TakeLab

Laboratorij za analizu teksta i inženjerstvo znanja
Text Analysis and Knowledge Engineering Lab
Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva
Unska 3, 10000 Zagreb, Hrvatska

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1326

Jezično modeliranje hrvatskoga jezika modelima dubokoga učenja

Sven Vidak

Zagreb, srpanj 2016.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA DIPLOMSKI RAD PROFILA

Zagreb, 11. ožujka 2016.

Predmet: **Strojno učenje**

DIPLOMSKI ZADATAK br. 1326

Pristupnik: **Sven Vidak (0036464598)**
Studij: Računarstvo
Profil: Računarska znanost

Zadatak: **Jezično modeliranje hrvatskoga jezika modelima dubokoga učenja**

Opis zadatka:

Jezični modeli služe za procjenu vjerojatnosti riječi u danom kontekstu i jedan su od osnovnih alata u obradi prirodnog jezika. Tradicionalni se jezični modeli oslanjaju na statistiku o pojavljivanju n-grama u korpusu, stoga iziskuju velike količine podataka te loše modeliraju odnose između udaljenih riječi. U posljednje vrijeme metode dubokog učenja temeljene na neuronskim mrežama su preuzele vodstvo nad tradicionalnim pristupima.
U okviru diplomskoga rada potrebno je proučiti jezične modele s naglaskom na jezične modele temeljene dubokim neuronskim mrežama. Upoznati se s trenutno dostupnim bibliotekama za duboko učenje te odabrati onu najprikladniju. Razviti programsku implementaciju jezičnih modela za hrvatski i engleski jezik, za nekoliko različitih domena. Provesti eksperimentalno vrednovanje jezičnog modela u smislu mjere zbunjenosti (engl. perplexity) te načiniti usporedbu s vodećim rezultatima u području kao i tradicionalnim modelima. Razraditi primjenu jezičnih modela na dva zadatka: ispravljanje pogrešaka u optičkom raspoznavanju znakova te generiranju korisničkih komentara na novinske članke. Razmotriti i druge moguće primjene modela. Radu priložiti izvorni i izvršni kod razvijenog sustava, skupove podataka i potrebnu dokumentaciju te citirati korištenu literaturu.

Zadatak uručen pristupniku: 18. ožujka 2016.
Rok za predaju rada: 1. srpnja 2016.

Mentor:

Doc. dr. sc. Jan Šnajder

Djelovođa:

Doc. dr. sc. Tomislav Hrkać

Predsjednik odbora za diplomski rad profila:

Prof. dr. sc. Siniša Srbljić

Zahvaljujem mentoru doc. dr. sc. Janu Šnajderu na tome što me prihvatio u svoj već prepunjeni tim te mi pomogao oko svih problema koje sam imao prilikom izrade ovog, ali i ostalih radova.

Također, hvala asistentima Mladenu Karanu i Martinu Tuteku koji su trpili i odgovarali na gomilu mailova kojima sam ih ponekad zatrpavao te hvala svima koji se brinu za Ljudmilu i Đurđu koje su zaslužne za nastanak ovog rada.

Na kraju, hvala roditeljima koji su mi omogućili sve ovo i uvijek bili podrška te hvala svim kolegama koji su uvijek bili spremni pomoći oko raznih problema.

Sadržaj

1. Uvod

Pojavom računala i spoznajom da računala mogu zaključivati i "razmišljati", ljudi su maštali o izradi robota koji će se moći ponašati kao čovjek: hodati, pričati, razmišljati i djelovati na odgovarajući način ili slušati naredbe. Također, mnoštvo filmova iz 70-tih i 80-tih godina prošlog stoljeća bavi se tom tematikom, no problem nije toliko jednostavan koliko bi netko mogao pomisliti. Naime, kako bi računalo moglo komunicirati s čovjekom, mora naučiti razumijevati, ali i "pričati" jezik kojim se služi čovjek. Jezik je vrlo kompleksan sustav pravila i normi koje je (čak i ljudima) ponekad izrazito teško u potpunosti naučiti. Za svakodnevnu komunikaciju nije nužno potrebno znati jezik do u detalje te se često svodi na učenje promatranjem i slušanjem tokom odrastanja dok se preostali dio nauči tokom obrazovanja. Međutim, kao i s većinom problema koji su ljudima relativno jednostavni (poznavanje jezika dovoljno dobro da mogu komunicirati), računala imaju puno problema. Ako promatramo razgovor dvoje ljudi, osobe se vrlo lako mogu prisjetiti nekog dijela razgovora od prije nekoliko minuta, sati pa čak i godina te na temelju toga odlučiti što će reći. Računala s druge strane imaju problem s pamćenjem što se dogodilo daleko u prošlosti (u kontekstu razgovora dvoje ljudi, nekoliko minuta i nekoliko desetaka rečenica može se smatrati dalekom prošlosti kada govorimo o računalima) jer je memorijski vrlo zahtjevno. Zbog toga su se razvile posebne tehnike pamćenja prošlosti u nizovima podataka koji se daju računalu.

Strojno učenje dio je područja umjetne inteligencije koje proučava algoritme koji mogu samostalno učiti iz podatka. Cilj ovog rada je proučiti problem jezičnog modeliranja hrvatskog i engleskog jezika te razmotriti primjenu na optičko raspoznavanje teksta i generiranje teksta. Ti problemi pripadaju području obrade prirodnog jezika (engl. *natural language processing*) unutar kojeg se proučavaju problemi kao što su: strojno prevođenje (engl. *machine translation*), generiranje prirodnog jezika (engl. *natural language generation*), sažimanje teksta (engl. *automatic summarization*) i sl. U samim počecima računarstva ti su se problemi po-

kušavali riješiti modelima jezika koji su u obzir uzimali broja pojavljivanja jedne ili više riječi (*n-gram* modeli), ali takvi modeli su problematični jer im treba puno podataka da bi modelirali neke jednostavne ovisnosti u jeziku. Pojavom dubokog učenja, modeliranje ovisnosti unutar jezika postalo je puno lakše te će unutar ovog rada biti razmotreni modeli dubokog učenja koji danas postižu bolje rezultate od modela temeljenih na *n-gramima*.

Duboko učenje posljednjih je nekoliko godina uzelo veliki zamah te je danas neizostavan alat za rješavanje bilo kakvog težeg problema u područjima računalnog vida i obrade prirodnog jezika. Sposobnost dubokih neuronskih mreža da samostalno iz podataka izvuku informacije za koje same zaključe da su im korisne te razvoj efikasnih metoda njihovog učenja i dizajniranja doveli su do značajnog napretka u efikasnosti i kvaliteti rješavanja složenih problema kao što su raspoznavanje objekata na slikama, opis objekata na slikama i njihovi relativni odnosi, raspoznavanje govora, prevođenje s jednog jezika na drugi i sl. Detaljnije o običnim (unaprijednim) neuronski mmrežama, ali i njihovim dubokim "verzijama" se može pročitati u poglavlju 2.

Jezično modeliranje jedan je od najznačajnijih problema u području obrade prirodnog jezika. Cilj jezičnog modeliranja je na neki niz jezičnih jedinki (slova, riječi, rečenica itd.) postaviti vjerojatnosnu razdiobu koja će na temelju trenutne riječi i riječi koje joj prethode moći procijeniti koja je najvjerojatnija sljedeća riječ. Počeci jezičnog modeliranja sežu u osamdesete godine 20. stoljeća [10] i primjene u području raspoznavanja govora. Uz to, jezično modeliranje se koristi i za strojno prevođenje, označavanje vrste riječi, optičko raspoznavanje teksta (engl. *optical character recognition*) i sl. Tehnike koje se koriste u nekim od tih područja bit će objašnjene u poglavlju 3.

Poglavlje **??** daje kratak pregled srodnih radova vezanih za temu jezičnog modeliranja. U poglavlju 2 dan je pregled neuronskih mreža od samih početaka pa sve do danas. Poglavlje 3 opisuje jezično modeliranje te razne tehnike kojima se pokušalo riješiti probleme nekih jednostavnijih modela, a potom su opisane metode koje danas daju najbolje rezultate. Poglavlje 4 daje pregled materijala, metoda i modela koji su korišteni prilikom izrade ovog rada, a poglavlje 5 pregled dobivenih rezultata. Konačno, u poglavlju 6 je dan kratak sažetak cijelog rada i dobivenih rezultata te osvrt na to što bi se moglo dalje raditi i koje su moguće preinake ili poboljšanja.

