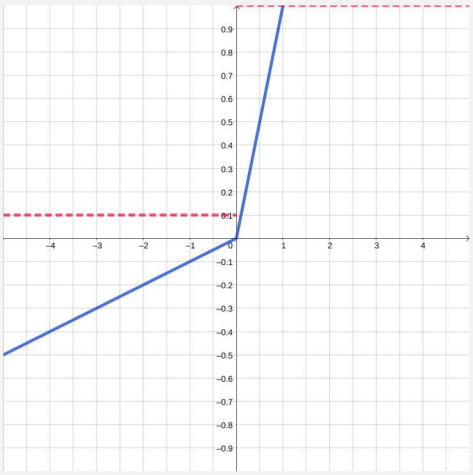
$$\text{ReLU}(z) = \max(z, 0)$$
$$\text{ReLU}'(z) = \begin{cases} 1, & z \geq 0 \\ 0, & \text{inače} \end{cases}$$

Ova funkcija je popularan izbor u novije vrijeme jer je jeftina za računati i ne zasiti neurone čak ni za velike vrijednosti od z. Kada je $z \leq 0$, gradijent će biti jednak 0, pa se taj neuron neće koristiti. To je dobra ideja jer može spriječiti pretreniranost (eng. *overfitting*), ali može i omesti mrežu u učenju ako previše neurona izumre. Kada je $z > 0$, gradijent će biti jednak 1, što znači da će mreža uvijek učiti. To rješava problem nestajućeg gradijenta koji je prisutan kod sigmoid funkcije. Ispostavlja se da ReLU značajno ubrzava konvergenciju gradijentnog spusta i ona time postaje dobar prvi izbor aktivacijske funkcije.

Leaky ReLU je dana s:

$$\text{leaky}(z) = \max(z, k \cdot z), \quad k \in (0,1)$$
$$\text{leaky}'(z) = \begin{cases} 1, & z \geq 0 \\ k & \text{inače} \end{cases}$$

Standardna ReLU funkcija ne uči za $z < 0$. Ova njena modifikacija propagira greške kroz neuron čak i za negativne vrijednosti z. Poželjno je birati male vrijednosti za k (oko 0.01) .

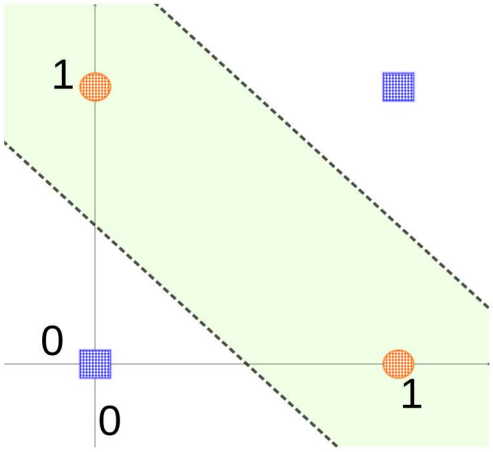


Slika 1.7: Leaky ReLU funkcija i derivacija ($k = 0.1$)

Za više informacija o djelovanju aktivacijskih funkcija u dubokom učenju možete pogledati [5].

Neuron koji koristi Heaviside step aktivacijsku funkciju se naziva i perceptron. Poznato je da jedan perceptron ne može predstaviti čak ni binarnu XOR funkciju (slika 1.8). Geometrijski gledano, računanje

težinske sume ulaza (formula 1.1) i binarna klasifikacija istog predstavlja razdvajanje vektorskog prostora ulaza hiperravninom. To znači da perceptron može jedino riješiti probleme u kojima je prostor linearno razdvojiv. Ovo ograničenje se može savladati višeslojnim perceptronima.



Slika 1.8: Problem klasifikacije binarne XOR funkcije

Općenito, možemo grupirati više umjetnih neurona u jedan sloj. U tom slučaju, svaki neuron dobiva sve ulaze $x_1, \ldots, x_n$. Ako označimo težine tih neurona s $w_1, \ldots, w_m$, pripadne parametre s $b_1, \ldots, b_m$ i aktivacije s $a_1, \ldots a_m$, možemo definirati sljedeće oznake kako bismo održali notaciju jednostavnom:

$$a = \begin{bmatrix} a_1 \\ \vdots \\ a_m \end{bmatrix} \in \mathbb{R}^m \qquad W = \begin{bmatrix} w_{1,1} & \ldots & w_{1,n} \\ & \vdots & \\ w_{m,1} & \ldots & w_{m,n} \end{bmatrix} \in \mathbb{R}^{m \times n}$$

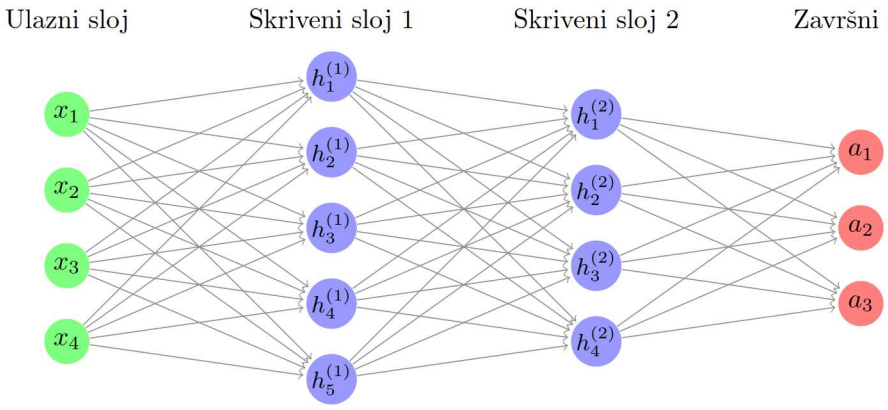
$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n \qquad b = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix} \in \mathbb{R}^m$$

Tada je izlaz sloja neurona:

$$a = f(z) = f(Wx + b)$$

gdje aktivacijska funkcija f djeluje na svakom elementu vektora z ponaosob. Na ove aktivacije možemo gledati kao indikatore prisutnosti nekih težinskih kombinacija ulaznih vrijednosti.

UNM se sastoje od ulaznog sloja, jednog ili više skrivenih slojeva i završnog sloja. Kada UNM ima dva ili



Slika 1.9: UNM s dva skrivena sloja, gdje $h_i^{(l)}$ označava i-ti neuron u l-tom skrivenom sloju

više skrivena sloja, nazivamo ju *duboka neuronska mreža (DNM)*[5]. Svaki sloj osim završnog sadrži parametar b i potpuno je povezan sa sljedećim slojem. Nenormalizirani vektor z, koji se dobije kao izlaz završnog sloja, prije djelovanja aktivacijske funkcije (poznat i pod eng. nazivom *logits*) se najčešće prosljeđuje nekoj normalizacijskoj funkciji. Ako rješavamo problem klasifikacije u k klasa, taj vektor se tipično prosljeđuje softmax funkciji. Ona vraća vektor normaliziranih vjerojatnosti za svaku klasu.

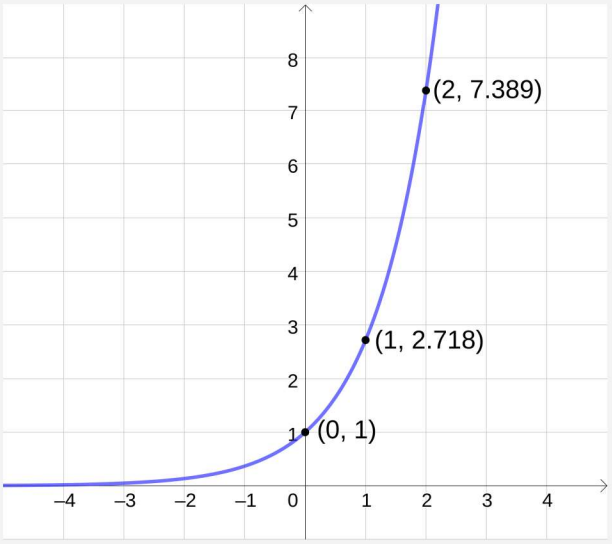
Softmax

Softmax funkcija je definirana s:

$$\text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)}, \qquad i = 1, \ldots, k \qquad z = \begin{bmatrix} z_1 \\ \vdots \\ z_k \end{bmatrix} \in \mathbb{R}^k$$

gdje je z ulazni vektor dimenzije broja klasa k. Ova funkcija ima nekoliko lijepih svojstava:

- Normalizira ulazne podatke tj. vraća pravu vjerojatnosnu distribuciju.
- Standardna argmax funkcija (hardmax) nije diferencijabilna. Softmax daje ne-nul vjerojatnost svakom elementu i glatka je. Otud i *soft* u nazivu.
- Prije normalizacije, svaki element se eksponencira.



Slika 1.10: Graf funkcije $x \mapsto e^x$

Zbog svoje nelinearne naravi, malo povećanje vrijednosti x uzrokuje veliko povećanje vrijednosti e^x. Kada ovo svojstvo koristimo s normalizacijom, uočit ćemo da velike vrijednosti x imaju puno veće vjerojatnosti nego male vrijednosti.

Primjerice, neka je $k = 5$ i $z = [1, 1, 2, 2, 4]$. argmax funkcija bi vratila:

$$[0, 0, 0, 0, 1]$$

To i jest cilj, ali argmax funkcija nije diferencijabilna i ne možemo ju koristiti za treniranje modela. Nakon obične normalizacije (koja je diferencijabilna), dobivamo:

$$[0.1, 0.1, 0.2, 0.2, 0.4]$$

To je prilično daleko od izlaza funkcije argmax. Softmax vraća:

$$[0.036, 0.036, 0.1, 0.1, 0.729]$$

Na ovom jednostavnom primjeru vidimo da softmax povećava vjerojatnosti za veće vrijednosti i smanjuje ih za manje. Otud i *max* u nazivu.

[5] eng. *deep neural network (DNN)*

1.2 Učenje neuronske mreže

Kada definiramo slojeve neuronske mreže, potreban nam je algoritam za računanje parametara W i b kojima radimo tranziciju na idući sloj mreže, i to za svaki sloj u mreži. Najčešće raspolažemo nekim skupom označenih podataka[6] na kojem želimo učiti svoju mrežu. Kada treniramo model, nije cilj memorizirati podatke za učenje, već želimo da model izračuna parametre pomoću kojih će mreža dobro raditi na novim podacima iz iste domene. Zbog toga, podaci za učenje bi trebali biti što sličniji podacima s kojima će mreža raditi. Naime, ako model treniramo na kazališnim tekstovima, imat će slab učinak na pravnim spisima.

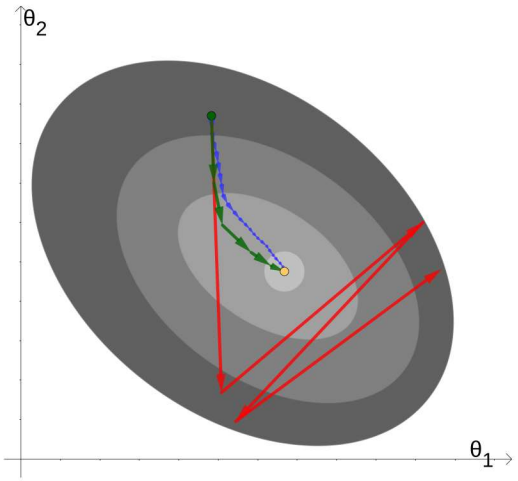
U svrhu vrednovanja parametara, definiramo funkciju cilja:

$$J(\theta) = \frac{1}{2N} \sum_{i=1}^{N} ||\hat{y}^{(i)} - y^{(i)}||^2$$

Ovdje, θ označava parametre mreže (W, b za svaki sloj), dok suma ide po svim primjerima za učenje i računa kvadratno odstupanje[7] predviđanja mreže $\hat{y}^{(i)}$ od točnog rješenja $y^{(i)}$. Uočimo da je vrijednost funkcije cilja nenegativna. Postiže minimum kada je izlaz mreže uvijek jednak točnom rješenju. S druge strane, greška je velika ukoliko naša mreža loše aproksimira mnoštvo primjera.

Za minimizaciju funkcije cilja koristimo iterativne optimizacijske metode. Najčešće se koristi metoda gradijentnog spusta (eng. *gradient descent*), jedna od najstarijih iterativnih metoda konveksne optimizacije. Gradijentni spust se temelji na ideji da za funkciju $J(\theta)$ u točki ekstrema vrijedi $\nabla J(\theta) = 0$, a da u ostalim točkama gradijent pokazuje smjer najbržeg porasta funkcije. Krenemo li od početne točke $\theta^{(0)}$, iterativnim ažuriranjem u smjeru $-\nabla J(\theta)$ možemo pronaći minimum funkcije J, ako postoji. Za konveksnu funkciju, pronađeni minimum jest globalni minimum. Funkcije cilja koje optimiziramo nisu uvijek konveksne, pa možemo dospjeti i u lokalni minimum umjesto globalnog. Pretpostavimo da parametre modela inicijaliziramo slučajnim vrijednostima. Nakon toga, ponavljamo sljedeći korak dok ne dođemo do konvergencije:

$$\theta_j^{(k+1)} = \theta_j^{(k)} - \alpha \frac{\partial}{\partial \theta_j} J(\theta), \qquad \forall j$$



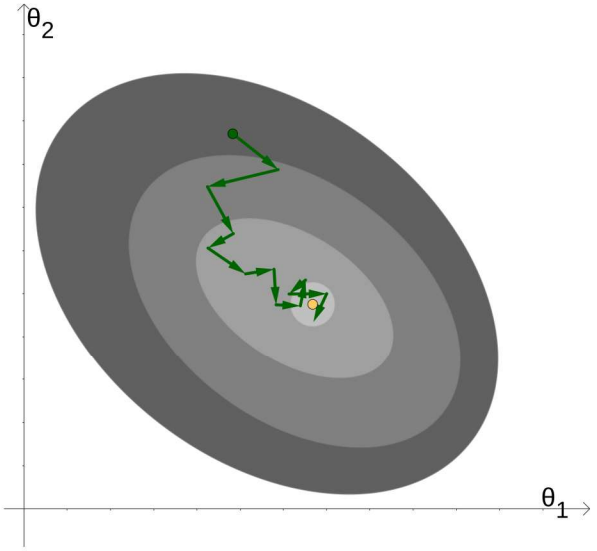
Slika 1.11: Ilustracija gradijentne metode za funkciju dvije varijable: θ_1, θ_2. Plavom bojom su prikazane iteracije za malu stopu učenja α, zelenom za dobru stopu učenja, a crvenom za veliku stopu učenja.

[6] Neke označene skupove podataka za strojno učenje možete pronaći na: `https://en.wikipedia.org/wiki/List_of_datasets_for_machine-learning_research`.

[7] Kvadratna funkcija nije jedini izbor, funkcija cilja ovisi o specifičnoj primjeni. Druge funkcije cilja će dati drukčije optimalne parametre.

Valja obratiti pozornost kod odabira stope učenja α u gradijentnoj metodi (vidi sliku 1.11). Naime, za premali α algoritmu može biti potrebno jako puno koraka do konvergencije. Ukoliko odaberemo preveliku stopu učenja, algoritam može divergirati tj. iteracije mogu poprimiti sve veće vrijednosti i tako promašiti optimalno rješenje. Dodatno, kako bismo poboljšali brzinu konvergencije, dobro je skalirati vrijednosti parametara da poprimaju vrijednosti iz istog segmenta: npr. $\theta_j \in [0, 1], \forall j$.

Za implementaciju gradijentne metode, trebaju nam parcijalne derivacije funkcije cilja po svakoj varijabli u θ. Kako je funkcija cilja definirana na cijelom skupu za učenje, računanje gradijenta će biti jako sporo kada imamo mnoštvo podataka. Ovaj algoritam se može pronaći i pod imenom grupni gradijentni spust (eng. *batch gradient descent*). Alternativa mu je stohastički gradijentni spust (eng. *stochastic gradient descent*), koji u svakoj iteraciji aproksimira gradijent koristeći samo jednu instancu (ili mali podskup skupa) za učenje.



Slika 1.12: Ilustracija stohastičkog gradijentnog spusta

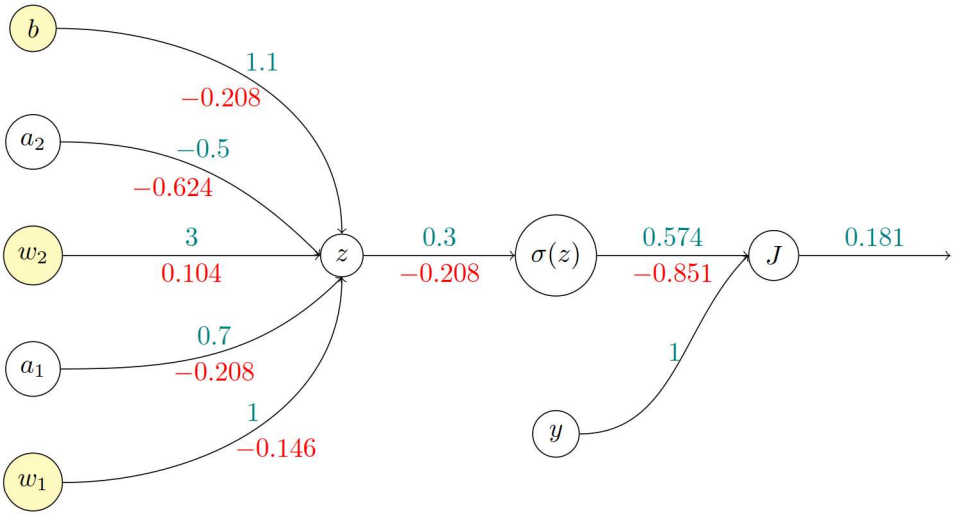
U praksi je manje računalno zahtjevan jer računalo ne mora u memoriji držati cijeli skup za učenje u svakoj iteraciji. Jednom kada dođe blizu optimalnog rješenja, poskakivat će oko njega. Dakle, kada se algoritam zaustavi imat ćemo jako dobre parametre, ali ne najbolje. Jedno rješenje u tom slučaju je postupno smanjivati stopu učenja α, dok algoritam ne pronađe minimum. Zbog svoje stohastičke (slučajne) naravi, može čak i izaći iz lokalnog minimuma, ako tamo zapne, i nastaviti tražiti globalno optimalno rješenje.