2. Umjetne neuronske mreže

2.1. Perceptron

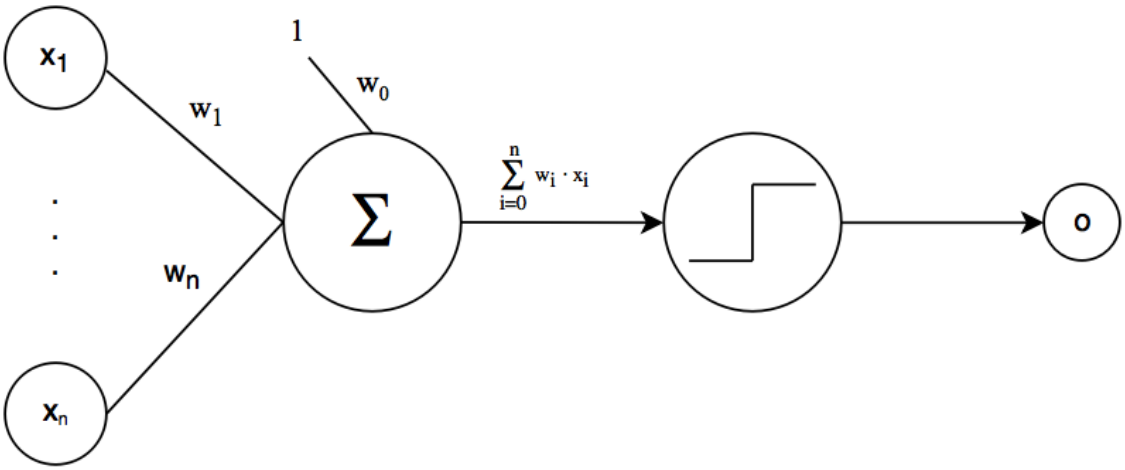
Umjetne neuronske mreže jedan su od brojnih računalnih modela koji su inspiraciju pronašli u prirodi. Ideja se razvila na temelju živčanog sustava koji se sastoji od velikog broja neurona (u terminologiji neuronskih mreža ponekad se koristi naziv procesna jedinica) koji međusobno komuniciraju signalima koje prenose jedan do drugog pa sve do npr. mišićnih tkiva. Bitno je napomenuti kako se danas pojmovi višeslojnog perceptrona i umjetne neuronske mreže (u većini slučajeva) koriste za istu stvar iako postoje razlike koje će biti objašnjenje u nastavku poglavlja.

Prva ideja o umjetnom neuronu seže u 1943. godinu kada su Warren McCulloh i Walter Pitts [6] [30] predložili model neurona koji može rješavati logičke operacije *I*, *ILI*, *NE* zbrajanjem binarnih ulaza koji bi, ako bi zbroj bio veći od nekog praga, na izlazu dali 1, a inače 0. U to je vrijeme ta ideja bila vrlo značajna jer su ljudi mislili kako će problem umjetne inteligencije biti riješen postojanjem računala koje će biti sposobno rješavati logičke probleme koristeći navedene operacije. Međutim, model je imao problem koji se pokazao ključnim za njegovu praktičnu primjenu – nije mogao učiti.

Godine 1949. Donald Hebb je objavio knjigu *The organization of behavior: A neuropsychological theory* [22] iz koje je, za područje umjetnih neurona, najznačajniji sljedeći citat: "Kada je neuron A dovoljno blizu neurona B da bi ga uzbudio te često ili periodički sudjeluje u njegovoj aktivaciji, u oba se neurona događa ili rast ili metabolička promjena tako da se učinkovitost neurona A povećava". Kraće rečeno, ako dva neurona često "komuniciraju", veza između njih postaje jača.

Kombinacijom dviju navedenih ideja, 1958. godine Frank Rosenblatt predložio je model umjetnog neurona koji je nazvao perceptron.

Kao što se vidi na slici 2.1, perceptron na ulaz dobiva niz brojeva, njih množi



Slika 2.1: Perceptron

odgovarajućim težinama te se rezultati tih umnožaka zbrajaju te aktiviraju neuron (dajući 1 na izlazu) ako je zbroj veći od nekog praga. Ono što je perceptron ima, a modeli prije nisu imali, jest pomak (engl. *bias*) označen s w_0, koji omogućava pomicanje naučene funkcije. Takav model sustava koji je sposoban zaključivati može obavljati jednostavnu kategorizaciju, ali je ograničen linearnom funkcijom razdvajanja (engl. *decision function*) kategorija u prostoru.

Izlaz neurona računa se prema formuli 2.1 u kojoj se koristi općenita aktivacijska funkcija f. Perceptron koristi funkciju skoka (engl. *step function*) definiranu formulom 2.2.

$$\begin{aligned} net &= \sum_j x_j \cdot w_j + w_0 \\ out &= f(net) \end{aligned} \tag{2.1}$$

$$f(x) = \begin{cases} 0, & \text{ako } x < 0 \\ 1, & \text{inače} \end{cases} \tag{2.2}$$

2.1.1. Pravilo perceptrona

Proces učenja perceptrona svodi se na povećavanje ili smanjivanje iznosa težina. Smjer promjene težina ovisi o izlazu perceptrona i točnom izlazu koji je dan u podacima kojima učimo perceptron. Ako je izlaz preceptrona 0, a točan izlaz 1, tada ćemo povećati odgovarajuće težine, a ako je izlaz preceptrona 1, a točan izlaz 0, tada ćemo ih smanjiti.

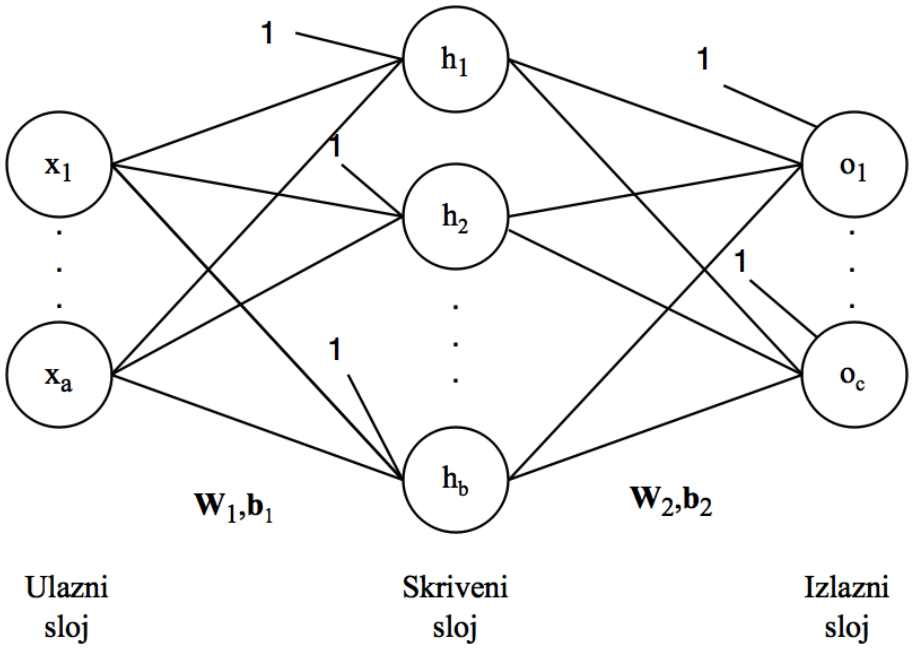
Ako se sa $\boldsymbol{x}$ označi primjer iz skupa podataka za učenje, a sa l oznaka klase tog primjera, a sa y izlaz perceptrona, tada težine perceptrona ažuriramo na sljedeći

način:

$$w_j = w_j + \eta \cdot (y - l) \cdot x_j.$$

2.2. Višeslojni perceptron

Slaganjem pojedinačnih perceptrona u slojeve te spajanjem tih slojeva težinskim vezama dolazimo do modela višeslojnog perceptrona (engl. *multi-layered perceptron, MLP*). Treba naglasiti kako se danas češće koristi naziv umjetna (unaprijedna) neuronska mreža (engl. *feed forward artificial neural network*) jer je došlo do razvoja više tipova neurona, pa se perceptron koristi vrlo rijetko.



Slika 2.2: Višeslojni perceptron

Na slici 2.2 prikazana je unaprijedna neuronska mreža koji se sastoji od tri sloja. Općenito, svaka se unaprijedna neuronska mreža sastoji od tri vrste sloja: ulazni, skriveni i izlazni. Ulazni sloj dobiva sirove podatke iz skupa podataka za učenje te ih množi odgovarajućim težinama i šalje ih na ulaz prvog skrivenog sloja. Skriveni slojevi (kojih može biti više) služe kako bi se povećala izražajnost mreže i time omogućilo učenje kompliciranijih funkcija razdvajanja. Također, skriveni slojevi mogu težinama modelirati neke zakonitosti podataka koje nisu očite i koje bi čovjek teško (ako uopće) mogao uočiti. Izlazi iz mreže su (općenito) realni

brojevi koji označavaju stupanj pripadnosti ulaznog primjera klasi kojoj odgovara promatrani neuron. Stupanj pripadnosti može se odrediti na različite načina, a najčešće se kao aktivacijska funkcija koristi funkcija identiteta (2.3), sigmoidalna (2.4) ili funkcija *softmax* (2.5) koja daje vjerojatnosti za svaku klasu na izlazu (svaki izlazni neuron predstavlja jednu klasu).

$$f(x) = x \tag{2.3}$$

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.4}$$

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{c} e^{x_j}}, i = 1, \ldots, c \tag{2.5}$$

Izračun izlaza umjentne neuronske mreže (perceptron s poopćenim aktivacijskim funkcijama f_1 i f_2) na primjeru slike 2.2 dan je formulom 2.6.

$$\begin{aligned} \boldsymbol{h} &= f_1(\boldsymbol{W}_1 \cdot \boldsymbol{x} + \boldsymbol{b}_1) \\ \boldsymbol{o} &= f_2(\boldsymbol{W}_2 \cdot \boldsymbol{h} + \boldsymbol{b}_2) \end{aligned} \tag{2.6}$$

Aktivacijske funkcije mogu, ali i ne moraju biti jednake. Također, ovisno o literaturi, mogu se naći primjeri gdje se pomak ne koristi u izlaznom sloju mreže.

U ovom trenutku treba primijetiti nekoliko bitnih stvari.

Višeslojni perceptron i dalje je ograničen model jer može naučiti samo linearne granice. Kako bi neuronska mreža (mreža neurona s proizvoljno odabranim tipom neurona) mogla naučiti nelinearne granice, aktivacijska funkcija mora biti nelinearna. Pojavom perceptrona i porastom interesa znanstvenika za neuronskim mrežama, razvilo se nekoliko tipova neurona s nelinearnim aktivacijskim funkcijama koji su se počeli koristiti umjesto perceptrona.

ADALINE (*Adaptive Linear Element*) [1] neuron kao aktivacijsku funkciju koristi funkciju identiteta. Popularniji tip neurona je onaj sigmoidalni koji se dugo godina koristio u neuronskim mrežama. Međutim, pojavom paradigme dubokog učenja, znanstvenici su uočili neke nedostatke [41] što je dovelo do pojave ispravljačkog neurona (engl. *rectified linear unit, ReLU*).

Drugo zapažanje tiče se učenja neuronske mreže o čemu će više riječi biti u nastavku.

2.3. Učenje umjetne unaprijedne neuronske mreže

Velika popularnost neuronskih mreža je među znanstvenicima, uz puno dobrih ideja, rezultirala i otkrivanjem nedostataka koji su onemogućavali njihovo korištenje za tada postojeće probleme.

Marvin Minsky i Seymour Papert su 1969. godine u [38], među ostalim, naglasili kako perceptron nije sposoban naučiti običnu EX-ILI (engl. *XOR*) funkciju jer primjeri nisu linearno odvojivi [6]. Za razliku od Rosenblatta (perceptron) te Widrowa i Hoffa (ADALINE) koji su bili uvjereni kako će konektivizam riješiti sve probleme umjetne inteligencije, ostatak znanstvenika nije bio siguran u to te je Minksyjev rad označio početak razdobolja poznatog pod nazivom "zima umjetne inteligencije" (engl. *AI winter*).

Problem je zapravo bio vrlo jednostavan – perceptron nije mogao naučiti EX-ILI funkciju, ali je to bilo moguće s više (skrivenih) slojeva perceptrona. Međutim, pravilo perceptrona jedino definira na koji način ažurirati težine do zadnjeg sloja, ali ne i slojeva prije te zbog toga Rosenblattov algoritam nije mogao naučiti višeslojni perceptron. Isto tako, perceptron koristi funkciju koja je linearna i nije diferencijabilna te je nemoguće pomoću derivacija finije odrediti iznos ažuriranja, ali i naučiti nelinearne funkcije odvajanja. Mreže ADALINE koristile su derivacije za ažuriranje težina, ali su te mreže i dalje učile običnu linearnu funkciju.

Rješenje je došlo u obliku algortima propagacije greške unatrag (engl. *back-propagation algorithm*). Ideja je bila da neuroni kao aktivacijsku funkciju koriste nelinearnu i diferencijabilnu funkciju, te da se greška koju mreža radi na izlaznom sloju prenosi sve do početka mreže i na taj način određuje iznos ažuriranje težina u svim slojevima, a ne samo u posljednjem.

Sam algoritam je predložen pojavio 60-tih i 70-tih godina 20. stoljeća te je čak i implementiran na računalima, ali sve do 1974. i doktorata Paula Werbosa [45] nitko nije pokušao algoritam primijeniti na neuronske mreže. Međutim, Werbos svoj rad nije objavio do 1982. jer nitko od znanstvenika nije vjerovao da će to riješiti problem niti je htio čuti njegovu ideju. Tek je 1986., kada su ga Rumelhart, Hinton i Williams u [42] jasno i precizno definirali, algoritam postao popularan te se u jednakom obliku zadržao do danas [6].

2.3.1. Algoritam propagacije greške unatrag

Prije opisa rada algoritma i formula koje algoritam koristi, treba objasniti korištenu notaciju. Naime, notacija koja se koristi u literaturi općenito nije standar-

dizirana tako da notacija koja se koristi u ovom radu (vjerojatno) neće odgovarati notaciji drugih radova.

Za treniranje neuronske mreže potreban nam je skup podataka za učenje $\boldsymbol{X} = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_N, \boldsymbol{y}_N)\}$. $\boldsymbol{x}_i$ je m-dimenzijski vektor koji predstavlja i-ti ulazni primjer za učenje, dok je $\boldsymbol{y}_i$ o-dimenzijski vektor koji predstavlja točan izlaz mreže za odgovarajući ulazni primjer.

Svaki neuron će kao aktivacijsku funkciju koristiti sigmoidalnu funkciju, ali može se koristiti bilo koja derivabilna, nelinearna funkcija.

Kako bi algoritam propagacije greške unatrag mogao učiti, treba nam funkcija koju ćemo optimizirati. Za tu funkciju odabrat ćemo funkciju srednje kvadratne pogreške (engl. *mean squared error*), točnije, funkciju koja je jednaka polovici srednje kvadratne pogreške (zbog kraćenja prilikom deriviranja) (formula 2.7).

$$E(\boldsymbol{\theta}) = \frac{1}{2N} \sum_{i=1}^{N} (\boldsymbol{y}_i - \boldsymbol{o}_i(\boldsymbol{\theta}))^2 \tag{2.7}$$

Vektor $\boldsymbol{o}_i$ predstavlja izlaz koji je neuronska mreža izračunala za dani ulazni primjer i parametre algoritma (težine mreže) $\boldsymbol{\theta}$ ($\boldsymbol{\theta}$ ovdje zapravo označava kombinaciju parametara $\boldsymbol{w}$ i $\boldsymbol{b}$).