Još nismo spomenuli kako *zapravo* računamo gradijent funkcije. Iako za računanje gradijenta u jednoj iteraciji ne koristimo cijeli skup za učenje, već samo mali podskup, i dalje je potrebno računati parcijalnu derivaciju po svakom parametru modela. Može biti riječ o milijunima težina. Ključan algoritam koji nam pomaže u rješavanju tog problema je propagacija greške unatrag (eng. *backpropagation*). Načelno, riječ je o tehnici za brzo računanje derivacija. Ova ideja nije nova, datira iz 1986. (vidi [12]).

Algoritam daje svaku instancu učenja neuronskoj mreži i računa izlaz svakog neurona u svakom uzastopnom sloju. Riječ je o tzv. prolazu unaprijed (eng. *forward pass*). Nakon toga, mjeri pogrešku mreže (razliku između željenog i pravog izlaza mreže) i računa koliko je koji neuron u završnom sloju mreže doprinio toj grešci. Proces računanja doprinosa greški pojedinih neurona se nastavlja kroz skrivene slojeve sve do ulaznog sloja. Ovim prolazom unatrag (eng. *reverse pass*) efikasno računamo udio greške na vezi između svaka dva neurona u mreži propagacijom greške unatrag. Posljednji korak u ovom algoritmu je učiniti iteraciju gradijentnog spusta i ažurirati sve parametre mreže koristeći prethodno izračunate parcijalne derivacije.

Primjer 1.1. (Deriviranje akumulacijom unatrag)

U ovom primjeru ćemo pojasniti postupak koji se koristi za računanje parcijalnih derivacija, poznat i pod eng. nazivom reverse mode autodiff. *Na parcijalnu derivaciju neke varijable gledamo kao ovisnost cijelog izraza o toj varijabli. Promotrimo djelovanje dijela neuronske mreže na jednoj instanci za učenje. Neka su a_1 i a_2 aktivacije neurona u prethodnom sloju neuronske mreže, i neka trenutni sloj mreže, radi jednostavnosti, ima samo jedan sigmoid neuron. Za njega možemo izračunati $z = w_1a_1 + w_2a_2 + b$ i pripadnu aktivaciju $\sigma(z)$. Za točno rješenje y, greška koju je mreža napravila jest $J = (\sigma(z) - y)^2$.*



Slika 1.13: Dio neuronske mreže prikazan pomoću izračuna u čvorovima. Žutom bojom su označeni čvorovi parametara koji se mogu izravno ažurirati

Na slici 1.13, zelenom bojom su označene trenutne vrijednosti parametara, međuvrijednosti izračunate prolaskom unaprijed i greška mreže. Ideja je sada ići unatrag kroz graf i izračunati parcijalnu derivaciju funkcije J u svakom čvoru. Ključni alat za to je lančano pravilo kod derivacija. Dobivene parcijalne derivacije su označene crvenom bojom na slici 1.13, a izračunate su koristeći sljedeće izraze:

$$\frac{\partial J}{\partial \sigma(z)} = 2(\sigma(z) - y)$$

$$\frac{\partial J}{\partial w_i} = \frac{\partial J}{\partial \sigma(z)} \frac{\partial \sigma(z)}{\partial z} \frac{\partial z}{\partial w_i},\ i = 1, 2 \qquad \frac{\partial J}{\partial z} = \frac{\partial J}{\partial \sigma(z)} \frac{\partial \sigma(z)}{\partial z}$$

$$\frac{\partial J}{\partial a_i} = \frac{\partial J}{\partial \sigma(z)} \frac{\partial \sigma(z)}{\partial z} \frac{\partial z}{\partial a_i},\ i = 1, 2 \qquad \frac{\partial J}{\partial b} = \frac{\partial J}{\partial \sigma(z)} \frac{\partial \sigma(z)}{\partial z} \frac{\partial z}{\partial b}$$

pritom koristeći derivacije elementarnih funkcija da bismo izračunali:

$$\frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1 - \sigma(z)) \qquad \frac{\partial z}{\partial w_i} = a_i,\ i = 1, 2$$

$$\frac{\partial z}{\partial a_i} = w_i,\ i = 1, 2 \qquad \frac{\partial z}{\partial b} = 1$$

Ova tehnika je jako korisna ako postoji puno ulaza i samo jedan izlaz, jer zahtijeva samo jedan prolaz unaprijed i jedan prolaz unatrag da bi izračunala parcijalne derivacije izlaza po svakom ulazu. [8]

[8] Više o ovoj temi možete pronaći u [4], na blogu Grant Sanderson-a (`https://www.3blue1brown.com/videos-blog/2019/4/29/backpropagation-calculus-deep-learning-chapter-4`) i na blogu Christopher Olah-a (`http://colah.github.io/posts/2015-08-Backprop/`).

1.3 Praktični savjeti

Nakon što smo opisali matematičke temelje neuronskih mreža, navest ćemo neke praktične savjete i tehnike koje se često koriste u praksi. Naime, fleksibilnost neuronskih mreža je u jednu ruku i njihova mana. Čak i u jednostavnim mrežama, možemo birati između različitog broja slojeva, broja neurona po sloju, vrsti aktivacijskih funkcija, inicijalizaciji parametara itd. Pronaći najbolju kombinaciju istih za zadatak koji rješavamo može zahtijevati puno vremena.

Broj skrivenih slojeva

Većinu problema možemo pokušati riješiti s jednim skrivenim slojem i dobit ćemo razumne rezultate. Štoviše, takve mreže mogu aproksimirati i najsloženije funkcije ukoliko im damo dovoljno velik broj neurona. Ipak, prednost DNM jest što mogu taj isti zadatak riješiti s eksponencijalno manje parametara nego obične UNM, što ih čini bržima za treniranje. Dobra usporedba se može pronaći u [4]: pretpostavimo da trebate nacrtati šumu u nekom softveru za crtanje, a da vam je pritom zabranjeno koristiti kopiraj / zalijepi (eng. *copy / paste*). Morali bismo crtati svako drvo posebno, granu po granu, list po list. Kada bismo mogli nacrtati jedan list i kopirati ga da napravimo granu, njih kopirati da napravimo drvo i konačno njega kopirati da nacrtamo šumu, bili bismo jako brzo gotovi. Tako su i podaci iz stvarnog svijeta često hijerarhijski strukturirani i DNM koriste tu činjenicu: niži slojevi modeliraju strukture niže razine (npr. linije i segmente različitih orijentacija), raniji skriveni slojevi kombiniraju te strukture kako bi prepoznali složenije oblike (npr. kvadrat, krug) dok dublji skriveni slojevi, koristeći prethodne strukture, modeliraju strukture više razine (npr. lice).

Broj neurona po skrivenom sloju

Jedna od ideja jest umanjivati broj neurona u svakom skrivenom sloju, s idejom da veći broj struktura niže razine sraste u manji broj struktura više razine. Ipak, u novije vrijeme se koristi jednak broj neurona u svakom sloju (samo jedan hiperparametar za sve slojeve) uz primjenu isključivanja čvorova (eng. *dropout*), kojeg ćemo u nastavku opisati.

Regularizacija i isključivanje čvorova

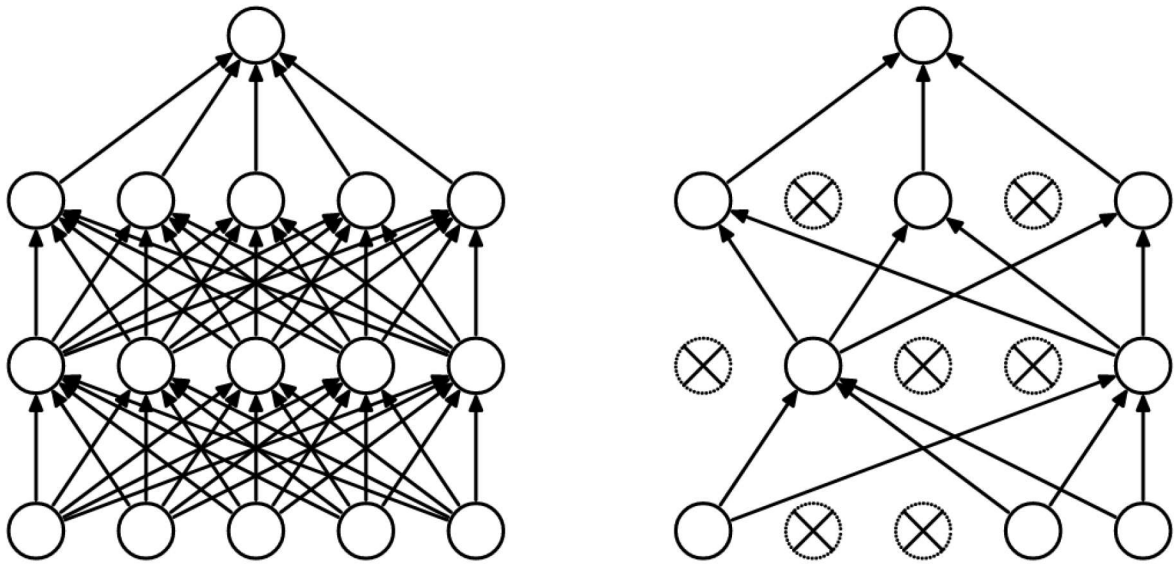
Kao i kod drugih modela u strojnom učenju, neuronske mreže su sklone pretreniranosti (eng. *overfitting*) tj. model ima gotovo savršen učinak na instancama za učenje, a daje loše rezultate na testnom skupu. Česta tehnika kojom se ova pojava otklanja jest regularizacija. Ideja je dodati novi član u funkciji cilja kako bismo kaznili prevelike vrijednosti parametara:

$$J_R = J + \lambda \sum_{i=1}^{L} ||W^{(i)}||_F$$

gdje je $||A||_F = \sqrt{\sum_i \sum_j a_{ij}^2}$ Frobeniusova norma matrice $A = [a_{ij}]$, a suma ide po svih L slojeva mreže. Zbog svoje kvadratične naravi, ova regularizacija smanjuje fleksibilnost modela i time otežava težinama da se potpuno prilagode skupu za učenje. Odabir dobrog parametra λ je jako važno. Naime, prevelika vrijednost λ će iz optimizacijskog postupka vratiti težine bliske nuli i mreža neće gotovo ništa naučiti. Premale vrijednosti λ neće uopće utjecati na težine, kao da regularizacije nikada nije ni bilo. Također, uočite da ne regulariziramo parametre b iz slojeva mreže.

Novija tehnika za regularizaciju jest isključivanje čvorova, koje je predstavljeno u [13]. Riječ je o jednostavnoj, ali vrlo efektivnoj ideji: tijekom svake iteracije učenja, svaki neuron (uključujući ulazni sloj, ali

ne i završni) će se koristiti s vjerojatnosti p. Dakle, za svaki neuron postoji vjerojatnost $1 - p$ da neće biti korišten tijekom učenja mreže. Često se uzima $p = 0.5$.



Slika 1.14: Na lijevoj strani je UNM s dva skrivena sloja. S desne strane je prorijeđena mreža. Prekriženi čvorovi se ne koriste pri treniranju.

Neuroni trenirani na ovaj način se ne mogu osloniti na susjedne neurone i moraju sami pokušati biti što korisniji. Također, ne mogu se prilagoditi nekolicini ulaznih neurona, nego će pratiti sve ulazne neurone. Tako mreža postaje robusnija i manje osjetljiva na male promjene na ulazu. Mreža će naučiti značajnije informacije o podacima i postizati bolji učinak u rješavanju danog zadatka.

Tijekom testiranja, koristit će se svi neuroni u mreži. Postoji jedan važan tehnički detalj na koji treba obratiti pozornost: očekivana aktivacija neurona tijekom testiranja treba biti približno jednaka aktivaciji tijekom učenja. U suprotnom, aktivacije će se značajno razlikovati i mreža neće dobro raditi. Zbog toga, tijekom učenja možemo podijeliti aktivaciju neurona s vjerojatnosti zadržavanja p (ili ju pomnožiti s p nakon učenja).

Inicijalizacija parametara

Važan korak pri izgradnji kvalitetne neuronske mreže je dobra inicijalizacija parametara. Uočeno je da se greške slabije propagiraju od završnih slojeva prema početnim odmah nakon inicijalizacije parametara.

Jednostavna strategija koja je prihvatljiva jest inicijalizirati parametre s malim vrijednostima normalno distribuiranima oko nule. U [5], autori predlažu sljedeću inicijalizacijsku shemu tj. specifičnu uniformnu distribuciju:

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n^{(L)}+n^{(L+1)}}}, \frac{\sqrt{6}}{\sqrt{n^{(L)}+n^{(L+1)}}}\right]$$

gdje je $n^{(L)}$ dimenzija ulaza za W, a $n^{(L+1)}$ dimenzija izlaza. U ovoj shemi, parametri b mreže se inicijaliziraju s 0. Ovaj pristup nastoji očuvati varijance aktivacija i varijance propagiranja greške unatrag kroz sve slojeve. Shema je poznata i pod nazivom Xavier inicijalizacija i Glorot inicijalizacija, prema autorima spomenutog rada.

Drugi savjeti

Postoji još ideja za poboljšati rad neuronskih mreža, a uključuju, povećanje skupa za treniranje (pomaci, rotacije, rezanje, kontrast itd.), priprema podataka (normalizacija), druge metode za optimizaciju (Nestorov Accelerated Gradient, AdaGrad, RMSProp, Adam), korištenje (dijelova) već istreniranih neuronskih mreža i druge. Više o tome možete pronaći u [4], [7] i drugoj literaturi o neuronskim mrežama.

2 Vektori riječi

U radu s prirodnim jezikom i riječima, susrećemo se s klasifikacijom riječi u različite klase. Na taj način se riječi pripremaju za uporabu u različitim zadacima obrade prirodnog jezika. Provedeno je mnoštvo istraživanja kako bi se riječi što bolje preslikale u oblik vektora kojeg mogu koristiti različiti algoritmi i procesi.

Međunarodni naziv za vektor riječi, *word embedding*, dolazi od engleskih riječi *word* - riječ i *embedding* - ulaganje. U matematici, ulaganje je primjer jedne matematičke strukture koja je sadržana u drugoj. Kažemo da je struktura X uložena u drugu strukturu Y, ako postoji injekcija $f : X \to Y$[9] koja čuva strukturu. Precizna definicija čuvanja strukture ovisi o matematičkoj strukturi čije su X i Y instance.

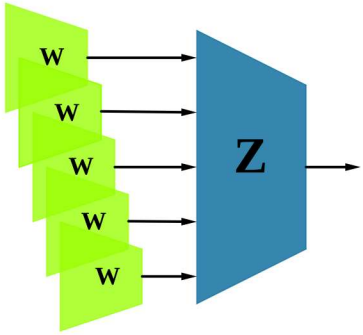
Word embedding je kolektivno ime za skup tehnika u modeliranju jezika i učenju značajki (eng. *feature learning* ili *representation learning*) koje riječi iz rječnika preslikaju u vektore realnih brojeva. Riječ je o ulaganju s prostora velikih dimenzija u vektorski prostor puno manje dimenzije. Preciznije, riječ je o vektorima koji oslikavaju strukturu riječi u smislu morfologije (oblik i građa), semantike (značenje), konteksta riječi, hijerarhije među riječima ili drugo. Ideja je obuhvatiti što je više moguće informacija o riječi.

Hrvatski jezik ima stotine tisuća osnovnih riječi, i znatno više izvedenih oblika koji nastaju deklinacijom, konjugacijom i komparacijom. Jesu li te riječi nepovezane? Miš i štakor, hotel i motel, dobar i bolji? Zasigurno nisu. Stoga, svaku riječ želimo reprezentirati vektorom u N-dimenzionalnom prostoru, za dovoljno mali N koji će obuhvatiti većinu semantike našeg jezika. Svaka dimenzija će kodirati neko značenje riječi. Primjerice, jedna dimenzija može predstavljati glagolsko vrijeme (prošlost, sadašnjost, budućnost), druga broj (jednina, množina), a neka treća spol (muški, ženski).

Primjer 2.1. (Vektori riječi)
Neka je W : ***riječi*** $\to \mathbb{R}^{100}$ *preslikavanje riječi hrvatskog jezika u vektore. Preslikavanje bi moglo raditi ovako:*

$$W(\text{"mačka"}) = (-0.163, -0.198, 0.161, \ldots, -0.159)$$
$$W(\text{"škola"}) = (-0.217, 0.156, -0.136, \ldots, -0.07)$$

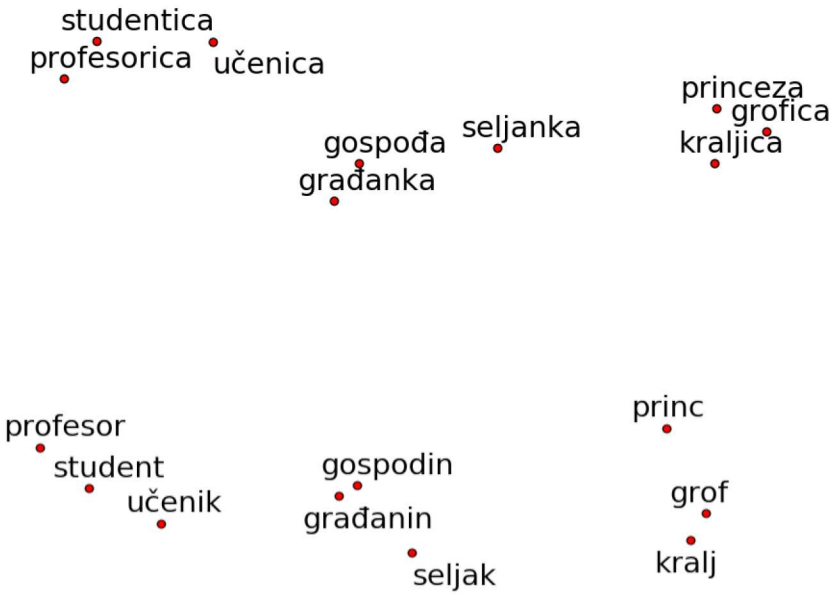
W se inicijalizira vektorom nasumičnih brojeva za svaku riječ i kroz strojno učenje poprimi vrijednosti koje pomažu u rješavanju nekog zadatka. Vektori riječi su kamen temeljac za mnoštvo zadataka u obradi prirodnog jezika. Primjerice, jednostavan zadatak bi bio pogađanje iduće riječi u rečenici na temelju nekoliko prethodnih riječi. Neki složeniji zadaci su odgovaranje na pitanja, stvaranje teksta i strojno prevođenje.



Slika 2.1: Mreža koja koristi 5 vektora riječi kao ulaz za zadatak Z

[9] Činjenica da je preslikavanje $f : X \to Y$ ulaganje se često označava "strelicom s kukom" tj. $f : X \hookrightarrow Y$.

Kako bismo dobili bolji osjećaj za vektorski prostor riječi, vizualizirat ćemo njegovu projekciju na dvodimenzionalni prostor koristeći osnovnu analizu komponenata (eng. *principal component analysis*).



Slika 2.2: Projekcija vektora izabranih riječi na dvije komponente

Ovo "preslikavanje" nam je vrlo intuitivno. Naime, slične riječi se preslikaju jedna pored druge. Ipak, pomalo je iznenađujuće kako je računalo naučilo koje riječi imaju bliska značenja. Možemo gledati i koje su najbliže riječi danoj fiksnoj riječi. Rezultati u tablici 2.1 su dobiveni korištenjem sličnosti kosinusa: za dane vektore w_1 i w_2, njihova sličnost $\cos\theta \in [-1, 1]$ jest:

$$\text{sličnost}(w_1, w_2) = \cos\theta = \frac{w_1 \cdot w_2}{||w_1||||w_2||}$$

Ova funkcija mjeri sličnost po smjeru, a ne po duljini vektora. Paralelni vektori će imati sličnost 1, okomiti vektori 0, dok će suprotni vektori imati sličnost -1.

dobar	**Osijek**	**istražujem**	**žućkast**	**xbox**
izvrstan (0.839)	Varaždin (0.865)	proučavam (0.768)	zelenkast (0.899)	playstation (0.894)
odličan (0.823)	Vinkovci (0.856)	zapisujem (0.762)	smećkast (0.891)	gamecube (0.891)
loš (0.802)	Đakovo (0.847)	ispitujem (0.743)	crvenkast (0.887)	ps3 (0.855)
sjajan (0.768)	Karlovac (0.823)	pronađem (0.736)	smeđ (0.881)	wii (0.854)
koristan (0.766)	Split (0.816)	razvijam (0.735)	bjelkast (0.868)	ps4 (0.847)
pristojan (0.754)	Požega (0.813)	usavršim (0.728)	sivkast (0.857)	nintendo (0.845)
lijep (0.745)	Koprivnica (0.811)	izučavam (0.728)	poluproziran (0.856)	gameboy (0.836)

Tablica 2.1: Riječi sa sličnim vektorskim reprezentacijama

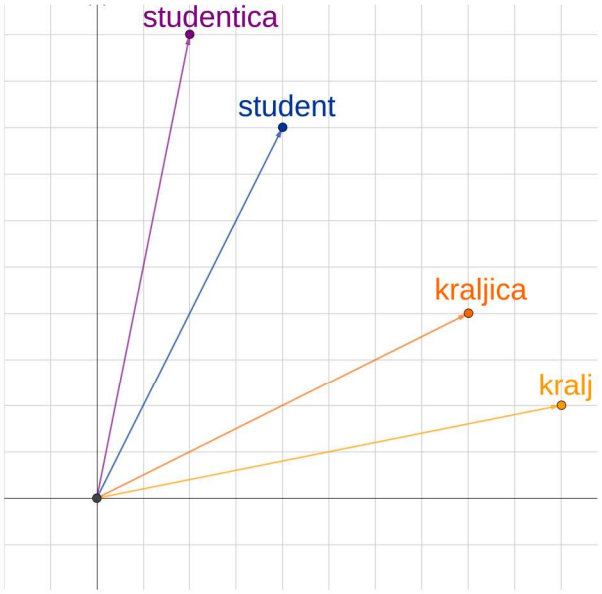
Između riječi postoje različite vrste sličnosti. Primjerice, riječ *velik* je slična riječi *najveći* na isti način na koji je riječ *visok* slična riječi *najviši*. Iako to zvuči neočekivano, ovakve sličnosti možemo pronaći osnovnim

algebarskim operacijama nad vektorima. Kako bismo pronašli riječ koja je slična *visok* na način koji je *velik* slična *najveći*, izračunamo: vektor(*najveći*) - vektor(*velik*) + vektor(*visok*). Analogije između riječi su kodirane u razlici vektora riječi. Uočimo da su na slici 2.2 vektori razlika između ženskih i muških riječi približno konstantni. Navedeno zapažanje možemo koristiti da pronađemo analogije.

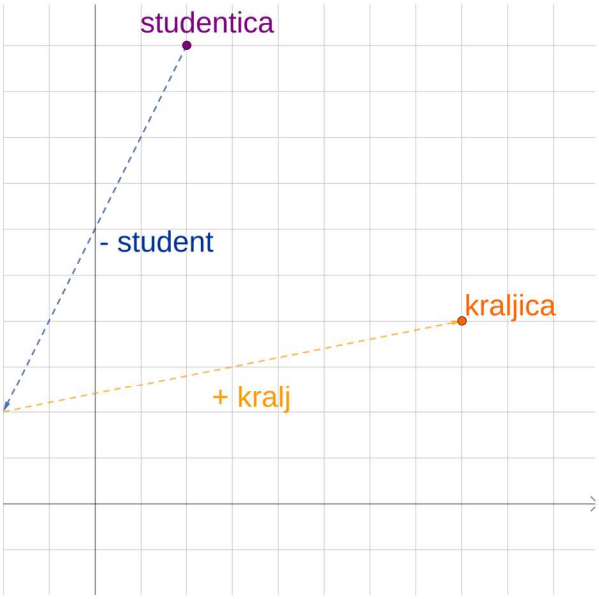
Primjer 2.2. (Analogije)

$$studentica - student + kralj = kraljica$$

Kada od riječi studentica *oduzmemo značenje značenja riječi* student, *preostaju nam osobine ženske osobe. Ako na to dodamo značajke riječi* kralj, *dobivamo riječ koje nosi značenje kralja i ženske osobe tj. riječ* kraljica.



Slika 2.3: Vektori riječi iz primjera



Slika 2.4: Operacije nad vektorima

Slijede još neke analogije koje se mogu izračunati koristeći vektore riječi.

Analogija	**Primjer 1**	**Primjer 2**	**Primjer 3**
Hrvatska - Zagreb	Francuska: Pariz	Engleska: London	BiH: Sarajevo
velik - najveći	visok: najviši	bogat: najbogatiji	lijep: najljepši
muškarac - žena	predsjednik: predsjednica	pobjednik: pobjednica	doktor: doktorica
učenik - učenici	stolica: stolice	krava: krave	pčela: pčele
čitam - čitao	jedem: jeo	pišem: pisao	vidim: vidio

Tablica 2.2: Analogije riječi

Važno je istaknuti da su ova svojstva samo nuspojava vektora riječi. Takva reprezentacija nije posljedica dizajna, već je nastala iz optimizacijskog procesa. Čini se da je to velika prednost umjetnih neuronskih mreža: samostalno uče bolje načine za reprezentaciju podataka.

U nastavku ćemo se vratiti korak unatrag i predstaviti neke jednostavnije vektorske reprezentacije riječi te istaknuti njihove prednosti i nedostatke. Nakon toga ćemo prezentirati noviji, složeniji pristup za računanje vektora riječi koji smo koristili u ovom poglavlju. Sličan pristup možete pronaći i u [7].

2.1 One-hot vektor

Najjednostavnija reprezentacija riječi je koristeći tzv. *one-hot*[10] vektor. Svaka riječ je predstavljena $\mathbb{R}^{|V|\times 1}$ vektorom popunjenog s 0, koji ima samo jednu 1 na indeksu te riječi u sortiranom rječniku tog jezika. $|V|$ označava broj riječi u jeziku. Primjerice,

$$w^{\text{a}} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, w^{\text{abak}} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \ldots, w^{\text{žvaka}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

Svaka riječ je zaseban entitet. Ova reprezentacija riječi ne sadrži nikakav pojam o sličnosti među riječima. Vektori različitih riječi su međusobno okomiti:

$$(w^{\text{miš}})^T w^{\text{štakor}} = (w^{\text{hotel}})^T w^{\text{motel}} = 0$$

Ovakav koncept, gdje ideju predstavljamo simbolom (riječ ili one-hot vektor) nazivamo **denotacijska semantika**. Reprezentacija je rijetko popunjena (eng. *sparse*) i ne može dohvatiti sličnost. S druge strane, koncept u kojem značenje riječi dobivamo iz njezinog konteksta nazivamo **distribucijska semantika**. Takva reprezentacija nije rijetka i bolje obuhvaća sličnosti između riječi.

2.2 Metode bazirane na SVD dekompoziciji

Kako bi ova klasa metoda pronašla vektore riječi, najprije treba proći kroz ogromnu bazu tekstova i bilježiti koje se riječi pojavljuju jedna pored druge u neku matricu X. Na njoj se provodi singularna dekompozicija matrice i dobivamo USV^T, gdje retke od U koristimo za vektore riječi iz rječnika. Postoje različiti načini za konstrukciju matrice X.

- **Matrica riječ-dokument**

  Možemo pretpostaviti da će se povezane riječi nalaziti u istim dokumentima. Primjerice, "učenik", "škola", "razred", "nastava" itd. će se nalaziti u istom dokumentu. S druge strane, "škola", "banana", "klavir" i "lovac" se neće konzistentno pojavljivati u istom dokumentu. Koristeći ovu činjenicu, možemo matricu X izgraditi na sljedeći način: iteriramo po velikoj količini dokumenata i kada god susretnemo riječ i u dokumentu j, uvećamo vrijednost X_{ij} za jedan. Ovo je očito jako velika matrica ($\mathbb{R}^{|V|\times M}$) i sve je veća s većim brojem dokumenata (M).

- **Matrica susjednih pojavljivanja**

  Matricu X možemo popunjavati s brojem pojavljivanja susjednih riječi i time bilježiti sklonost riječi da se pojavljuju jedna pored druge. Slijedi primjer s tri rečenice i susjedstvom veličine 1.

  Primjer 2.3.

  1. *Ja sam student.*
  2. *Ja volim matematiku.*

[10] Naziv *one-hot* dolazi iz digitalne elektronike, predstavlja grupu bitova gdje su jedine valjane kombinacije one koje imaju jednu 1 i sve ostale 0.

3. *Ja volim svemir.*

Rezultirajuća matrica bi izgledala ovako:

$$X = \begin{array}{c} \begin{matrix} \textit{ja} & \textit{matematiku} & \textit{sam} & \textit{student} & \textit{svemir} & \textit{volim} & . \end{matrix} \\ \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 2 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \end{pmatrix} \end{array} \begin{matrix} \textit{ja} \\ \textit{matematiku} \\ \textit{sam} \\ \textit{student} \\ \textit{svemir} \\ \textit{volim} \\ . \end{matrix}$$

Provodimo SVD na matrici X i promatramo singularne vrijednosti σ_i (dijagonalni elementi matrice S). Uzmemo samo prvih k indeksa na temelju postotka varijance koji želimo uhvatiti:

$$\frac{\sum_{i=1}^{k} \sigma_i}{\sum_{i=1}^{|V|} \sigma_i}$$

Tada bismo uzeli podmatricu $U_{1:|V|,\,1:k}$ da bude matrica vektora riječi. Time bismo dobili k-dimenzionalnu reprezentaciju svake riječi u rječniku.

Obje metode nam daju dovoljno informacija kako bismo kodirali semantičke i sintaktičke informacije o riječima, ali se susrećem s mnoštvo drugih problema:

- Dimenzije matrica se često mijenjaju: nerijetko se dodaju nove riječi i veličina korpusa[11] se mijenja
- Matrica je rijetko popunjena, jer se većina riječi ne podudara
- Matrica je generalno velikih dimenzija ($\approx 10^6 \times 10^6$)
- Vrijeme za treniranje je kvadratično (provođenje SVD[12])

Postoje rješenja za neke od ovih problema, ali nisu u potpunosti praktična. S druge strane, u nastavku ćemo pokazati kako iterativne metode puno elegantnije savladavaju većinu ovih problema.

2.3 Iterativne metode - word2vec

"You shall know a word by the company it keeps."

– John Rupert Firth

Umjesto da računamo i spremamo globalne informacije o velikoj bazi tekstova, možemo napraviti model koji će iterativno učiti i s vremenom naučiti kodirati vjerojatnost pojavljivanja riječi, uz zadani kontekst. Kontekst riječi je skup od m riječi koji ju okružuju. Primjerice, za $m = 2$ kontekst riječi *"orahu"* u rečenici *"Tko nije zahvalan na orahu nije ni na tovaru."* je {*"zahvalan"*, *"na"*, *"nije"*, *"ni"* }.

Ideja je dizajnirati model čiji će parametri biti vektori riječi. Nadalje, treniramo model minimizirajući neku funkciju cilja. U svakoj iteraciji pokrenemo model, analiziramo greške i ažuriramo parametre tako da

[11] Korpus je cjelovita zbirka podataka, dokumenata i građe.
[12] Računanje SVD dekompozicije zahtijeva $O(mn^2)$ operacija, za matricu iz $\mathbb{R}^{m \times n}$, $m \geq n$.

popravimo parametre koji su uzrokovali greške. Tako učimo vektore riječi. Riječ je o algoritmu propagacije greške unatrag.

Ovdje ćemo izložiti noviji, vjerojatnosni pristup iz [8]: word2vec. Word2vec je softverski paket koji se sastoji od:

- **2 algoritma**: *continuous bag-of-words* (CBOW) i *skip-gram*. CBOW predviđa središnju riječ, za dani kontekst. Skip-gram čini suprotno: predviđa vjerojatnosnu distribuciju konteksta za danu središnju riječ.
- **2 metode treniranja**: negativno uzorkovanje (eng. *negative sampling*) i hijerarhijski softmax. Negativno uzorkovanje definira funkciju cilja pomoću odabranih negativnih uzoraka, dok hijerarhijski softmax definira funkciju cilja koristeći strukturu stabla kako bi izračunao vjerojatnosti za riječi u rječniku.

2.3.1 Modeli jezika

Najprije, trebamo model koji će pridruživati vjerojatnosti za zadani niz riječi.

Primjer 2.4.

"Pas je zalajao vidjevši mačku."

Dobar model jezika bi ovoj rečenici dao veliku vjerojatnost jer je gramatički i semantički valjana. S druge strane,

"Siromah san je šapnuo prozor."

bi trebao imati malu vjerojatnost jer nema nikakvog smisla.