Ideja algoritma propagacije greške unatrag je korištenjem pravila lanca (engl. *chain rule*) [2] za izračun gradijenta funkcije cilja te nekom od varijanti metoda gradijentnog spusta (engl. *gradient descent*) (opisanih u poglavlju 2.6) ažurirati težine neuronske mreže te na taj način smanjivati grešku E koju mreža radi.

Pravilo lanca je zapravo primjena parcijalnih derivacija na početni izraz tako dugo dok se ne dođe do izraza koji ovisi o varijablama koje možemo direktno mijenjati. Npr. neka je zadan izraz $f(a, b) = (a + b) * (b + 1)$ [3]. Kako bi se pokazalo kako radi pravilo lanca, uvode se dvije nove varijable c i d pri čemu je $c = (a + b)$, a $d = (b + 1)$ iz čega slijedi da f možemo zapisati kao $f(a, b) = c \cdot d$. Sada je ideja naći parcijalne derivacije funkcije f po varijablama o kojima ona ovisi: a i b. U formuli 2.8 vidimo da se pravilo lanca primjenjuje na f tako što množi dvije parcijalne derivacije. Međuizraz d ovdje nije bitan jer a ne ovisi o njemu.

$$\frac{\partial f}{\partial a} = \frac{\partial f}{\partial c} \frac{\partial c}{\partial a} = d \cdot 1 \tag{2.8}$$

Parcijalna derivacija po varijabli b je nešto kompliciranija jer oba međuizraza ovise o njoj. Zbog toga oni neovisno jedan o drugome deriviraju te na kraju sumiraju u konačno rješenje (2.9)

$$\frac{\partial f}{\partial b} = \frac{\partial f}{\partial c}\frac{\partial c}{\partial b} + \frac{\partial f}{\partial d}\frac{\partial d}{\partial b} = d \cdot 1 + c \cdot 1 \tag{2.9}$$

Algoritam propagacije greške unatrag koristi isto pravilo za izračun parcijalnih derivacija $\frac{\partial E}{\partial \boldsymbol{w}}$ i $\frac{\partial E}{\partial \boldsymbol{b}}$. Konačne formule i dokaz formula može se pronaći na [4].

U nastavku će biti dan pregled danas često korištenih dubokih neuronskih mreža te neke njihove primjene na probleme obrade prirodnog jezika.

2.4. Konvolucijske neuronske mreže

Konvolucijske neuronske mreže (engl. *convolutional neural network, CNN*) poseban su oblik unaprijednih umjetnih neuronskih mreža koje umjesto matričnog množenja težina između slojeva koriste konvoluciju. Mogu se primjeniti na podatke koji imaju rešetkastu strukturu te je upravo zbog toga ovaj oblik neuronskih mreža često korišten u području računalnog vida. U nastavku će prvo biti objašnjena sama operacija konvolucije te njena prilagodba na probleme strojnog učenja i umjetne inteligencije, potom će se opisati svojstva i struktura konvolucijskih neuronskih mreža i objasniti način na koji one rade u slučaju kada imamo dvodimenzionalan ulaz, a na kraju će biti opisana njihova prilagodba na jednodimenzonalne tekstne podatke.

2.4.1. Konvolucija

Konvolucija je matematička operacija definirana kao:

$$s(t) = (x * w)(t) = \int_t x(a)w(t-a)da$$

gdje su x i w neke dvije funkcije te neka t označava vrijeme. Najjednostavnije objašnjenje konvolucije je ono koje kaže kako je konvolucija operacija koja nam daje količinu preklapanja dviju funkcija. Malo preciznija definicija je ona koja kaže da je konvolucija težinsko usrednjenje funkcije $x(a)$ u trenutku t gdje su težine određene težinskom funkcijom $w(-a)$. Iz te definicije se vidi kako će težinska funkcija u različitim vremenskim periodima t isticati različite dijelove funkcije x. Kako su podaci u računalu u diskretnom obliku, definiramo diskretnu konvoluciju:

$$s[t] = (x * w)[t] = \sum_{a=-\infty}^{\infty} x[a]w[t-a]$$

.

Dosad opisani oblik konvolucije danas se često koristi u područjima obrade signala, akustike ili elektičnih krugova. Međutim, ako konvoluciju želimo primjeniti na probleme računarske znanosti, moramo definirati terminologiju koja se koristi te oblik konvolucije koji je prilagođen za dvodimenzionalan ulaz.

U terminologiji konvolucijskih neuronskih mreža, x se naziva ulazom (engl. *input*), w se naziva jezgra (engl. *kernel*), a izlaz (engl. *output*) konvolucije je mapa značajki (engl. *feature map*). Također, pošto se konvolucijske neuronske mreže često koriste s podacima koji su dvodimenzionalni, definira se i dvodimenzijska konvolucija i to kao:

$$s[i,j] = (X * K)[i,j] = \sum_m \sum_n X[i-m, j-n]K[m,n]$$

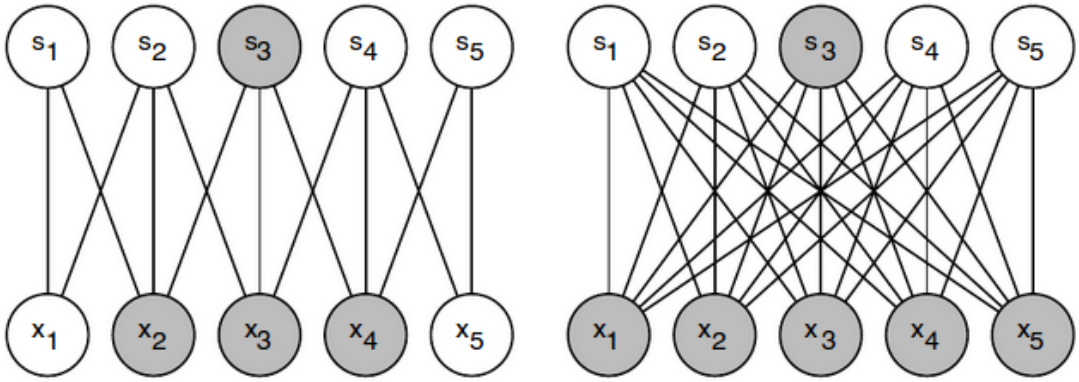
gdje je X neki dvodimenzijski ulaz, a K jezgra – polje parametara koji se uče [11].

2.4.2. Svojstva konvolucijskih neuronskih mreža

Konvolucijske neuronske mreže karakterizira nekoliko zanimljivih svojstava koja su opisana u nastavku.

Za razliku od običnih unaprijednih neuronskih mreža kod kojih je neuron izlaznog sloja povezan sa svakim neuronom ulaznog sloja, kod konvolucijskih neuronskih mreža neuron izlaznog sloja povezan je s manjim brojem neurona na ulazu te se takav način povezanosti zove raspršena povezanost (engl. *sparse connectivity, sparse interactions*). Slika 2.3 prikazuje navedenu razliku. Na lijevoj strani vidimo kako kod konvolucijske neuronske mreže izlazni neuron s_3 ovisi samo o ulaznim neuronima x_2, x_3 i x_4 dok je na desnoj strani primjer klasične neuronske mreže. Ulazni neuroni koji utječu na izlazni neuron zovu se vizualno ili receptivno polje (engl. *receptive field*) izlaznog neurona. Raspršenom povezanosti smanjuje se složenost modela, ali i omogućava algoritmu učenje lokalnih značajki kao što su orijentirani vrhovi, krajnje točke ili kutevi koji se na dubljim slojevima kombiniraju i tvore značajke višeg reda [28].

Mogućnost iskorištavanja osnovnih detektora značajki na bilo kojem dijelu slike je svojstvo koje se u kovolucijskim neuronskim mrežama ostvaruje pomoću dijeljenja težina (engl. *shared weights*). Ako jedan sloj promatramo kao dvodimenzionalnu ravninu tada je svakoj ravnini u tom sloju pridružen jedan skup parametara. Na taj način imamo različite skupove parametara koje učimo – različite jezgre, koje nam u svakom koraku daju neki izlaz. Taj izlaz, koji je



Slika 2.3: Raspršena povezanost (iz [11])

kombinacija prethodnog sloja i jezgara koje koristimo zovemo mapom značajki.