Matematički, domena ove funkcije je bilo koji niz od n riječi, tj. možemo računati $P(w_1, w_2, \ldots, w_n)$. Možemo uzeti **unarni model** jezika i pretpostaviti da su pojavljivanja riječi međusobno neovisna:

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{N} P(w_i)$$

Međutim, znamo da je ovo suludo, jer svaka riječ uvelike ovisi o riječima koje ju okružuju. Koristeći ovaj model, možda bi rečenica koja nema nikakvog smisla imala veliku vjerojatnost. Stoga, možemo definirati vjerojatnost niza riječi kao vjerojatnost pojavljivanja, uz uvjet pojavljivanja prethodne riječi. Ovaj model nazivamo **model bigrama**:

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=2}^{N} P(w_i|w_{i-1})$$

Naravno, i ovaj pristup je pomalo naivan, kako promatramo samo susjedne riječi umjesto cijele rečenice, ali ova reprezentacija je već solidna. Uočimo da iz matrice susjednih pojavljivanja (sa susjedstvom veličine 1) možemo naučiti ove vjerojatnosti, ali to bi zahtijevalo računanje i spremanje globalnih informacija o velikoj bazi riječi.

Shvaćajući kako predstaviti vjerojatnost niza riječi, proučit ćemo modele kojima ćemo moći naučiti te vjerojatnosti.

2.3.2 Continuous bag of words model (CBOW)

Jedan pristup je tretirati {*"Pas", "je", "vidjevši", "mačku"*} kao kontekst kojim ćemo generirati središnju riječ *"zalajao"*. Definirajmo najprije parametre CBOW modela. Neka su poznati parametri u modelu riječi (iz rečenice) predstavljene one-hot vektorima. Vektore ulaznih riječi označit ćemo s $x^{(i)}$, a izlaznih s $y^{(j)}$. Kako u CBOW modelu imamo samo jednu izlaznu riječ (središnju), označit ćemo s y njezin one-hot vektor.

Načinit ćemo dvije matrice: $\mathcal{V} \in \mathbb{R}^{n \times |V|}$, $\mathcal{U} \in \mathbb{R}^{|V| \times n}$, gdje je n dimenzija vektora riječi koju unaprijed definiramo.[13] $\mathcal{V}$ je matrica ulaznih riječi, njen i-ti stupac je n-dimenzionalan vektor koji se koristi kada je riječ w_i ulaz za model. Označit ćemo taj vektor s v_i. Slično, $\mathcal{U}$ je matrica izlaznih riječi. Njen j-ti redak je n-dimenzionalan vektor koji se koristi kada je riječ w_j izlaz za model. Označit ćemo taj vektor s u_j. Uočimo da za svaku riječ w_i učimo dva vektora: ulazni vektor v_i i izlazni vektor u_i.
Model radi na sljedeći način:

1. Generiramo one-hot vektore za riječi iz konteksta veličine m

$$x^{(c-m)}, \ldots, x^{(c-1)}, x^{(c+1)}, \ldots, x^{(c+m)} \in \mathbb{R}^{|V|}$$

2. Dohvatimo vektore riječi za kontekst:

$$v_{c-m} = \mathcal{V}x^{(c-m)}, v_{c-m+1} = \mathcal{V}x^{(c-m+1)}, \ldots, v_{c+m} = \mathcal{V}x^{(c+m)} \in \mathbb{R}^n$$

3. Izračunamo prosjek tih vektora i dobijemo

$$\hat{v} = \frac{v_{c-m} + v_{c-m+1} + \cdots + v_{c+m}}{2m} \in \mathbb{R}^n$$

4. Definiramo vektor ocjene $z = \mathcal{U}\hat{v} \in \mathbb{R}^{|V|}$. Ovdje koristimo skalarni umnožak kao mjeru sličnosti i očekujemo da će slični vektori ostvariti veće ocjene.

5. Pretvaramo vektor ocjene u vjerojatnosti $\hat{y} = \text{softmax}(z) \in \mathbb{R}^{|V|}$.

6. Želimo da se dobivene vjerojatnosti $\hat{y} \in \mathbb{R}^{|V|}$ podudaraju s točnim vjerojatnostima $y \in \mathbb{R}^{|V|}$, što je one-hot vektor središnje riječi.

Model će raditi na ovaj način, kada budemo imali $\mathcal{U}$ i $\mathcal{V}$, ali kako izračunati te dvije matrice? Definirat ćemo funkciju cilja. Kada želimo naučiti neku vjerojatnosnu distribuciju, pogledamo u teoriju informacija da bismo našli mjeru udaljenosti između dvije distribucije. Ovdje ćemo koristiti unakrsnu entropiju $H(\hat{y}, y)$.

Unakrsna entropija

Unakrsna entropija dolazi iz teorije informacija. Prema teoriji koju je iznio Claude Shanon[a], prenošenje 1 bita informacije predstavlja smanjenje neizvjesnosti primatelja za faktor 2. Neizvjesnost je recipročna vrijednost vjerojatnosti danog događaja. Pretpostavimo da želimo efikasno prenijeti informaciju o vremenu svaki dan. Ukoliko postoje 2 jednako vjerojatne mogućnosti (sunčano, kiša), informaciju o vremenu bismo mogli kodirati u 1 bit. Kada dobijemo informaciju o vremenu, naša neizvjesnost se smanjila za faktor $\frac{1}{0.5} = 2$. Broj bitova korisnih informacija koje su prenesene je logaritam od faktora neizvjesnosti, u ovom slučaju $\log_2 2 = 1$.

[13] U praksi, dimenzija vektora riječi je između 300 i 500. Pokazuje se da nakon ovog broja nema značajnog poboljšanja u kvaliteti vektora. Korištenjem premale dimenzije za vektorski prostor nećemo moći obuhvatiti sva različita značenja svih riječi iz skupa za treniranje.

Što se događa kada nisu sve mogućnosti jednako vjerojatne? Pretpostavimo da živimo u sunčanom predjelu gdje je vjerojatnost sunčanog vremena 75%, a kiše 25%. Ako dobijemo informaciju da je kiša, naša neizvjesnost se smanjuje za faktor $\frac{1}{0.25} = 4$ ($\log_2 4 = 2$ bita informacije). Ako dobijemo informaciju da je sunčano, naša neizvjesnost se nije jako smanjila, faktor $\frac{1}{0.75} = 1.\dot{3}$ ($\log_2 1.\dot{3} \approx 0.42$ bita informacije). Broj informacija koje ćemo dobivati, u prosjeku, jest $0.42 \cdot 0.75 + 2 \cdot 0.25 = 0.815$ bita. Ovaj broj se zove **entropija**. Formalnije, entropija (diskretne) distribucije p je

$$H(p) = \sum_i p_i \log \frac{1}{p_i} = -\sum_i p_i \log p_i$$

Taj broj je mjera neizvjesnosti događaja tj. prosječni broj bitova informacija koji dobijemo iz jednog uzorka koji dolazi iz vjerojatnosne distribucije p. Govori nam koliko je ta distribucija nepredvidljiva.

Ukoliko ima 8 mogućnosti (sunčano, oblačno, kišno, itd.), svaku mogućnost bismo mogli kodirati koristeći 3 bita, jer ukupno ima $2^3 = 8$ mogućnosti. Ipak, kada bismo znali da je većinu vremena sunčano, mogli bismo tu informaciju kodirati s jednim bitom (0), a ostalih 7 opcija na 4 bita ($1***$). Time, implicitno stvaramo pretpostavku o distribuciji: ako kodiramo mogućnost s k bita, vjerojatnost te mogućnosti je $\frac{1}{2^k}$. Primjerice,

$$\text{stvarna distribucija: } p = \left(\tfrac{80}{100} \;\; \tfrac{5}{100} \;\; \tfrac{5}{100} \;\; \tfrac{1}{100} \;\; \tfrac{1}{100} \;\; \tfrac{1}{100} \;\; \tfrac{1}{100} \;\; \tfrac{1}{100}\right)$$
$$\text{pretpostavljena distribucija: } q = \left(\tfrac{8}{16} \;\; \tfrac{1}{16} \;\; \tfrac{1}{16} \;\; \tfrac{1}{16} \;\; \tfrac{1}{16} \;\; \tfrac{1}{16} \;\; \tfrac{1}{16} \;\; \tfrac{1}{16}\right)^b$$

Unakrsna entropija (diskretnih) distribucija p i q je

$$H(p, q) = \sum_i p_i \log \frac{1}{q_i} = -\sum_i p_i \log q_i$$

Ona mjeri prosječan broj bitova koji šaljemo. U našem primjeru, $H(p, q) \approx 1.4$, $H(p) \approx 1.02$. Ukoliko je naša pretpostavka o vremenu savršena, unakrsna entropija će biti jednaka samoj entropiji vremena. Ako je naša pretpostavka o vremenu netočna (npr. često pada kiša), unakrsna entropija će se povećati za broj zvan *Kullback-Leibler divergencija*, poznat i kao relativna entropija:

$$D_{KL}(p||q) = H(p, q) - H(p)$$

[a] Claude Elwood Shannon (1916 - 2001), bio je američki matematičar, inženjer elektrotehnike, i kriptograf poznat kao "otac teorije informacija".

[b] Uočimo da koristimo samo 7 od 8 opcija za kodove oblika 1***, pa suma vjerojatnosti nije jednaka 1.

U našem slučaju, funkcija cilja bit će:

$$H(\hat{y}, y) = -\sum_{j=1}^{|V|} y_j \log(\hat{y_j})$$

Ako je y one-hot vektor, funkcija se pojednostavljuje u:

$$H(\hat{y}, y) = -y_i \log(\hat{y_i})$$

gdje je i indeks jedinice u one-hot vektoru.

Ako je naš model dobar i dobijemo $\hat{y}_c = 1$, $H(\hat{y}, y) = -\log(1) = 0$, stoga nema kazne za ovo predviđanje. S druge strane, ako je model loš i dobijemo $\hat{y}_c = 0.01$, $H(\hat{y}, y) = -\log(0.01) \approx 4.605$. Vidimo da unakrsna

entropija pruža dobru mjeru za udaljenosti vjerojatnosnih distribucija. Stoga, definiramo funkciju cilja s:

$$\begin{aligned}\min J &= -\log P(w_c|w_{c-m},\dots,w_{c-1},w_{c+1},\dots,w_{c+m}) \\ &= -\log P(u_c|\hat{v}) \\ &= -\log \frac{\exp(u_c^T\hat{v})}{\sum_{j=1}^{|V|}\exp(u_j^T\hat{v})} \\ &= -u_c^T\hat{v} + \log\sum_{j=1}^{|V|}\exp(u_j^T\hat{v})\end{aligned}$$

Koristimo stohastički gradijentni spust kako bismo ažurirali vektore u_j i v_j.

2.3.3 Skip-gram model

Drugi pristup je napraviti model koji će za danu središnju riječ *"zalajao"*, dobro predviđati okolne riječi {*"Pas"*, *"je"*, *"vidjevši"*, *"mačku"*}. Postava je slična kao u CBOW modelu, ali su x i y zamijenjeni. Ulazni one-hot vektor (središnja riječ) je označen s x. Izlazni vektori su označeni s $y^{(j)}$. Definiramo $\mathcal{V}$ i $\mathcal{U}$ isto kao u CBOW modelu.
Model radi na sljedeći način:

1. Generiramo one-hot vektor $x \in \mathbb{R}^{|V|}$ za središnju riječ.
2. Dohvatimo vektor riječi za središnju riječ $v_c = \mathcal{V}x \in \mathbb{R}^n$
3. Definiramo vektor ocjene $z = \mathcal{U}v_c \in \mathbb{R}^{|V|}$.
4. Pretvaramo vektor ocjene u vjerojatnosti $\hat{y} = \text{softmax}(z) \in \mathbb{R}^{|V|}$.
   Uočimo da su $\hat{y}_{c-m},\dots,\hat{y}_{c-1},\hat{y}_{c+1},\hat{y}_{c+m}$ vjerojatnosti za okolne riječi.
5. Želimo da se dobiveni vektor vjerojatnosti podudara s točnim vjerojatnostima $y^{(c-m)},\dots,y^{(c-1)},y^{(c+1)},\dots,y^{(c+m)}$, one-hot vektorima susjednih riječi.

Kao i CBOW modelu, trebamo funkciju cilja kojom ćemo ocijeniti model. Kako bismo se riješili uvjetnih vjerojatnosti, uvest ćemo naivnu Bayesovu pretpostavku. Naime, pretpostavit ćemo da su za danu središnju riječ, izlazne riječi međusobno nezavisne.

$$\begin{aligned}\min J &= -\log P(w_{c-m},\dots,w_{c-1},w_{c+1},w_{c+m}|w_c) \\ &= -\log \prod_{j=0,j\neq m}^{2m} P(w_{c-m+j}|w_c) \\ &= -\log \prod_{j=0,j\neq m}^{2m} P(u_{c-m+j}|v_c) \\ &= -\log \prod_{j=0,j\neq m}^{2m} \frac{\exp(u_{c-m+j}^T v_c)}{\sum_{k=1}^{|V|}\exp(u_k^T v_c)} \\ &= -\sum_{j=0,j\neq m}^{2m} u_{c-m+j}^T v_c + 2m\log\sum_{k=1}^{|V|}\exp(u_k^T v_c)\end{aligned}$$

Možemo izračunati gradijent ove funkcije, i minimizirati ju koristeći stohastički gradijentni spust kako bismo izračunali tražene parametre.

Uočimo da je:

$$\begin{aligned} J &= -\sum_{j=0, j\neq m}^{2m} \log P(u_{c-m+j}|v_c) \\ &= \sum_{j=0, j\neq m}^{2m} H(\hat{y}, y_{c-m+j}) \end{aligned}$$

gdje je $H(\hat{y}, y_{c-m+j})$ unakrsna entropija između vjerojatnosti $\hat{y}$ i one-hot vektora y_{c-m+j}. Dakle, kod ovog modela računamo samo jedan vektor vjerojatnosti $\hat{y}$. Skip-gram tretira svaku riječ iz konteksta jednako, bez obzira koliko je daleko od središnje riječi.

2.3.4 Negativno uzorkovanje

Svratimo li pogled na funkciju cilja kod CBOW i skip-gram modela, uočit ćemo da postoji sumiranje po $|V|$, što je vrlo skupo za računati kako je riječ o ogromnoj bazi riječi. Ne možemo si dozvoliti da svako ažuriranje funkcije cilja zahtijeva $O(|V|)$ operacija. Jedno jednostavno rješenje je aproksimirati taj izraz.

Tijekom treniranja, osim što ćemo ažurirati vektore za riječi iz trenutnog primjera (središnja riječ / okolina), trebamo ažurirati i vektore svih drugih riječi iz rječnika. Umjesto toga, popravit ćemo samo vektore nekoliko[14] negativnih uzoraka. Kako je cilj dobro modeliranje jezika, važnije je izračunati kvalitetne vektore za primjere na kojima učimo, nego smanjiti zbunjenost sustava za besmislene primjere.

Negativno uzorkovanje je bazirano na skip-gramu, ali zapravo optimizira drugu funkciju cilja. Razmotrimo par (w, c) riječi i konteksta. Pitamo se dolazi li ovaj par iz korpusa nad kojim treniramo ili ne. Označimo s $P(D = 1|w, c)$ vjerojatnost da (w, c) dolazi iz korpusa. Slično, s $P(D = 0|w, c)$ vjerojatnost da (w, c) ne dolazi iz korpusa. Modelirat ćemo $P(D = 1|w, c)$ sigmoid funkcijom.

$$P(D = 1, w, c, \theta) = \sigma(v_c^T v_w) = \frac{1}{1 + \exp(-v_c^T v_w)}$$

Definiramo novu funkciju cilja koja će maksimizirati vjerojatnost da su riječ i njezin kontekst u korpusu, ako zaista jesu te vjerojatnost da riječ i kontekst nisu u korpusu, ako zaista nisu. Koristit ćemo pristup maksimalne vjerodostojnosti (eng. *maximum likelihood*) za ove dvije vjerojatnosti. θ je parametar modela: u našem slučaju su to $\mathcal{V}$ i $\mathcal{U}$.

$$\begin{aligned} \theta &= \underset{\theta}{\text{argmax}} \prod_{(w,c)\in D} P(D = 1|w, c, \theta) \prod_{(w,c)\in \tilde{D}} P(D = 0|w, c, \theta) \\ &= \underset{\theta}{\text{argmax}} \prod_{(w,c)\in D} P(D = 1|w, c, \theta) \prod_{(w,c)\in \tilde{D}} (1 - P(D = 1|w, c, \theta)) \\ &= \underset{\theta}{\text{argmax}} \sum_{(w,c)\in D} \log P(D = 1|w, c, \theta) + \sum_{(w,c)\in \tilde{D}} \log(1 - P(D = 1|w, c, \theta)) \\ &= \underset{\theta}{\text{argmax}} \sum_{(w,c)\in D} \log \frac{1}{1 + \exp(-u_w^T v_c)} + \sum_{(w,c)\in \tilde{D}} \log\left(1 - \frac{1}{1 + \exp(-u_w^T v_c)}\right) \\ &= \underset{\theta}{\text{argmax}} \sum_{(w,c)\in D} \log \frac{1}{1 + \exp(-u_w^T v_c)} + \sum_{(w,c)\in \tilde{D}} \log \frac{1}{1 + \exp(u_w^T v_c)} \end{aligned}$$

[14] Prema [9] odabir 5-20 riječi je dobar za manje baze tekstova, dok je za velike dovoljno 2-5 riječi.