Zbog dijeljenja težina slojevi konvolucijske neuronske mreže imaju svojstvo invarijantnosti na translaciju. To znači da se izlaz mijenja na isti način kao i ulaz. To je korisno upravo kod osnovnih detektora koji detektiraju npr. rubove, ali može stvarati probleme kada se radi detekcija lica jer gornji i donji dio lica nemaju jednake značajke [11].

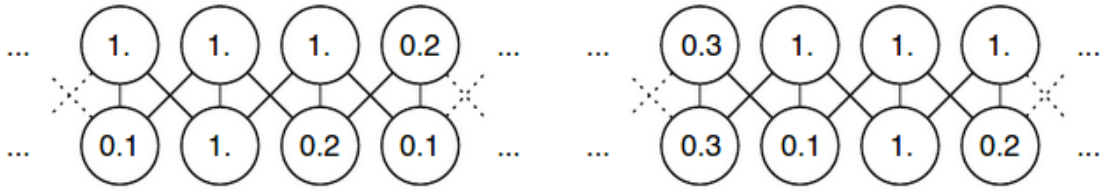
Jedno od zanimljivijih svojstava konvolucije je to da na nju možemo gledati kao nametanje beskonačno jake apriorne distribucije na parametre nekog sloja. To znači da bi sloj trebao naučiti funkciju koja je lokalizirana te invarijantna na translaciju. Iz ovoga slijedi da će konvolucija kao zamjena za obično matrično množenje u neuronskim mrežama dati dobre rezultate samo kada funkcija koju sloj mora naučiti ima ta svojstva. Ako ti nije slučaj, korištenje konvolucije će dovesti do velike pogreške treniranja.

2.4.3. Izvlačenje

Izvlačenje (engl. *pooling*) smanjuje veličinu mapa značajki [28] na način da jedan izlaz iz mreže dobiva kao kombinaciju nekoliko podataka iz mape značajki (najčešće su to najbliži susjedi). Postoji nekoliko načina na koji se podaci mogu kombinirati, a najpopularniji su: izvlačenje maksimuma (engl. *max pooling*), izvlačenje usrednjavanjem (engl. *average pooling*), izvlačenje težinskim usrednjavanjem (engl. *weighted average pooling*) ili izvlačenje pomoću L2-norme (engl. *L2-norm pooling*) [11].

Na primjer, ako imamo ulaznu matricu $A = \begin{bmatrix} 3 & 4 & 5 & 6 \\ 2 & 1 & 0 & 9 \\ 1 & 5 & 6 & 7 \\ 6 & 1 & 2 & 0 \end{bmatrix}$, tada bi nakon izvlačenja maksimuma u kojem koristimo prozor veličine 2×2 bez preklapanja (s pomakom 2) dobili matricu $B = \begin{bmatrix} 4 & 9 \\ 6 & 7 \end{bmatrix}$.

Ono što također dobivamo izvlačenjem je smanjenje preciznosti kojom su zapisane pozicije određenih značajki u ulaznim podacima tj. invarijantnost na male promjene u ulaznim podacima. Na slici 2.4 je vidljivo to svojstvo. Donji sloj prikazuje izlaz u mapi značajki dok je gornji sloj nastao izvlačenjem. Lijevo vidimo brojeve u početnom stanju, a desno je situacija koja nastane kada izlaz u mapi značajki pomaknemo za jedno mjesto udesno. Ono što se dogodi je promjena vrijednosti samo na 2 gornja neurona dok su u donjem sloju promijenjene sve vrijednosti.



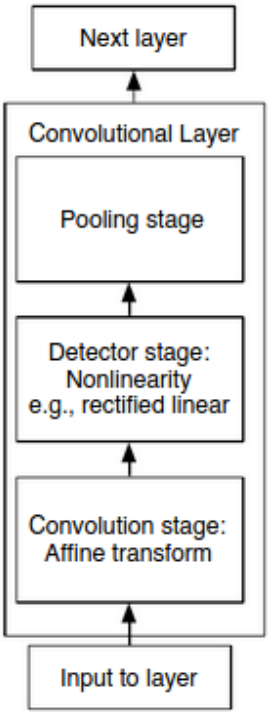
Slika 2.4: Invarijantnost uslijed izvlačenja (*max pooling*) (iz [11])

Invarijantnost na male promjene u ulaznim podacima korisna je u slučajevima kada je korisnije znati pojavljuje li se neka značajka u ulazu te u kojem je odnosu s drugim značajkama koje promatramo, nego na kojoj se točno poziciji pojavljuje.

Uz gore navedena svojstva, izvlačenje ima još jednu upotrebu, a to je rješavanje problema koji se pojavljuje kada su ulazni podaci različite veličine. Naime, u velikom broju praktičnih problema (posebno kada na ulazu imamo rečenice ili, općenitije, neki tekst) veličina ulaza varira između primjera koji su nam dostupni. Korištenjem različitih susjedstva koja se kombiniraju u jedan izlaz ili korištenjem različitih veličina pomaka prozora u izlaznom sloju moguće je svaki ulaz na kraju prikazati kao vektor fiksne veličine koji tada možemo iskoristiti za neki od postojećih algoritama ili kao ulaz u potpuno povezanu unaprijednu neuronsku mrežu.

2.4.4. Struktura konvolucijskih neuronskih mreža

Na slici 2.5 prikazan je najbitniji element svake konvolucijske neuronske mreže – konvolucijski sloj (engl. *convolutional layer*). Niz konvolucijskih slojeva koji čine kaskadu zajedno s potpuno povezanim slojem kakvi se koriste u običnim neuronskim mrežama čini konvolucijsku neuronsku mrežu.

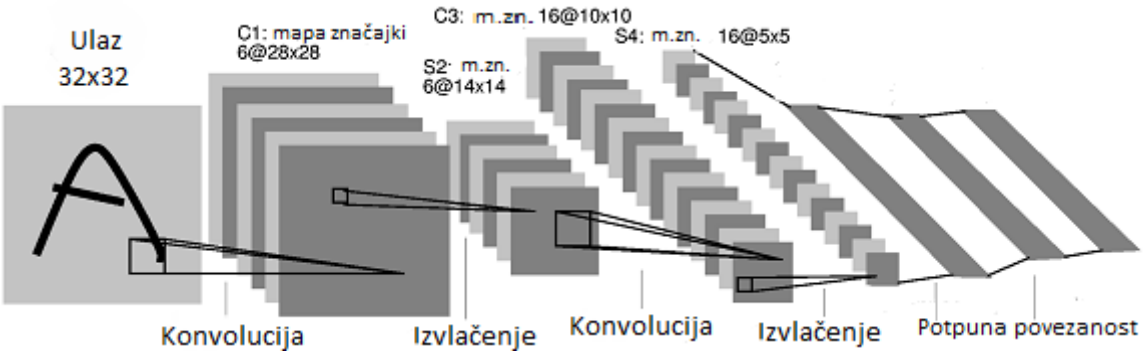


Slika 2.5: Konvolucijski sloj (iz [11])

Svaki se konvolucijski sloj sastoji od tri dijela (faze): faza u kojoj se obavlja konvolucija, faza u kojoj se pomoću aktivacijske funkcije izračunava vrijednost konvolucije i faze u kojoj se obavlja izvlačenje. Svaka od ovih faza redom je opisana u nastavku poglavlja.

Slika konvolucijsku neuronsku mrežu 2.6 prikazuje *LeNet-5* koja je korištena u [28]. U tome prikazu konvolucija objedinjuje fazu konvolucije i izračunavanja aktivacijske funkcije, dok je faza izvlačenja prikazana zasebno kako bi se lakše mogla prikazati dimenzionalnost svakog pojedinog sloja. Bitno je napomenuti kako je svaka mapa značajki nastala korištenjem različite jezgre.

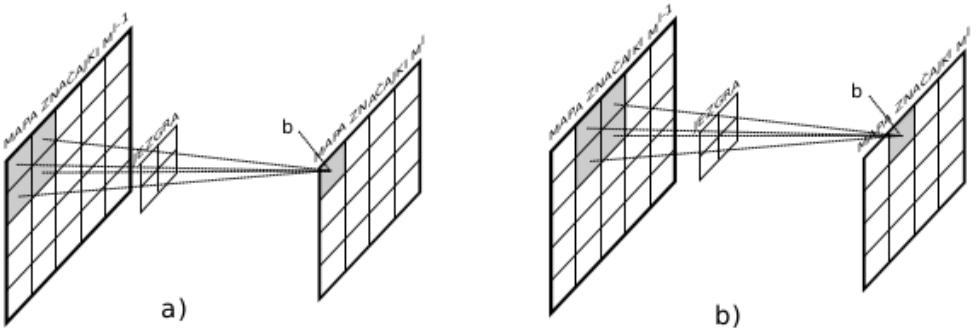
Nakon svih konvolucijskih slojeva možemo staviti potpuno povezanu neuronsku mrežu, ali možemo i izlaze kaskade konvolucijskih slojeva iskoristiti kao ulaz u neki drugi algoritam strojnog učenja.



Slika 2.6: Konvolucijska neuronska mreža *LeNet-5* (iz [28])

Faza konvolucije

Konvolucija se obavlja na način da jezgrom prolazimo po ulazu, množimo vrijednosti ulaza s vrijednostima jezgre, zbrajamo te vrijednosti i na njih dodajemo vrijednost praga (engl. *bias*). Veličina jezgre i korak kojim pomičemo jezgru po ulazu su parametri koji se moraju definirati prije ovog koraka. Na slici 2.7 taj korak iznosi 1, a veličina jezgre je 2×2.



Slika 2.7: Faza konvolucije. a) i b) označavaju dva uzastopna koraka

Aktivacijske funkcije

Aktivacijska funkcije u neuronskoj mreži transformira ulaznu vrijednost u izlaznu vrijednost neurona. Dva bitna faktora određuju kvalitetu aktivacijske funkcije: način na koji se nosi s problemom iščezavajućeg gradijenta te raspršenošću.

Problem iščezavajućeg gradijenta će biti detaljnije opisan u poglavlju 2.5.2.

Problem raspršenosti (engl. *sparsity*) je biološki inspiriran. Naime, u mozgu čovjeka nikad se neće dogoditi slučaj da se svi neuroni aktiviraju u istome trenutku, pa se iz tog razloga niti svi neuroni u neuronskoj mreži ne bi trebali aktivirati svi odjednom.

Od samih početaka neuronskih mreža najpopularnija aktivacijska funkcija je sigmoidalna. Međutim, svojstva te funkcije zbog kojih onda za jako male pomake poprima vrijednosti 0 ili 1 stvaraju problem kod dubokih neuronskih mreža te se javlja problem iščezavajućeg gradijenta. Autori u [19] predlažu novu funkciju, tzv. ispravljač (engl. *rectifier*) (formula 2.10) koja biološke procese

$$f(x) = \max(0, x) \tag{2.10}$$

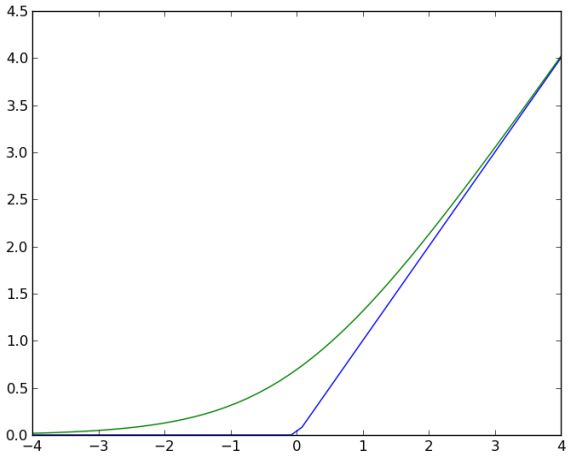
opisuje bolje od sigmoidalne funkcije. Neuron koji koristi ovu funkciju zove se ispravljačka linearna jedinica (engl. *rectified linear unit, ReLU*). Kako je ispravljačka funkcija nederivabilna, postoji njena zaglađena verzija koja se zove funkcija *softplus*, a dana je formulom:

$$f(x) = \ln(1 + e^x)$$

Zanimljivo svojstvo ove funcije je da njena derivacija:

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

što je formula sigmoidalne funkcije.



Slika 2.8: Ispravljačka i *softplus* funkcija

Izvlačenje

O izvlačenju je već prije bilo riječi pa će ovdje biti opisana novija ideja koja je nastala kao posljedica otkrivanja ispravljačke funkcije, a to je stohastičko maksimalno izvlačenje (engl. *stochastic max-pooling*).

Stohastičko maksimalno izvlačenje obavlja se na način da se svakom izlazu aktivacijske funkcije koji nije 0 dodijeli neka vjerojatnost koja je proporcionalna s vrijednosti aktivacijske funkcije. Tada se izvlačenje radi tako da se, s obzirom na tu distribuciju, slučajno odabere neka izlazna vrijednost. Prema [47] ova metoda neće brzo prenaučiti model pa je samim time moguć dulji trening.

2.4.5. Konvolucijske neuronske mreže za obradu prirodnog jezika

Iako se konvolucijske neuronske mreže danas dominantno koriste u računalnom vidu, moguće ih je prilagoditi na probleme obrade prirodnog jezika. Neki od primjera uspješne prilagodbe mogu se naći u [26] i [46], gdje autori predlažu da se na ulaz konvolcijske neuronske mreže dovedu raspodijeljeni vektori riječi koji su dostupni (npr. *word2vec* [35] [36] [37] za engleski jezik). Problem tog pristupa je što mnogi drugi jezici nemaju istrenirane raspodijeljene vektore riječi koji bi se mogli iskoristiti, pa je potrebno naučiti vektore riječi za odgovarajući jezik.

Kako bi to izbjegli, autori u [24] predlažu dva modela koji će na ulazu dobiti tekst (bilo koje duljine) te će iz tog teksta sami konstruirati vektore koje će se onda koristiti kao ulazi neuronske mreže.

Neka je $D = \{w_1, w_2, \ldots\}$ rečenica nekog teksta, a V rječnik nastao na temelju nekog (ne nužno istog) teksta. Na tekst možemo gledati kao na sliku dimenzija $|D| \times 1$ s $|V|$ kanala (engl. *channels*) te svaku riječ prestaviti kao $|V|$-dimenzijski vektor. Na primjer, ako imamo rečenicu "Auto se pokvario" i rječnik $V = \{''auto'', '' bicikl'', '' se'', '' kupiti'', '' pokvario''\}$, rečenicu možemo predstaviti kao vektor:

$$\left[\underbrace{1\ 0\ 0\ 0\ 0}_{\text{auto}} \,|\, \underbrace{0\ 0\ 1\ 0\ 0}_{\text{se}} \,|\, \underbrace{0\ 0\ 0\ 0\ 1}_{\text{pokvario}}\right] \tag{2.11}$$

Ovakva metoda kodiranja riječine u vektore zove se *one-hot encoding*.

Prvi predloženi model je tzv. sekvencijska konvolucijska neuronska mreža (engl. *seq-CNN*). Definira se klizni prozor veličine p kojim prolazimo po slici te iz svakog kliznog prozora stvorimo vektor. Vektori se grade na način da se vektori riječi iz 2.11 spajaju (engl. *concatenate*) u jedan vektor. Tako nastao skup vektora ulaz je u neuronsku mrežu.

Problem ovog modela je u tome što je dimenzionalnost rječnika, a samim time i dimenzionalnost vektora kliznog prozora (koja iznosi $p|V|$), može biti jako velika. To će voditi do toga da je broj težina koje treba istrenirati prevelik te će

trening biti spor i, u slučaju nedostatka podataka, neefikasan.

Taj problem rješava drugi predloženi model – *bow-CNN*. Veličina vektora kliznog prozora u ovom slučaju ovisi samo o dimenzionalnosti rječnika pa ne moramo smanjivati veličinu kliznog prozora nauštrb kvalitete modela. Vektori se u ovom slučaju grade tako da se vektori iz 2.11 spajaju logičkom operacijom *ILI* (engl. *OR*).

Izvlačenje kod konvolucijskih neuronskih mreža za obradu prirodnog teksta ne može biti izvedeno na jednak način kao i kod slika jer se slike kod problema koji se bave slikama normaliziraju na jednaku veličinu. Kako je kod rečenica tako nešto nemoguće, ideja je da se kod izvlačenja odredi broj izlaza iz sloja koji će obavljati izvlačenje te se dinamički određuje područje koje će biti pokriveno jednim izvlačenjem.