Kako za proizvoljnu funkciju f vrijedi $\max(f) = \min(-f)$, nova funkcija cilja jest:

$$J = -\sum_{(w,c)\in D} \log \frac{1}{1+\exp(-u_w^T v_c)} - \sum_{(w,c)\in \tilde{D}} \log \frac{1}{1+\exp(u_w^T v_c)}$$

Uočimo da je $\tilde{D}$ negativni korpus. Sastoji se od izmišljenih rečenica poput *"Siromah san je šapnuo prozor."*. Besmislene rečenice trebaju imati minimalnu vjerojatnost pojavljivanja u korpusu. Njih možemo uvijek generirati uzimajući nasumične riječi iz rječnika.

Za skip-gram, promatrajući riječ $c-m+j, j = 0,\ldots,2m, j \neq m$ iz konteksta za danu središnju riječ, nova funkcija cilja je (2.1) u usporedbi sa starom (2.2):

$$-\log\sigma(u_{c-m+j}^T \cdot v_c) - \sum_{k=1}^{K} \log\sigma(-\tilde{u}_k^T \cdot v_c) \qquad (2.1)$$

$$-u_{c-m+j}^T v_c + \log\sum_{k=1}^{|V|} \exp(u_k^T v_c) \qquad (2.2)$$

Slično, za CBOW, promatrajući središnju riječ u_c za dani vektor konteksta $\hat{v} = \frac{v_{c-m}+v_{c-m+1}+\cdots+v_{c+m}}{2m}$ nova funkcija cilja bi bila (2.3) u usporedbi sa starom (2.4):

$$-\log\sigma(u_c^T \cdot \hat{v}) - \sum_{k=1}^{K} \log\sigma(-\tilde{u}_k^T \cdot \hat{v}) \qquad (2.3)$$

$$-u_c^T \hat{v} + \log\sum_{k=1}^{|V|} \exp(u_k^T \hat{v}) \qquad (2.4)$$

U navedenim izrazima, $\{\tilde{u}_k,\ k = 1,\ldots,K\}$ su uzorci iz $P_n(w)$, unarnog modela jezika, gdje je vjerojatnije da ćemo odabrati riječi koje se češće pojavljuju u korpusu. Primjerice, vjerojatnost da odaberemo riječ *mačka* bi bila jednaka omjeru broja pojavljivanja te riječi u korpusu i broja riječi u korpusu. Formalnije,

$$P_n(w_i) = \frac{f(w_i)}{\sum_{j=0}^{n} f(w_j)} \qquad (2.5)$$

gdje je $f(w_i)$ broj pojavljivanja (frekvencija) riječi w_i u korpusu. Nakon isprobavanja različitih mogućnosti, prema [9], ispostavlja se da je puno bolje potencirati frekvencije na $\frac{3}{4}$ tj.

$$\hat{P}_n(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n} f(w_j)^{3/4}} \qquad (2.6)$$

Na taj način, rjeđe riječi će se pojavljivati malo češće, dok će se češće riječi pojavljivati malo rjeđe.

Primjer 2.5. (Izmijenjeni unarni model za odabir negativnih uzoraka)
Kako bismo ilustrirali razlike između izraza 2.5 i 2.6, konstruirat ćemo mali primjer. Neka je rječnik za ovaj primjer skup svih riječi koje se pojavljuju u romanu Augusta Šenoe: Zlatarevo Zlato. Ukupno ima 80238 riječi. Pogledajmo koje vrijednosti poprimaju odabrane riječi iz teksta.

$\mathbf{w_i}$	je	ne	ban	zlato	kosa	pečat
$\mathbf{f(w_i)}$	2147	834	117	27	12	3
$\mathbf{P_n(w_i)}$	0.02676	0.01039	0.00146	0.00034	0.00015	0.00004
$\mathbf{\hat{P}_n(w_i)}$	0.00815	0.00401	0.00092	0.00031	0.00017	0.00006
$\mathbf{P_n(w_i)/\hat{P}_n(w_i)}$	3.28	2.59	1.59	1.10	0.90	0.63

Tablica 2.3: Odnos unarnog i izmijenjenog unarnog modela za odabrane riječi (vrijednosti su zaokružene na 2 ili 5 decimala, ovisno o retku)

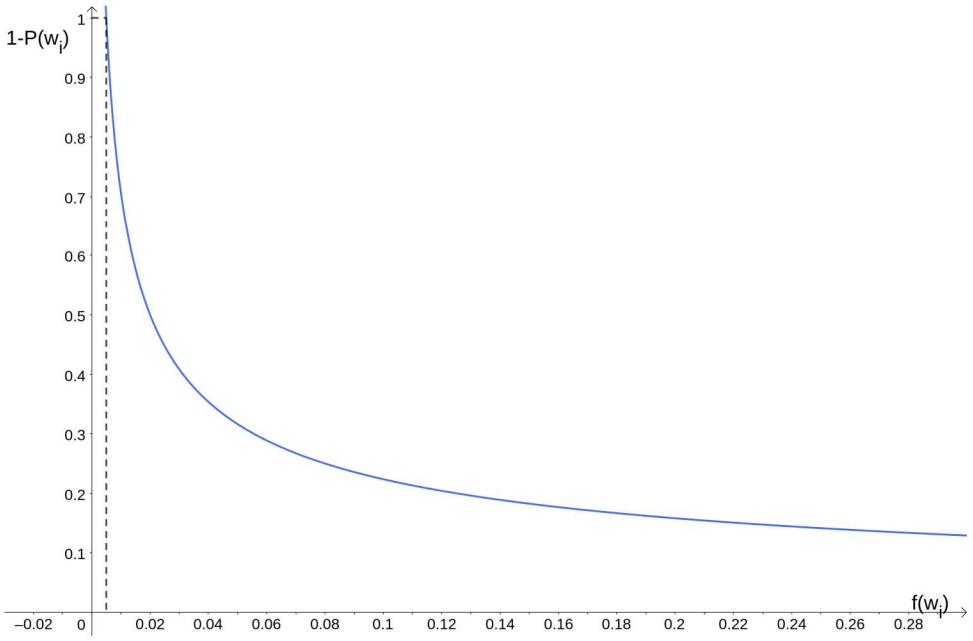
2.3.5 Poduzorkovanje čestih riječi

U jako velikim korpusima, česte riječi se pojavljuju na stotine milijuna puta. U hrvatskom jeziku, to su nepunoznačnice poput *i*, *u*, *je*, *se*, *na*, *da*, *za*, *od* itd. Takve riječi nose puno manje informacije nego rijetke riječi. Za treniranje skip-gram modela je korisnije pronaći susjedne riječi *dobar* i *učenik* nego riječi *dobar* i *je*. Također, riječ *je* se nalazi uz jako puno riječi u rečenici. Ipak, vektorska reprezentacija čestih riječi se ne mijenja značajno nakon treniranja na nekoliko milijuna primjera.

Kako bismo se nosili s ovom pojavom, u [9] koriste poduzorkovanje riječi na način da se svaka riječ u skupu za treniranje odbacuje s vjerojatnosti

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

gdje je $f(w_i)$ frekvencija riječi i t unaprijed definirani parametar (često oko 10^{-5}). Manje vrijednosti parametra ukazuju da će se više riječi odbacivati.



Slika 2.5: Vjerojatnost *zadržavanja* riječi u skupu za treniranje za $t = 0.005$. Riječi s frekvencijom $f(w_i) \leq t$ se sigurno ne odbacuju iz korpusa.

Poduzorkovanje eliminira riječi s velikim frekvencijama, dok čuva one s manjim frekvencijama. Na taj način je učenje ubrzano i preciznost vektora riječi na rjeđim (punoznačnim) riječima je poboljšana.

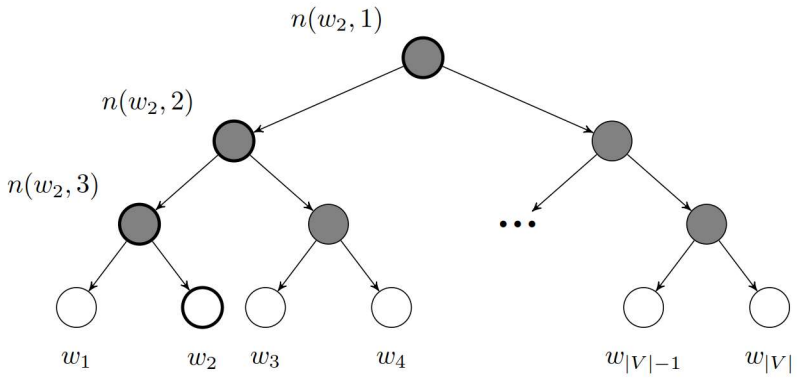
2.3.6 Hijerarhijski softmax

Prema [8], hijerarhijski softmax je predstavljen kao puno efikasnija alternativa uobičajenom softmax-u. U praksi, hijerarhijski softmax je skloniji boljem učinku kod riječi koje se rjeđe pojavljuju, dok negativno uzorkovanje bolje radi kod češćih riječi i manjih dimenzija vektora riječi.

Hijerarhijski softmax koristi binarno stablo za prikaz riječi u rječniku. Svaki list u stablu je jedna riječ te postoji jedinstven put od korijena do nje. U ovom modelu, nema *izlazne reprezentacije riječi* (ranije, to je bila matrica $\mathcal{U}$). Ovdje, svaki čvor (osim listova) ima vektor koji će model naučiti.

U ovom modelu, vjerojatnost riječi w za dani vektor riječi w_i, $P(w|w_i)$, jednaka je vjerojatnosti slučajne

šetnje koja započinje u korijenu i završava u listu koji pripada w. Glavna prednost ovog pristupa je da traženu vjerojatnost možemo izračunati u $O(\log(|V|))$ operacija, prema odgovarajućoj duljini puta.



Slika 2.6: Binarno stablo za hijerarhijski softmax

Označimo s $L(w)$ broj čvorova na putu od korijena do lista w. Primjerice, $L(w_2)$ na slici 2.6 je 3. Označimo s $n(w, i)$ i-ti čvor na tom putu i njemu pridruženi vektor s $v_{n(w,i)}$. Tada je $n(w, 1)$ korijen stabla, a $n(w, L(w))$ roditelj od w. Za svaki od unutrašnjih čvorova n, slučajno odaberemo jedno dijete i označimo ga s $ch(n)$. Tada, možemo vjerojatnost riječi w za dani vektor riječi w_i izračunati s:

$$P(w|w_i) = \prod_{j=1}^{L(w)-1} \sigma([n(w, j+1) = ch(n(w, j))] \cdot v_{n(w,j)}^T v_{w_i})$$

gdje je

$$[x] = \begin{cases} 1, & \text{ako je } x \text{ istina} \\ -1, & \text{inače} \end{cases}$$

Proučimo malo prethodnu formulu: najprije računamo umnožak po putu od korijena ($n(w, 1)$) do lista (w). Pretpostavimo li da je $ch(n)$ uvijek lijevo dijete od n, izraz $[n(w, j+1) = ch(n(w, j))]$ će biti jednak 1 kada put vodi lijevo, a -1 kada vodi desno.

Također, izraz $[n(w, j+1) = ch(n(w, j))]$ donosi normalizaciju. Naime, u čvoru n, ako zbrojimo vjerojatnost odlaska lijevo i desno, za proizvoljnu vrijednost $v_n^T v_{w_i}$, imamo:

$$\sigma(v_n^T v_{w_i}) + \sigma(-v_n^T v_{w_i}) = 1$$

Ta normalizacija osigurava da $\sum_{w=1}^{|V|} P(w|w_i) = 1$, kao i originalnom softmax-u.
Konačno, uspoređujemo sličnost danog ulaznog vektora v_{w_i} s vektorom svakog unutrašnjeg čvora $v_{n(w,j)}^T$ koristeći skalarni produkt. Primjerice, za w_2 na slici 2.6, trebamo dva puta ići lijevim bridom i jednom desnim da dođemo u w_2. Stoga,

$$\begin{aligned} P(w_2|w_i) &= p(n(w_2, 1), \text{lijevo}) \cdot p(n(w_2, 2), \text{lijevo}) \cdot p(n(w_2, 3), \text{desno}) \\ &= \sigma(v_{n(w_2,1)}^T v_{w_i}) \cdot \sigma(v_{n(w_2,2)}^T v_{w_i}) \cdot \sigma(-v_{n(w_2,3)}^T v_{w_i}) \end{aligned}$$

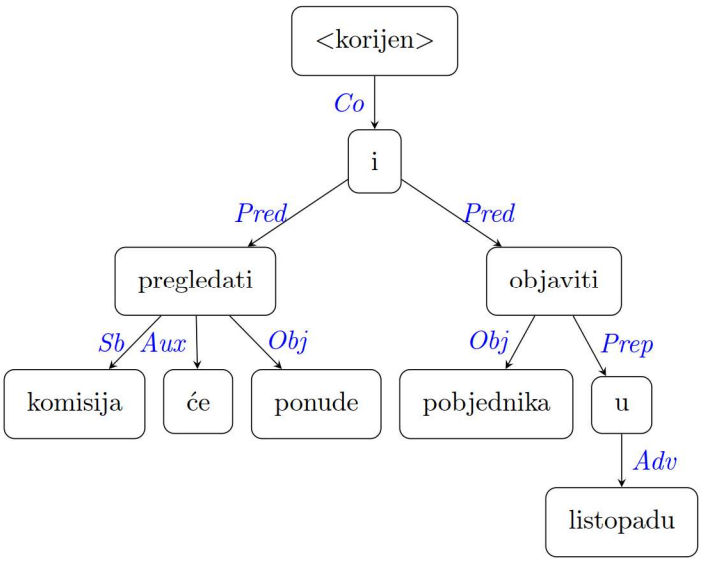
Kako bismo trenirali model, i dalje želimo minimizirati $-\log P(w|w_i)$. Ovdje, umjesto izlaznih vektora, ažuriramo vektore čvorova u binarnom stablu na putu od korijena do lista.

Brzina ove metode ovisi o načinu na koje je binarno stablo konstruirano i kako su riječima pridruženi listovi. U [9] koriste binarno stablo s Huffmanovim kodom, koje češćim riječima pridružuje kraće putove u stablu, što rezultira bržim treniranjem.

3 Parsiranje međuovisnosti

Nakon što smo naučili što vektori riječi jesu i kako ih možemo izračunati, prirodni je nastavak proučiti gdje ih možemo koristiti. U ovom poglavlju ćemo predstaviti jednu njihovu primjenu: parsiranje međuovisnosti između riječi u rečenici. Riječ je o parserima koji nam daju uvid u unutarnju strukturu rečenice, koja može biti korisna u drugim primjenama (npr. razumijevanje ljudskih rečenica za govorno sučelje mobitela).

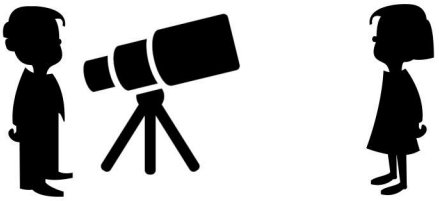
U obradi prirodnog jezika, kao kod prevoditelja (eng. *compiler*), stabla parsiranja se koriste za sintaktičku analizu rečenice. Struktura rečenice se može promatrati na različite načine. Jedan od glavnih načina jest promatrati kako riječ ovisi (mijenja ili nadopunjuje) druge riječi. Ova vrsta ovisnosti je binarna i asimetrična, označava se strelicom koja ide od nadređene riječi prema podređenoj riječi. Štoviše, svaka strelica ima na sebi oznaku koja ukazuje na vrstu međuovisnosti. U pravilu, ovakvi odnosi grade stablo kao na slici 3.1. Ponekad se dodaje fiktivni korijenski čvor (eng. *root*) da svaka riječ bude podređena točno jednoj riječi.



Slika 3.1: Stablo parsiranja za rečenicu *Komisija će pregledati ponude i objaviti pobjednika u listopadu.*

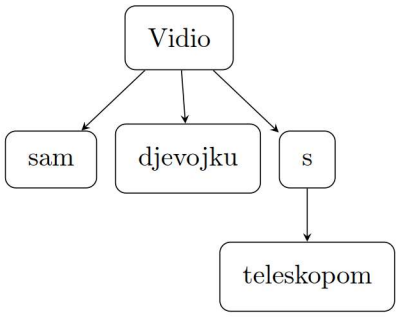
Možda se pitate zašto uopće trebamo strukturu rečenice. Razlog je pravilno tumačenje rečenice. Naime, ljudi u komunikaciji prenose kompleksne ideje slažući riječi u veće cjeline kako bi prenijeli složeno značenje. Zbog toga želimo znati što je s čime povezano.

Primjer 3.1. (Dvosmislenost)
Promotrimo sljedeću rečenicu: Vidio sam djevojku s teleskopom. *Ova rečenica se može tumačiti dvojako. Jedno moguće značenje jest da je djevojka viđena uz pomoć teleskopa (vidi slike 3.2, 3.4). Alternativno, viđena je djevojka koja posjeduje teleskop (vidi slike 3.3, 3.5).*

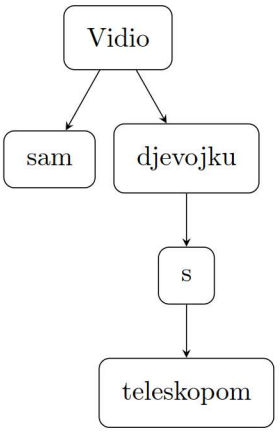


Slika 3.2: Ilustracija prvog načina

Slika 3.3: Ilustracija drugog načina



Slika 3.4: Prvo stablo parsiranja



Slika 3.5: Drugo stablo parsiranja

Zadatak parsiranja međuovisnosti jest analizirati sintaktičku strukturu zadane rečenice R. Izlaz parsiranja je stablo međuovisnosti koje prikazuje veze između riječi iz rečenice. Formalnije, zadatak je ulaznu rečenicu $R = w_0 w_1 \dots w_n$ (gdje je w_0 korijenski čvor) preslikati u graf stabla međuovisnosti G.

Generalno govoreći, pristupi rješavanju ovog zadatka se mogu podijeliti u dvije skupine:

- pristupi koji se temelje **na podacima** - koriste strojno učenje na označenim podacima kako bi naučili parsirati nove rečenice,
- pristupi koji se temelje **na gramatici** - koriste formalnu gramatiku (koja definira formalni jezik) i pitaju se podliježe li nova rečenica danoj gramatici ili ne.

U novije vrijeme, metode koje se temelje na podacima privlače najveću pozornost. Usredotočit ćemo se na metode nadziranog učenja tj. pretpostavit ćemo da su rečenice kojima smo učili sustav označene točnim odnosnima međuovisnosti. Prema [6], kod nadziranog učenja postoje dva različita problema koja treba riješiti:

- **Učenje:** za dani skup rečenica za treniranje D (s označenim međuovisnostima), konstruirati model parsiranja M koji se može koristiti za parsiranje novih rečenica.
- **Parsiranje:** za dani model M i rečenicu R, izračunati optimalni graf međuovisnosti G za rečenicu S prema modelu M.

Pristupi koji se temelje na podacima razlikuju se u modelu parsiranja koji koriste, algoritmima za treniranje modela iz podataka te algoritmima za parsiranje novih rečenica iz modela. Ovdje ćemo razmotriti model koji se temelji na prijelazima (eng. *transition-based*). Takav model definira sustav prijelaza ili konačni automat koji rečenicu preslikava u pripadni graf. Problem učenja jest istrenirati model koji će dobro predviđati idući prijelaz (stanje), ukoliko je poznata povijest prijelaza. Problem parsiranja je konstruirati optimalni niz prijelaza za ulaznu rečenicu, uz dani model. Ova metoda je poznata i kao parser koji izvodi pomak i redukciju (eng. *shift-reduce parser*) na temelju definirane gramatike. Ipak, većina modela koji se temelje na prijelazima ne koriste formalnu gramatiku.

3.1 Pohlepni deterministički pristup

Predstavit ćemo pohlepni deterministički pristup temeljen na prijelazima. Ideju koju je Nivre predstavio 2004. godine u [10] kombinirat ćemo s neuronskom mrežom.

Stanja:

Za danu rečenicu $R = w_0 w_1 \dots w_n$, stanje se može opisati kao uređena trojka (σ, β, A):

1. stog σ riječi w_i iz R,
2. polje β riječi w_i iz R,
3. skup međuovisnosti A oblika (w_i, r, w_j), gdje su $w_i, w_j \in R$, a r predstavlja odnos između njih.

Za svaku rečenicu $R = w_0 w_1 \dots w_n$:

1. početno stanje c_0 je oblika $([w_0]_\sigma, [w_1, \dots, w_n]_\beta, \emptyset)$ - samo je $<korijen>$ na stogu σ, sve ostale riječi su u polju β i nijedan odnos nije još definiran,
2. završno stanje je oblika $(\sigma, [\,]_\beta, A\,)$.

Prijelazi:

Postoje tri vrste prijelaza između stanja:

1. `POMAK`: Uklanja prvu riječ iz polja i stavlja ju na vrh stoga. (Uz uvjet da je polje neprazno.)
2. `LIJEVI LUK`$_r$`:` Dodaje vezu oblika (w_j, r, w_i) u skup A, gdje je w_j riječ s vrha stoga, a w_i riječ iza nje. Ukloni w_i iz stoga. (Uz uvjet da stog ima barem dva elementa i w_i ne može biti $<korijen>$.)
3. `DESNI LUK`$_r$`:` Dodaje vezu oblika (w_i, r, w_j) u skup A, gdje je w_j riječ s vrha stoga, a w_i riječ iza nje. Ukloni w_j iz stoga. (Uz uvjet da stog ima barem dva elementa.)

Cilj modela koji ćemo definirati je predvidjeti niz prijelaza koji će početno stanje c_0 dovesti do završnog stanja. Kako je riječ o pohlepnom modelu, on pokušava predvidjeti prijelaz $P \in \{\texttt{POMAK}, \texttt{LIJEVI LUK}_r, \texttt{DESNI LUK}_r\}$ u svakom trenutku, na temelju parametara iz stanja $c = (\sigma, \beta, A)$.

Parametri:

U ovisnosti o željenoj složenosti modela, moguće je na različite načine definirati ulazne parametre za neuronsku mrežu. Najčešće se biraju neki od sljedećih:

- R_{word}: vektori riječi za neke riječi iz R s vrha stoga σ i polja β.
- R_{tag}: Part-of-Speech (POS) oznake za neke riječi iz R. POS oznake dolaze iz malog, diskretnog skupa: $\mathcal{P} = \{ADJ, ADV, DET, NOUN, VERB, \dots\}$.[15]
- R_{label} Oznake međuovisnosti za neke riječi iz R. Oznake dolaze iz malog, diskretnog skupa i opisuju vrstu odnosa između dvije riječi: $\mathcal{L} = \{aux, cc, conj, det, iobj, nsubj, obj, \dots\}$.[16]

Za svaku vrstu parametra, definira se preslikavanje s one-hot vektora toga parametra na d-dimenzionalnu vektorsku reprezentaciju istog. Matrica riječi za R_{word} je $E^w \in \mathbb{R}^{d \times N_w}$, gdje je N_w broj riječi u rječniku. Slično, matrice POS oznaka i oznaka međuovisnosti su $E^t \in \mathbb{R}^{d \times N_t}$ i $E^l \in \mathbb{R}^{d \times N_l}$, gdje su N_t i N_l, redom, broj različitih POS oznaka i broj različitih međuovisnosti. Konačno, označimo broj odabranih elemenata iz svakog pojedinog skupa parametara s n_{word}, n_{tag} i n_{label}.

[15] POS oznake predstavljaju kategorije riječi koje imaju slična gramatička svojstva. Popis svih POS oznaka se može naći na: `https://universaldependencies.org/u/pos/index.html`.

[16] Popis svih oznaka međuovisnosti se može naći na: `https://universaldependencies.org/u/dep/index.html`.

Primjer 3.2. (Odabir parametara)

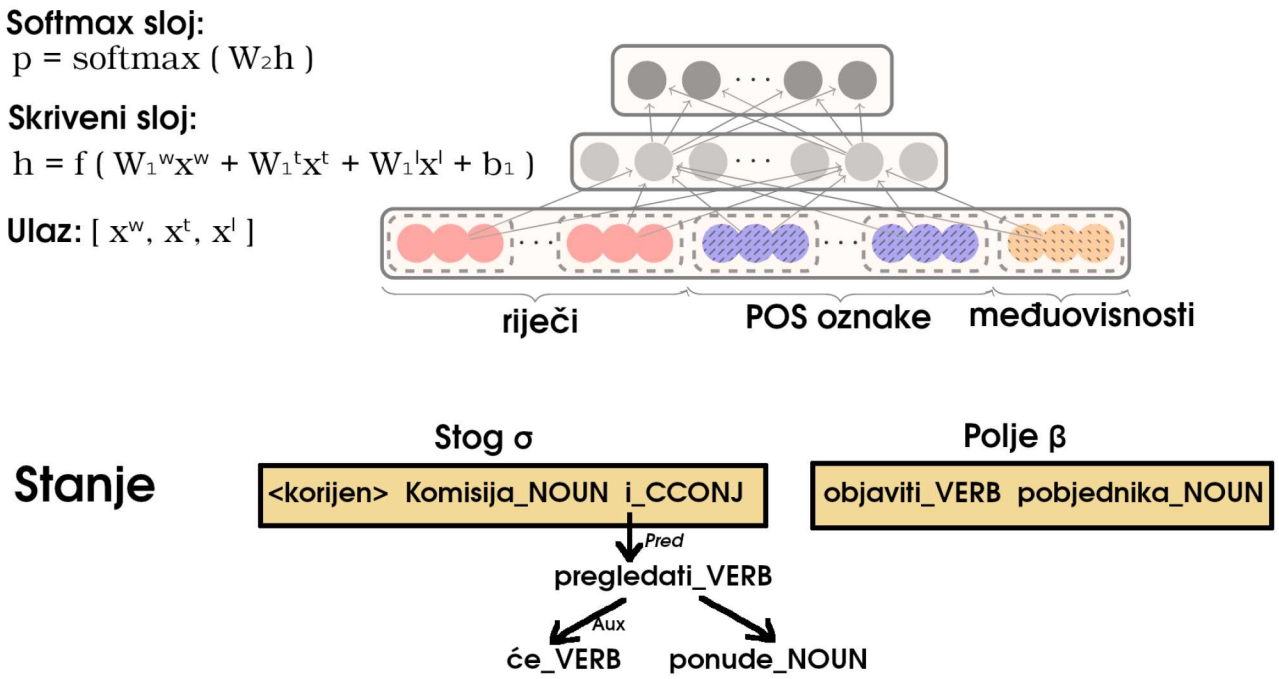
Pogledajmo jedan mogući odabir parametara za R_{word}, R_{tag} i R_{label}.

- *R_{word}: Prve tri riječi na stogu i u polju: $s_1, s_2, s_3, b_1, b_2, b_3$. Prvo i drugo najljevije / najdesnije dijete od prve dvije riječi na stogu: $lc_1(s_i), lc_2(s_i), rc_1(s_i), rc_2(s_i), i = 1, 2$. Najljevije dijete od najljevijeg djeteta i najdesnije dijete od najdesnijeg djeteta od prve dvije riječi na stogu: $lc_1(lc_1(s_i)), rc_1(rc_1(s_i)), i = 1, 2$. ($n_{word} = 18$)*
- *R_{tag}: Pripadne POS oznake za riječi iz R_{word}. ($n_{tag} = 18$)*
- *R_{label}: Pripadne oznake međuovisnosti za riječi iz R_{word}, osim 6 riječi sa stoga/polja. ($n_{label} = 12$)*

Za nepostojeće elemente (prazan stog ili polje, djeca još nisu određena) se koristi specijalna oznaka <NULL>. Za danu rečenicu, odaberu se riječi, POS oznake i oznake međuovisnosti prema gornjoj shemi, odrede se njihovi vektori parametara iz matrica vektorskih reprezentacija E^w, E^t, E^l i ti vektori se konkateniraju kao ulaz $[x^w, x^t, x^l]$ za neuronsku mrežu. Tijekom treniranja, greške se propagiraju unatrag kroz parametre modela do vektorskih reprezentacija.

Neuronska mreža:

Neuronska mreža se sastoji od ulaznog sloja $[x^w, x^t, x^l]$, skrivenog sloja i završnog softmax sloja s funkcijom cilja definiranom unakrsnom entropijom. Možemo primijeniti jednu matricu težina koja će raditi na konkateniranom ulazu $[x^w, x^t, x^l]$ ili koristiti tri matrice težina $[W_1^w, W_1^t, W_1^l]$, jednu za svaki ulazni vektor kao što je prikazano na slici 3.6. Nakon toga, primjenjujemo neku nelinearnu aktivacijsku funkciju f i koristimo još jednu linearnu transformaciju $[W_2]$ kako bi broj mogućnosti u softmaxu bio jednak broju različitih prijelaza (dimenzija izlaza).



Slika 3.6: Shematski prikaz neuronske mreže i trenutnog stanja

Za treniranje neuronske mreže se koriste rečenice s ručno označenim točnim međuovisnostima. Za svaku rečenicu se rekonstruiraju koraci kojima bi se moglo doći do konačnog stabla parsiranja. Dakle, skup za treniranje mreže jest $\{(s_i, p_i),\ i = 1, \ldots, m\}$, gdje su s_i trenutno stanje i p_i prijelaz koji vodi konačnom stablu parsiranja.

Nakon što se neuronska mreža trenira, parsiranje se vrši pohlepnim dekodiranjem. Za danu rečenicu, na svakom koraku se iz trenutnog stanja određuju vektori riječi, POS oznake i oznake međuovisnosti te se izračuna skriveni sloj h. Konačno, odabere se prijelaz kojemu završni sloj mreže daje najveću vjerojatnost.

Više informacija o ovakvom parseru možete pronaći u [3].

3.2 Primjeri

U novije vrijeme, počele su se pojavljivati označene baze tekstova. Izgradnja takvih baza je puno sporija i naizgled manje korisna od konstruiranja gramatike. Ipak, označene baze se koriste za konstrukciju parsera, POS označivača (eng. *tagger*) itd., daju informacije o frekvencijama i okolinama riječi te se mogu koristiti za ocjenu različitih sustava. Jedan takav vrijedan lingvistički resurs se pojavio i u Hrvatskoj: SETIMES.HR (vidi [1]). Ubrzo nakon toga, pojavila se i njegova nadogradnja (vidi [2]) koja sadrži 9012 rečenica (199441 riječ) koje su označene oznakama međuovisnosti, POS i drugim oznakama. Ta baza je korištena za treniranje parsera opisanih u nastavku.

U nastavku ćemo predstaviti trenutno najbolje poznate parsere međuovisnosti za hrvatski jezik. Jedan parser smo sami trenirali, koristeći spaCy biblioteku[17]. SpaCy je biblioteka za složeniju obradu prirodnog jezika implementirana u Cythonu[18]. Temelji se na najnovijim znanstvenim rezultatima i izgrađena je za industrijsku primjenu. Sama biblioteka ne daje modele za hrvatski jezik, ali nudi mogućnost treniranja koristeći standardne označene baze. Prema člancima[19] [20] jednog od autora biblioteke, spaCy koristi inačicu parsera temeljenom na prijelazima opisanog u ovom poglavlju.

S druge strane, postoji i StanfordNLP paket[21], napisan u Pythonu. Ovaj paket sadrži vrlo precizne komponente neuronskih mreža koje omogućuju učinkovito treniranje na vlastitim označenim podacima. StanfordNLP pruža podršku za mnoge ljudske jezike. Između ostalog, postoje i modeli za hrvatski jezik, također naučeni na proširenju SETimes-HR korpusa (vidi [2]). Znanstvenici sa Stanforda su ovdje izgradili novi pristup (vidi [11]) koji koristi LSTM (long short-term memory) rekurzivne neuronske mreže. Više o rekurzivnim neuronskim mrežama i LSTM ćelijama možete pronaći u [4].

Baza na kojoj su parseri trenirani je podijeljena u tri skupa: skup za **treniranje** (eng. *training*) koji sadrži 6844 rečenica, skup za **razvoj** (eng. *development*) koji sadrži 954 rečenica te skup za **testiranje** (eng. *test*) koji sadrži 1214 rečenica. Za ocjenu modela parsiranja međuovisnosti se koriste:

- UAS (eng. *unlabeled attachment score*) - postotak riječi koji imaju točnu nadređenu riječ. Ova ocjena se može koristiti jer svaka riječ ima točno jednu nadređenu riječ u stablu parsiranja, za razliku od drugih problema gdje gledamo različite ocjene točnosti (precision, recall, F-measure)
- LAS (eng. *labeled attachment score*) - postotak riječi koji imaju točnu nadređenu riječ i oznaku međuovisnosti

	trening skup	razvojni skup	testni skup
spaCy UAS	92.17	82.99	84.48
StanfordNLP UAS	94.97	90.52	91.24
spacy LAS	86.97	75.81	77.21
StanfordNLP LAS	91.90	85.75	86.07

Tablica 3.1: Ocjena spaCy i StanfordNLP modela

[17] Službena stranica: `https://spacy.io/`
[18] Službena stranica: `https://cython.org/`
[19] Matthew Honnibal: `https://explosion.ai/blog/parsing-english-in-python` (2014)
[20] Matthew Honnibal: `https://explosion.ai/blog/how-spacy-works` (2015)
[21] Službena stranica: `https://stanfordnlp.github.io/stanfordnlp/index.html`

LAS po rečenicama

Postoje i drugi načini za ocjenu parsiranih međuovisnosti. Može se koristiti i LAS po rečenicama kao osnovnoj jedinici. U ovoj inačici računamo koliko je točnih međuovisnosti u rečenici te računamo prosjek po rečenicama. Primjerice, pretpostavimo da testiramo dvije rečenice. Na prvoj smo točno označili svih 10 riječi, a na drugoj 16 od mogućih 40 riječi. Tada bi:

- LAS po riječima bio $(10 + 16)/(10 + 40) = 0.52$
- LAS po rečenicama bio $(10/10 + 16/40)/2 = 0.7$

Prema [6], na LAS po riječima možemo gledati kao *mikro-prosjek*, a na LAS po rečenicama kao *makro-prosjek* točno označenih riječi. Također, navodi da se za makro-prosjek može koristiti i prosječni LAS po različitim vrstama oznaka međuovisnosti.