2.5. Povratne neuronske mreže

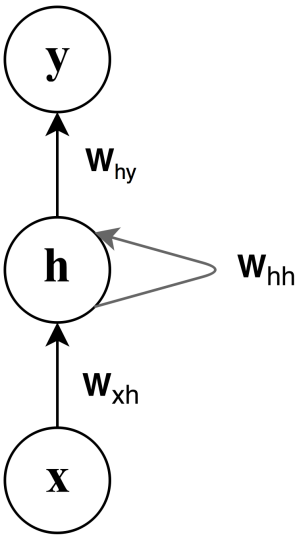
Povratne neuronske mreže (engl. *recurrent neural networks, RNN*) tip su neuronskih mreža koje nam omogućavaju rad s nizom vektora. Naime, obične unaprijedne neuronske mreže (u što spadaju i konvolucijske neuronske mreže) su modeli koji mogu raditi s unaprijed definiranom veličinom ulaza i izlaza. Uz to, preslikavanje ulaza u izlaz ostvaruje se unaprijed definiranim brojem skrivenih slojeva među kojima se ne pretpostavlja nikakva zavisnost.

Ako na ulazu i/ili izlazu mreže želimo imati niz (npr. znakovi ili slikovni elementi) ili zvuk kod kojih trenutno stanje (npr. riječ) ovisi i prethodnim stanjima, tada želimo naučiti zakonitosti na temelju kojih se ta stanja mijenjaju. Za učenje tih zakonitosti koristimo povratne neuronske mreže.

Povratne neuronske mreže imaju vrlo jednostavnu strukturu: ulazni sloj (oznaka $\boldsymbol{x}$), skriveni sloj (oznaka $\boldsymbol{h}$) i izlazni sloj (oznaka $\boldsymbol{y}$).

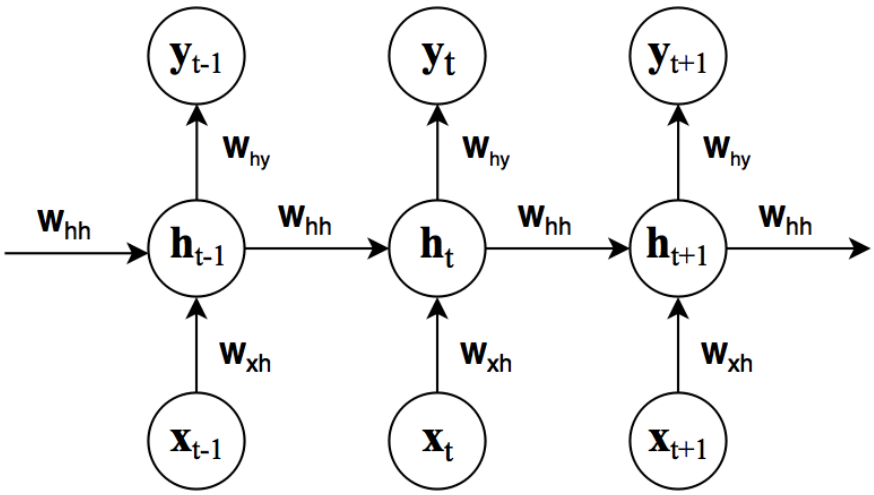
Svaki sloj ima pripadajuće težine. Ulazni i izlazni sloj imaju jednaku funkciju kao i kod običnih, unaprijednih neuronskih mreža.

Skriveni sloj je ono što povratnim neuronskim mrežama omogućava učenje zakonitosti te predstavlja neku vrstu memorije – pamti što se do tada pojavilo na ulazu mreže. Iz ovoga bi se moglo zaključiti kako povratne neuronske mreže mogu iskoristiti informacije iz proizvoljno dugih nizova, ali to nije moguće zbog problema o kojima će više riječi biti kasnije. Zbog tog ograničenja, uvodi se pojam "odmotavanja" mreže (engl. *unrolling, unfolding*). Na slici 2.10 vidimo mrežu koja je odmotana tri puta. Trenutni sloj (engl. *unfold, unroll*) označen je



Slika 2.9: Povratna neuronska mreža

s t.



Slika 2.10: Povratna neuronska mreža odmotana tri puta

Svi slojevi dijele težine tako da svaka povratna neuronska mreža uvijek ima tri skupa težina. Na taj se način značajno smanjuje broj parametara koje mreža treba naučiti. $\boldsymbol{W}_{xh}$ označava težine koje se nalaze između ulaznog i skrivenog sloja, $\boldsymbol{W}_{hh}$ označava težine koje se nalaze između dva skrivena sloja (skriveni slojevi iz koraka $t-1$ i t), a $\boldsymbol{W}_{hy}$ označava težine koje se nalaze između skrivenog i izlaznog sloja mreže.

Koristeći ove oznake možemo napisati formule prema kojima se određuje izlaz povratne neuronske mreže [5].

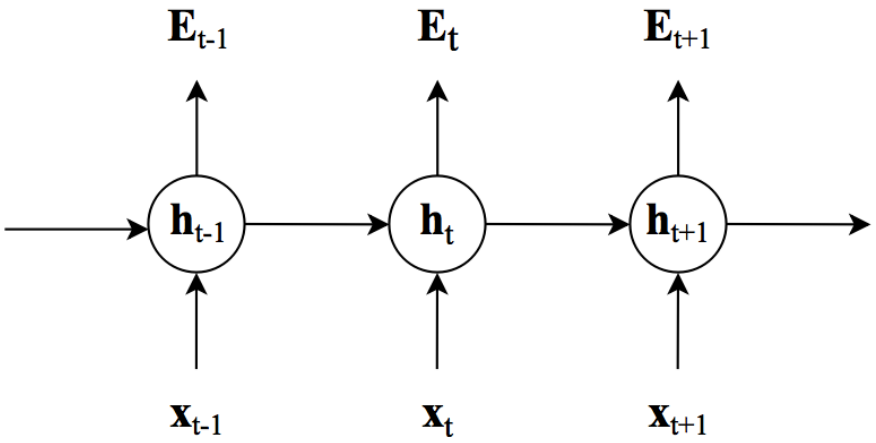
$$\begin{aligned} \boldsymbol{h_t} &= tanh\left(\boldsymbol{W}_{hh} \cdot \boldsymbol{h}_{t-1} + \boldsymbol{W}_{xh} \cdot \boldsymbol{x_t}\right) \\ \boldsymbol{y_t} &= \boldsymbol{W}_{hy} \cdot \boldsymbol{h_t} \end{aligned} \tag{2.12}$$

Kao što se može vidjeti iz formule 2.12, izračun izlaza povratne neuronske mreže svodi se na skalarni umnožak (engl. *dot product, inner product*) matrica i vektora. Prilikom izračuna vrijednosti skrivenog sloja u obzir se uzima prethodno stanje mreže i novi ulaz koji se potom funkcijom *tanh* ograničava na prostor $[-1, 1]$. Izlaz se u povratnim neuronskim mrežama često računa kao funkcija *softmax* kako bi se dobile vjerojatnosti klasa koje se predviđaju. Kako bi se poboljšala učinkovitost povratnih neuronskih mreža, moguće ih je vrlo lako povezati tako što se izlaz jedne koristi kao ulaz druge mreže. Međutim, treba imati na umu da će mreži trebati više vremena da nauči.

Bitan parametar ovih mreža je veličina vektora skrivenog sloja ("memorije") te ga treba pažljivo odabrati. Što je vektor veći, to će moći naučiti više koncepata iz danih podataka, ali bi mogao, ako je prevelik, značajno usporiti proces učenja mreže.

2.5.1. Učenje povratnih neuronskih mreža

Metoda učenja povratnih neuronskih mreža zove se metoda propagacije greške unatrag kroz vrijeme (engl. *backpropagation through time, BPTT*). Kao što je iz naziva vidljivo, metoda je slična metodi učenja običnih, unaprijednih neuronskih mreža s razlikom što se ovdje greška mora propagirati kroz više vremenskih perioda zbog odmatanja povratne neuronske mreže.