U tablici 3.1 vidimo ocjene spaCy i Stanford NLP modela[22]. Uočimo da imaju veću preciznost na skupu za treniranje, ali da i dalje jako dobro rade na podacima koje nisu ranije vidjeli (testni skup).

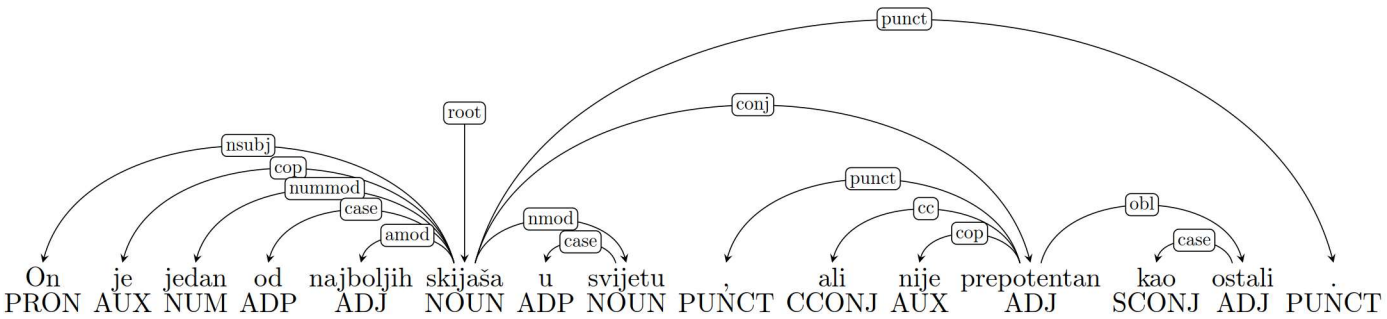
StanfordNLP parser ima izvanredan učinak a greške koje pravi su ponekada zanemarive važnosti (loša nadređena riječ interpunkciji ili riječima u složenom vlastitom imenu). Povremeno se dogodi da dobro prepozna međuovisnost ali dodijeli krivu oznaku, a u vrlo rijetkim slučajevima krivo spoji korijen riječi pa stablo bude upitne kvalitete. Možemo zaključiti da je StanfordNLP izvrstan i vrlo pouzdan parser koji se može koristiti u drugim primjenama.

Parser koji smo trenirali koristeći spaCy ima nešto lošiji učinak. On povremeno zastrani s izborom korijena i češće stavi krivu oznaku međuovisnosti nego StanfordNLP parser. Općenito, spaCy modeli na drugim jezicima imaju bolji učinak nego na hrvatskom jeziku. Za hrvatski jezik postoji samo jedna baza označenih lingvističkih podataka za ovaj zadatak i ovo je trenutno najbolje što iz nje možemo dobiti.

Možemo zaključiti da obje biblioteke daju izvrsne rezultate u parsiranju međuovisnosti. Ipak, čini se da StanfordNLP sa svojom složenijom arhitekturom postiže zapaženije rezultate.

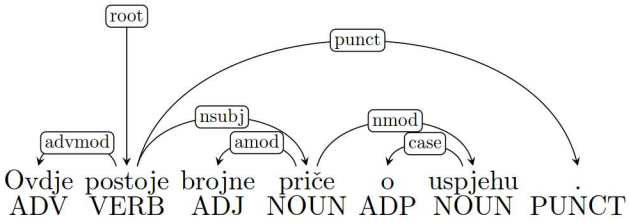
Do kraja poglavlja ćemo prikazivati primjere stabla parsiranja dobivenih s dva opisana parsera. U prvom retku teksta su riječi iz rečenice, a ispod svake riječi je pripadna POS oznaka. Iznad teksta se nalaze strelice koje označavaju međuovisnosti među riječima s pripadnim oznakama. Primjeri bez opisa su oni gdje je parser točno konstruirao stabla međuovisnosti. Parseri pokazuju odlične rezultate i na dužim, složenijim rečenicama ali ih ne možemo čitko prikazati zbog njihove duljine.

Slijede primjeri stabla parsiranja dobivenih sa StanfordNLP parserom.

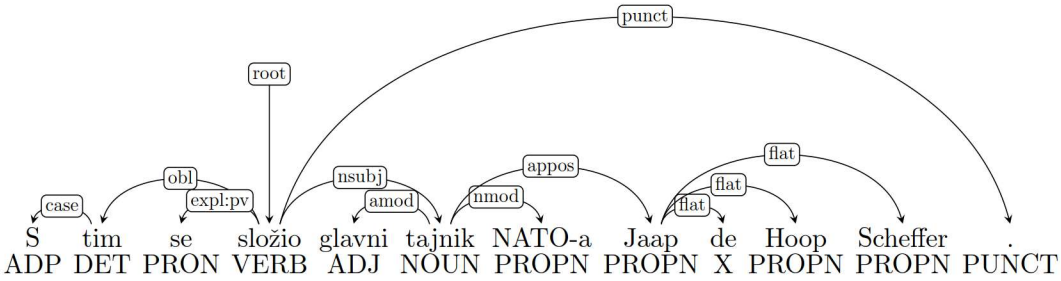


Slika 3.7

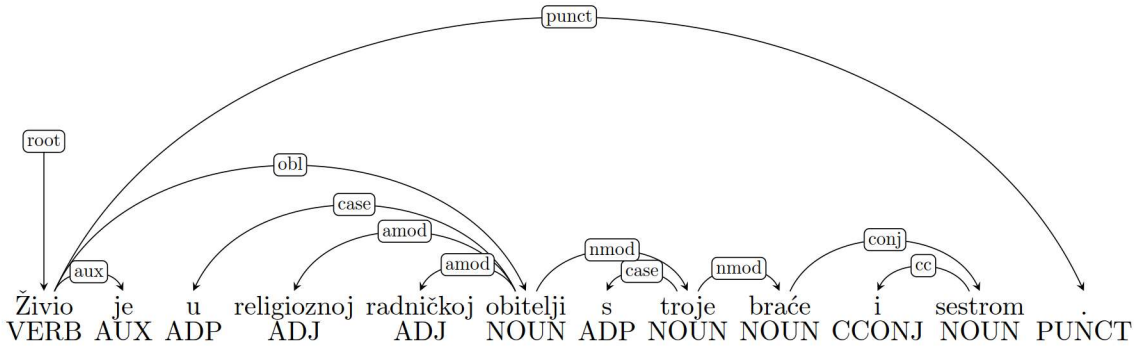
[22]spaCy model smo sami trenirali i testirali, dok smo za StanfordNLP samo testirali već gotovi model.



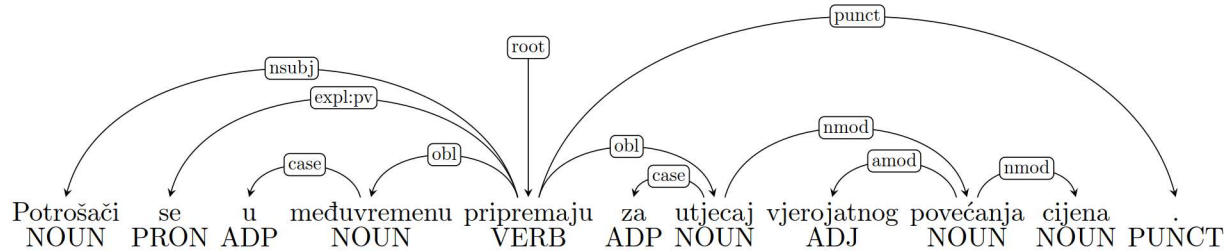
Slika 3.8



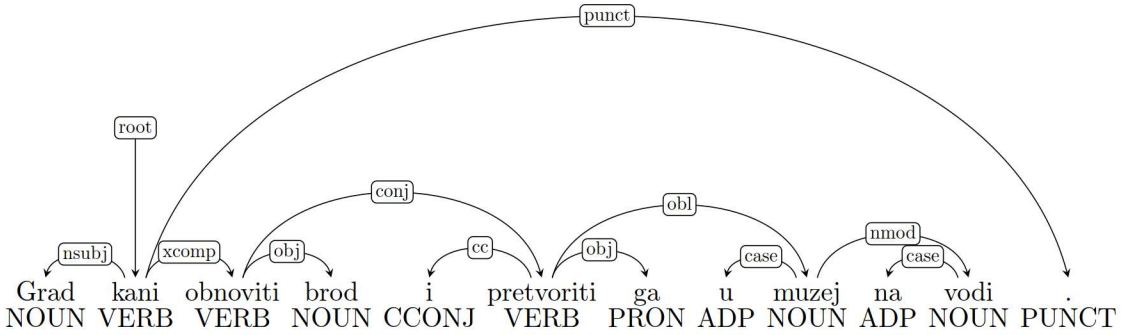
Slika 3.9: Ispravno je *Hoop* $\xrightarrow{case}$ *de*.



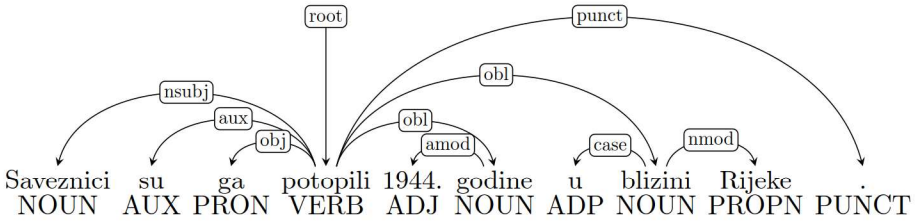
Slika 3.10: Ispravne međuovisnosti su: *braće* $\xrightarrow{case}$ *s*, *braće* $\xrightarrow{nummod}$ *troje*, *živio* $\xrightarrow{obl}$ *braće*.
U ovom primjeru je parser podstablo *s troje braće i sestrom* pridružio *obitelji*, dok u originalnoj rečenici to podstablo nadopunjuje predikat rečenice *živio je*. Također, parser je krivo prepoznao korijen tog podstabla.



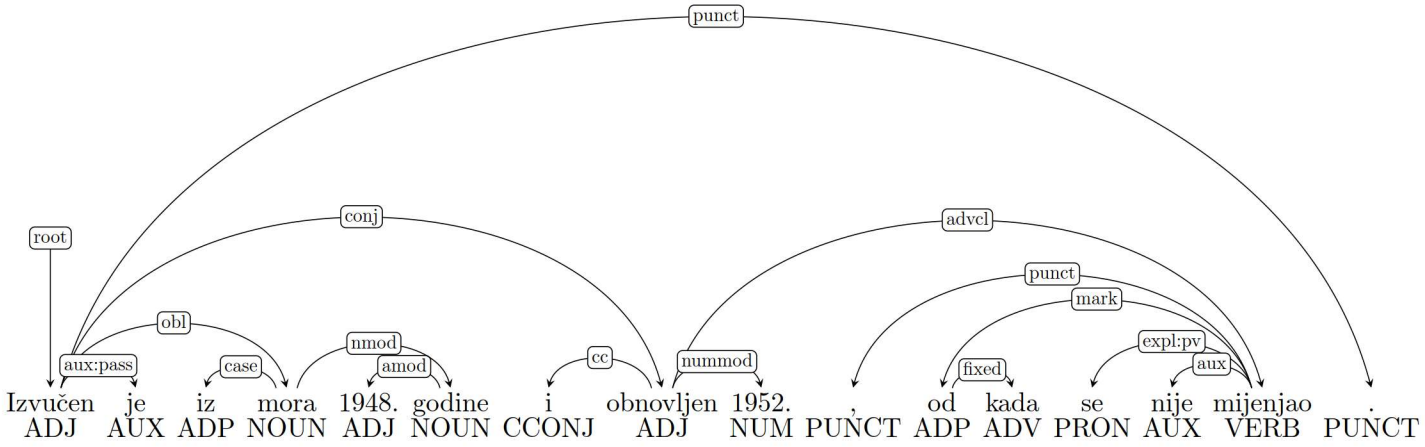
Slika 3.11



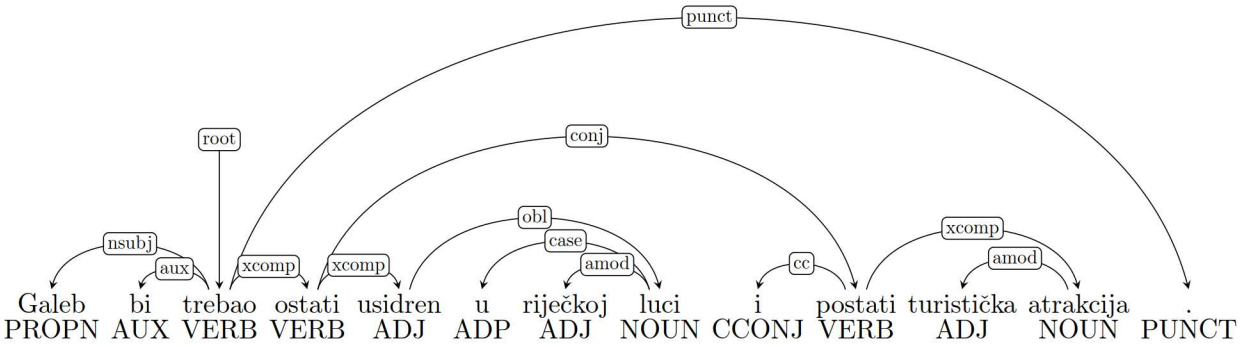
Slika 3.12



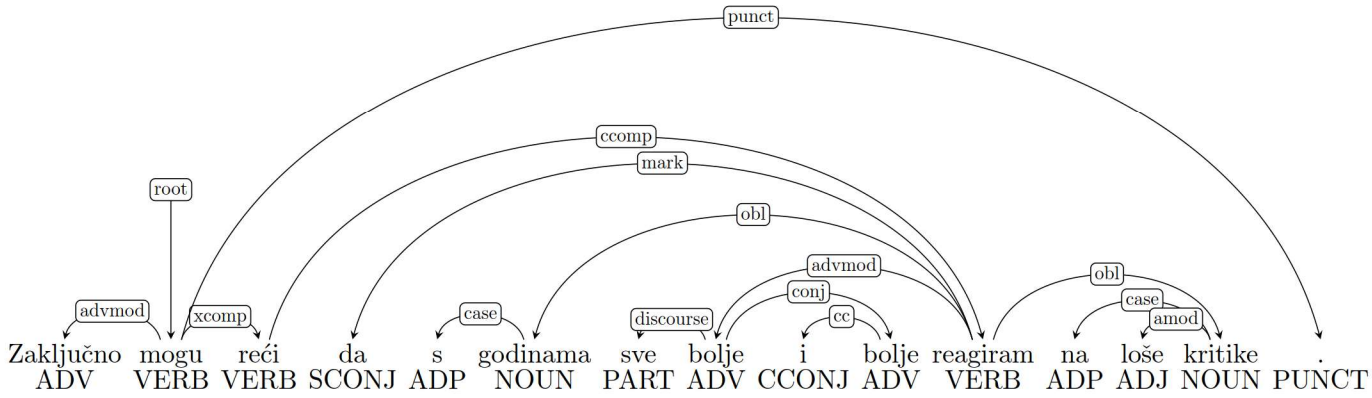
Slika 3.13



Slika 3.14: Ispravno je: *izvučen* $\xrightarrow{obl}$ *godine* jer *1948. godine* ne opisuje *mora* već godinu izvlačenja. Osim promašene oznake *od* $\xrightarrow{compound}$ *kada*, trebalo je označiti i *godine* $\xrightarrow{conj}$ *1952.*. Ispravno stablo u ovom slučaju ima bridove koji se sijeku (eng. *non-projective*). Takva stabla nastaju kada je neka riječ postavljena na neuobičajenom mjestu. Parseri temeljeni na prijelazima, uz trenutačne prijelaze, ne mogu prepoznavati takva stabla. Problem parsiranja rečenica s bridovima koji se sijeku se rjeđe rješava, a za to se često koriste parseri temeljeni na grafovima (eng. *graph-based parsers*).

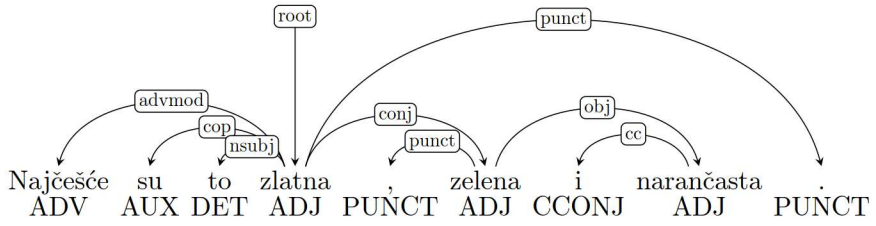


Slika 3.15: Ispravno je *ostati* $\xrightarrow{obl}$ *luci* jer *u riječkoj luci* posredno opisuje gdje će brod *ostati*, dok *usidren* opisuje način na koji će brod *ostati* i rečenica bi mogla stajati i bez te riječi. Također, točna međuovisnost je i *postati* $\xrightarrow{case}$ *i*.

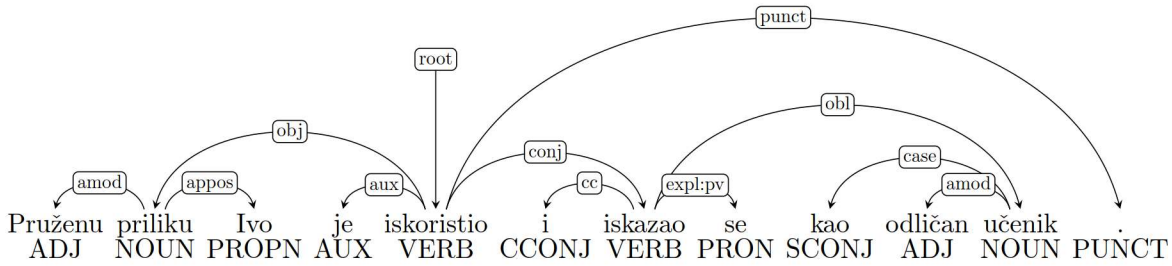


Slika 3.16: Ovaj primjer su točno parsirali oba parsera.

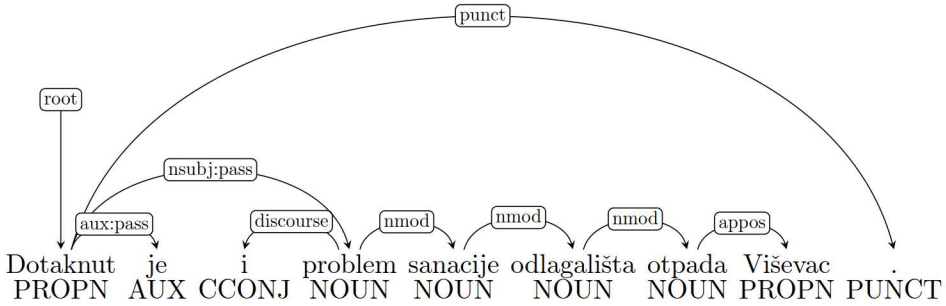
Slijede primjeri stabla parsiranja dobivenih s spaCy parserom.



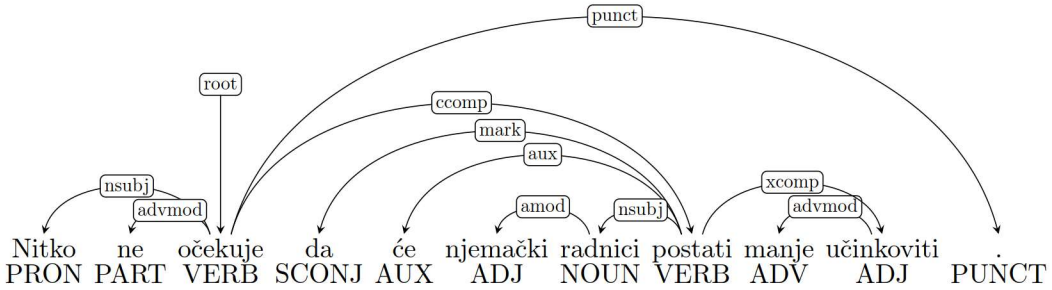
Slika 3.17: Ispravno je *zlatna* $\xrightarrow{conj}$ *narančasta*, slično kao što je točno označio riječ *zelena*.



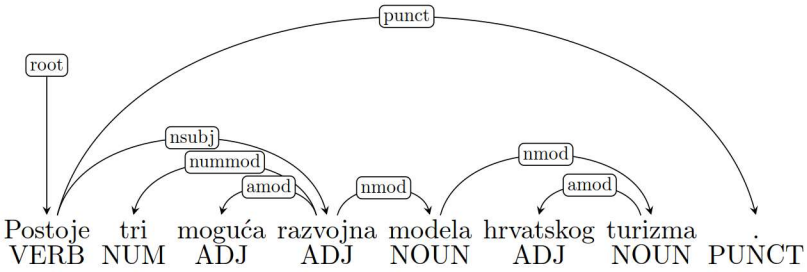
Slika 3.18: Ovdje je parser prepoznao riječ *Ivo* kao apoziciju riječi *priliku*. Ispravno je *iskoristio* $\xrightarrow{nsubj}$ *Ivo* jer je *Ivo* ovdje vršitelj radnje *je iskoristio*.



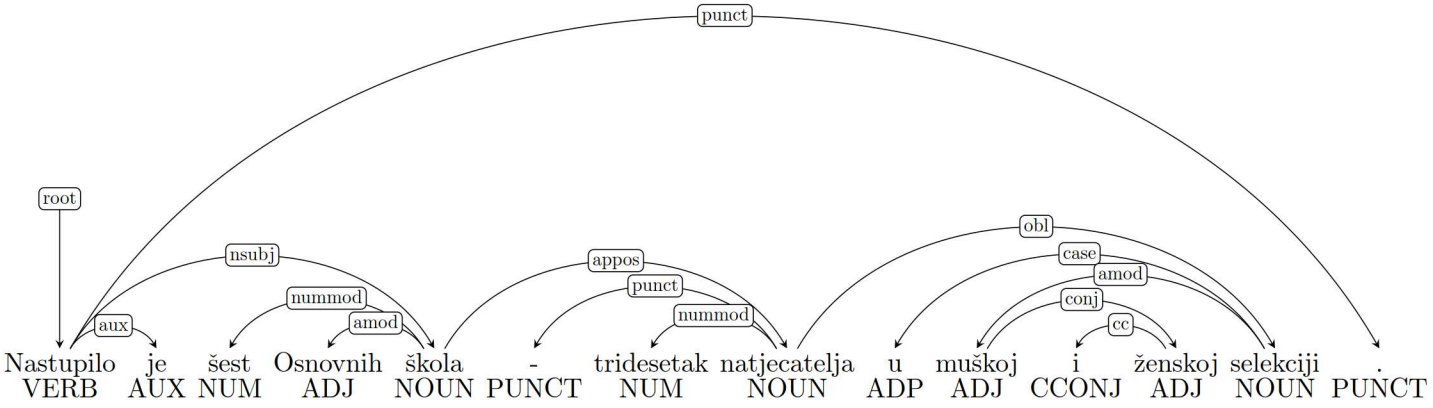
Slika 3.19: Ispravno je *odlagališta* $\xrightarrow{appos}$ *Viševac*, jer je imenica *Viševac* očito naziv *odlagališta*.



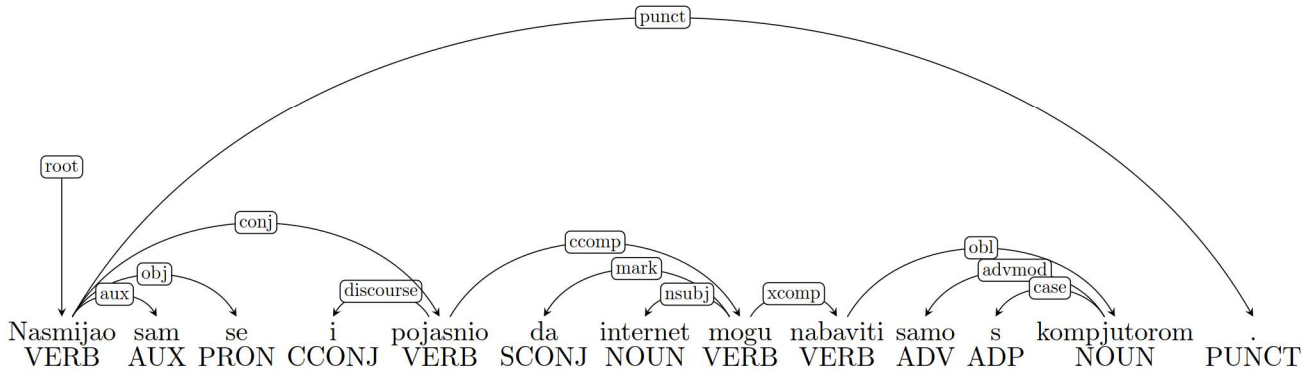
Slika 3.20



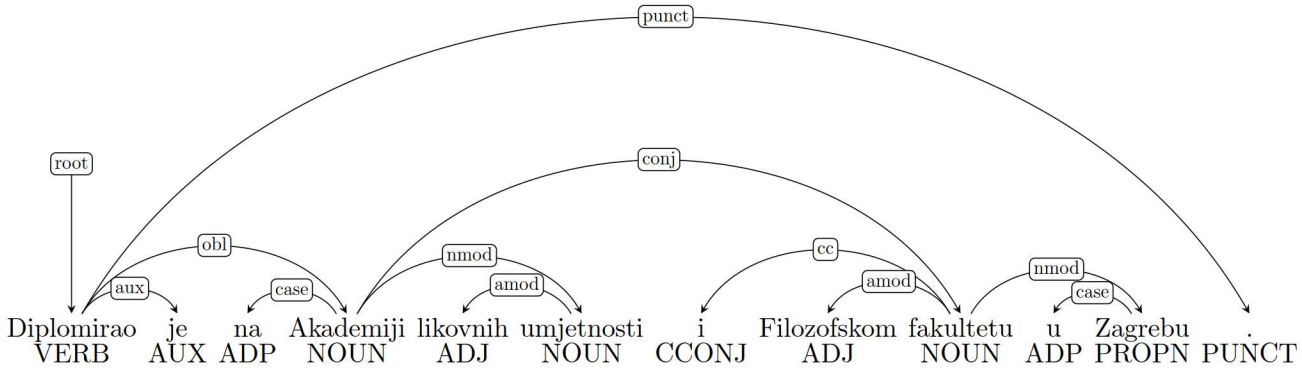
Slika 3.21: Ovdje je parser shvatio bit rečenice, ali zbog jedne krive oznake mu je učinak loš na ovom primjeru. Naime, ispravno je *modela* $\xrightarrow{amod}$ *razvojna*, a svaku od riječi u *postoje tri moguća* je s istim oznakama trebalo spojiti na riječ *modela*.



Slika 3.22: Točno je *nastupilo* $\xrightarrow{parataxis}$ *natjecatelja*. Parataxis je međuovisnost gdje dvije riječi stoje jedna pored druge bez eksplicitne koordinacije. U svakom slučaju, *natjecatelja* nije apozicija riječi *škola*. Također, na jednoj međuovisnosti je pogrešna oznaka: ispravno je *nastupilo* $\xrightarrow{obl}$ *škola*



Slika 3.23: Ovdje je spaCy parser dobro prepoznao sve nadređene i podređene riječi, ali je ponegdje dodijelio krive oznake međuovisnosti. Ispravno je bilo: *nasmijao* $\xrightarrow{expl:pv}$ *se*, *pojasnio* $\xrightarrow{cc}$ *i*, *mogu* $\xrightarrow{obj}$ *internet*.



Slika 3.24: Ispravno je *Akademiji* $\xrightarrow{nmod}$ *Zagrebu*. Naime, *u Zagrebu* opisuje i *Akademiju likovnih umjetnosti* i *Filozofski fakultet*.

Literatura

[1] Ž. Agić, N. Ljubešić: *The SETIMES.HR Linguistically Annotated Corpus of Croatian*, Proceedings of the Ninth International Conference on Language Resources and Evaluation (2014), 1724-1727.

[2] Ž. Agić, N. Ljubešić: *Universal Dependencies for Croatian (that work for Serbian, too)*, The 5th Workshop on Balto-Slavic Natural Language Processing (2015), 1-8.

[3] D. Chen, C. Manning: *A Fast and Accurate Dependency Parser using Neural Networks*, Empirical Methods in Natural Language Processing (2014), 740-750.

[4] A. Géron: *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, O'Reilly Media, Sjedinjene Američke Države, 2017.

[5] X. Glorot, Y. Bengio: *Understanding the difficulty of training deep feedforward neural networks*, Journal of Machine Learning Research - Proceedings Track 9(2010), 249-256.

[6] S. Kübler, R. McDonald, J. Nivre: *Dependency parsing (Synthesis Lectures on Human Language Technologies)*, Morgan & Claypool Publishers, 2009.

[7] C.Manning et al.: *CS224n: Natural Language Processing with Deep Learning*, Stanford / Winter 2019.

[8] T. Mikolov, K. Chen, G. Corrado, J. Dean: *Efficient estimation of word representations in vector space*, Proceedings of Workshop at ICLR 2013(2013).

[9] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean: *Distributed representations of words and phrases and their compositionality*, Advances in Neural Information Processing Systems 26(2013), 3111-3119.

[10] J. Nivre: *Incrementality in Deterministic Dependency Parsing*, Proceedings of the Workshop on Incremental Parsing (2004), 50-57.

[11] P. Qi, T. Dozat, Y. Zhang, C. Manning: *Universal Dependency Parsing from Scratch*, Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies (2018), 160-170.

[12] D. E. Rumelhart, G. E. Hinton, R. J. Williams: *Learning representations by back-propagating errors*, Nature 323(1986), 533-536.

[13] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov: *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, Journal of Machine Learning Research 15(2014), 1929-1958.

Sažetak

U novije vrijeme, prisutna je renesansa neuronskih mreža. Njihova osnovna građevna jedinica jest neuron. Neuron možemo shvatiti kao metodu kojom ćemo odvagnuti ulazne informacije za donošenje neke odluke. Neurone grupiramo u slojeve, a više takvih slojeva čini neuronsku mrežu. U prvom poglavlju smo opisali kako neuronske mreže uče te algoritme i tehnike koje se pritom koriste.

Glavni dio diplomskog rada se bavi primjenom strojnog učenja u obradi prirodnog jezika. Opisali smo kako se riječi mogu predstaviti pomoću vektora koji što bolje obuhvaćaju strukturu riječi, značenje i kontekst riječi te hijerarhije među riječima. Vektori riječi su u vektorskom prostoru smješteni tako da riječi koje se nalaze u sličnom kontekstu, imaju slične vektore. Objasnili smo algoritme i metode treniranja koje word2vec koristi.

Konačno, pokazali smo kako vektore riječi možemo koristiti za parsiranje međuovisnosti između riječi u rečenici. Opisali smo pohlepni parser temeljen na prijelazima koji se za ovaj zadatak trenira koristeći neuronsku mrežu. Naposljetku, prikazali smo rad trenutno najboljih poznatih parsera međuovisnosti za hrvatski jezik, treniranih na jednom hrvatskom označenom korpusu.

Ključne riječi: strojno učenje, umjetne neuronske mreže, obrada prirodnog jezika, vektori riječi, word2vec, pohlepni algoritmi, stabla parsiranja

Modeling word representations and dependency parsing with machine learning techniques

Abstract

Recently, there is a renaissance of neural networks. Their fundamental working unit is a neuron. We can think of a neuron as method for weighing input informations in order to make a decision. We can group neurons in layers that form neural networks. In the first section we described how neural networks learn, together with the algorithms and techniques used in the background.

The main part of the paper deals with the application of machine learning in natural language processing. We described how words can be represented with vectors that best capture the word structure, semantics and context as well as the hierarchies among words. Word vectors are placed in the vector space such that words that share common contexts, have similar vectors. We explained the algorithms and training methods that word2vec uses.

Finally, we showed how to use word vectors for parsing the dependencies between words in a sentence. We described a greedy transition-based parser that is trained for this task using a neural network. Lastly, we presented the results of cutting-edge parsers for Croatian that were trained on a Croatian annotated corpus.

Keywords: machine learning, artificial neural networks, natural language processing, word embeddings, word2vec, greedy algorithms, parse trees

4 Životopis

Patrick Nikić je rođen 1996. godine u mjestu Isola della Scala (Italija). Pohađao je tri godine osnovnu školu Marco Pizzicaroli u mjestu Bionde (Italija). Nakon toga, 2005. godine se seli u Vinkovce gdje pohađa osnovnu školu Vladimira Nazora i potom gimnaziju Matija Antun Reljković. U srednjoj školi sudjeluje na školskim i županijskim natjecanjima iz matematike, fizike, kemije, geografije i latinskog jezika. Završio je preddiplomski studiji matematike na Odjelu za matematiku u Osijeku s prosjekom ocjena 5.0. Nakon završetka preddiplomskog studija, upisao je diplomski studiji matematike i računarstva na Odjelu za matematiku u Osijeku.

Na preddiplomskom studiju je radio kao demonstrator iz kolegija Uvod u strukture podataka i algoritme kod doc.dr.sc. Slobodana Jelića. U ljeto 2017. godine je obavljao stručnu praksu na Lehrstuhl für Angewandte Mathematik kod prof. dr. sc. Lars Grüne u mjestu Bayreuth (Njemačka). Na međunarodnoj konferenciji za operacijska istraživanja (KOI 2018) je prezentirao rad *A Fast Algorithm for Solving the Multiple Ellipse Detection Problem* koji je izradio sa skupinom autora.

Sudjelovao je na mnogim natjecanjima od kojih se ističu Srednjoeuropsko studentsko ACM natjecanje (CERC) 2016. i 2018. godine te IEEEXtreme natjecanje iz programiranja 2016., 2017. i 2018. godine. Tijekom studiranja je primao državnu stipendiju Ministarstva znanosti i obrazovanja Republike Hrvatske te stipendiju tvrtke Atos Convergence Creators. Dobitnik je nagrade za uspješnost u studiranju, Pročelnikove nagrade i Rektorove nagrade.