Slika 2.11: Greške povratne neuronske mreže

Ideja je ista kao i kod metode propagacije greške unatrag samo što ovdje postoje tri skupa težina po kojima treba parcijalno derivirati funkciju koja se optimizira te zbrojiti gradijente u svakom koraku. Međutim, kao funkcija koja se optimizira, ovdje (a i općenito) se koristi negativna *log* izglednost prikazana formulom 2.13.

$$E\left(\boldsymbol{\theta}\right) = -\frac{1}{N}\sum_{\boldsymbol{x},\boldsymbol{y}}^{N} log\, p\left(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta}\right) \tag{2.13}$$

Većina funkcija koje se koriste na izlazu neuronskih mreža i time određuju oblik funkcije p u sebi sadrži izraz u kojem se potencira Eulerov broj e. Te funkcije imaju područja zasićenja u kojima je iznos derivacije blizu 0. U tom području, gradijentnim metodama je teško učiti jer su pomaci mali te je učenje sporo te logaritmiranje tih izraza rješava taj problem [11]. Ako se s $\boldsymbol{l_t}$ označi točan izlaz mreže (bilo jedan bilo više njih) u trenutku t, tada je greška koju radi povratna neuronska mreža izražena pomoću *log* izglednosti dana formulom 2.14.

$$E\left(\boldsymbol{y},\boldsymbol{l}\right) = -\sum_{t} \boldsymbol{l_t} \cdot log\ \boldsymbol{y_t} \tag{2.14}$$

Ako se promotre formule 2.12 te se pravilom lanca pokušaju naći izrazi za izračun gradijenta, uočava se kako je za izračun izraza za težine $\boldsymbol{W}_{hy}$ potrebno promatrati vrijednosti iz trenutka t, ali ne i iz prošlosti, dok je za preostala dva skupa težina potrebno vratiti se u prošlost. Izrazi za oba slučaja su prikazani formulama 2.15, 2.16 i 2.17.

$$\frac{\partial E_{t+1}}{\partial \boldsymbol{W}_{hy}} = \frac{\partial E_{t+1}}{\partial \boldsymbol{y}_{t+1}} \frac{\partial \boldsymbol{y}_{t+1}}{\partial \boldsymbol{W}_{hy}} \tag{2.15}$$

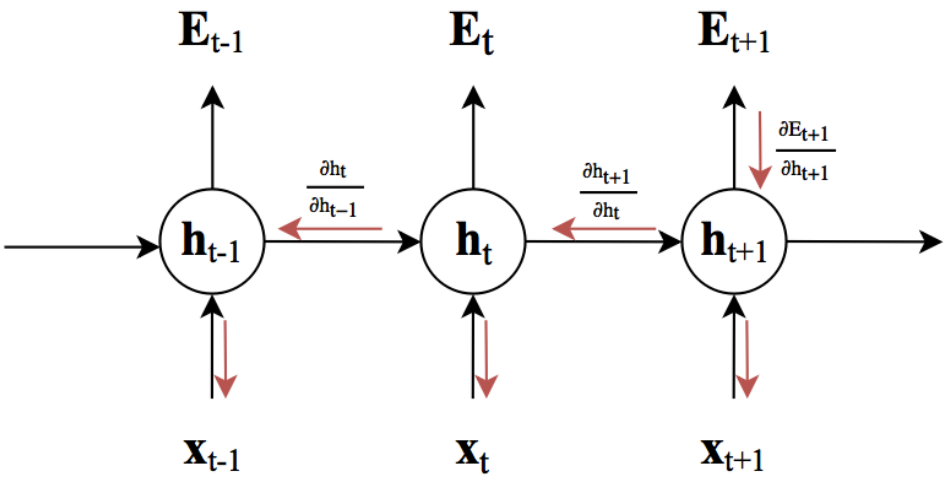
Gore spomenuta ovisnost o vrijednostima iz prethodnih koraka vidi se u izrazu 2.16. Naime, $\boldsymbol{h}_{t+1}$ ovisi o $\boldsymbol{h}_t$ koji ovisi o $\boldsymbol{h}_{t-1}$ itd.

$$\frac{\partial E_{t+1}}{\partial \boldsymbol{W}_{hh}} = \frac{\partial E_{t+1}}{\partial \boldsymbol{y}_{t+1}} \frac{\partial \boldsymbol{y}_{t+1}}{\partial \boldsymbol{h}_{t+1}} \frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{W}_{hh}} \tag{2.16}$$

Zbog toga se za izračun konačnog izraza za ažuriranje težina $\boldsymbol{W}_{hh}$ i $\boldsymbol{W}_{xh}$ moraju zbrojiti gradijenti po svim koracima kojih ima K, što je prikazano formulom 2.17.

$$\frac{\partial E_{t+1}}{\partial \boldsymbol{W}_{hh}} = \sum_{k}^{K} \frac{\partial E_{t+1}}{\partial \boldsymbol{y}_{t+1}} \frac{\partial \boldsymbol{y}_{t+1}}{\partial \boldsymbol{h}_{t+1}} \frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_{t+1-k}} \frac{\partial \boldsymbol{h}_{t+1-k}}{\partial \boldsymbol{W}_{hh}} \tag{2.17}$$

Na slici 2.11 prikazana je povratna neuronska mreža s izlazima koji predstavljaju grešku, a na slici 2.12 se vidi smjer gradijenata.



Slika 2.12: Smjer gradijenata u povratnoj neuronskoj mreži

Gornja razmatranja dovode do povećanja broja matrica u umnošku prilikom izračuna iznosa gradijenta u pojedinom sloju što dovodi do problema pri učenju optimalnih težina. Jasno je kako će vrijeme potrebno za jedan korak ažuriranja težina povratne neuronske mreže (neovisno koju metodu odabaremo), trajati poprilično dugo ako odlučimo puno puta odmotati mrežu. Kako se rekurzivne mreže koriste za nizove te je uvijek bolje ako se koristi puno podataka iz prošlosti (npr. kod teksta), vrlo je teško naučiti takvu mrežu jer je to vremenski zahtjevno. Alternativa je skraćena metoda propagacije unatrag kroz vrijeme (engl. *truncated backpropagation through time*) [44]. Ta metoda napravi k_1 iteracija povratne neuronske mreže, nakon kojih odvrti metodu propagacije greške unatrag kroz vrijeme za k_2 koraka gdje je $k_2 < k_1$. Time se omogućava da mreža nauči dovoljno iz viđenih podataka, ali i da zapamti neke zakonitosti iz daleke prošlosti.

2.5.2. Problem gradijenata

Kao što smo vidjeli u potpoglavlju 2.5.1, povratne neuronske mreže koriste metodu (skraćene) propagacije greške unatrag kroz vrijeme kako bi naučile optimalne parametre. Problem te metode su Jacobijeve matrice koje se koriste prilikom deriviranja optimizacijske funckije jer njihov umožak ili eksplodira ili ode u nulu. Prvi slučaj zove se problem eksplodirajućeg gradijenta (engl. *exploding gradients problem*), a drugi problem iščezavajućeg gradijenta (engl. *vanishing gradient problem*). Problem je u iznosu svojstvenih vektora matrice $\boldsymbol{W}_{hh}$, a dokaz toga se može naći u [41].

Problem eksplodirajućeg gradijenta rješava se jednostavnim postupkom odsi-

jecanja gradijenta normom (engl. *norm clipping*), kao što je prikazano algoritmom 1.

Algoritam 1 Odsijecanje gradijenta normom

$\boldsymbol{g} = todo$
if $||\boldsymbol{g}|| > prag$ **then**
  $\boldsymbol{g} = \frac{prag}{||\boldsymbol{g}||} \cdot \boldsymbol{g}$
end if

Problem iščezavajućeg gradijenta je teži jer ne postoji očit način kako ga riješiti. Najjednostavnije rješenje je promijeniti način incijalizacije težina ili koristiti regularizaciju, ali to ne mora nužno pomoći. Druga metoda je promjena aktivacijske funkcije tj. korištenje ispravljačkog neurona čija je derivacija 0 ili 1 te iznosi ažuriranje neće otići u nulu nakon množenja većeg broja matrica.

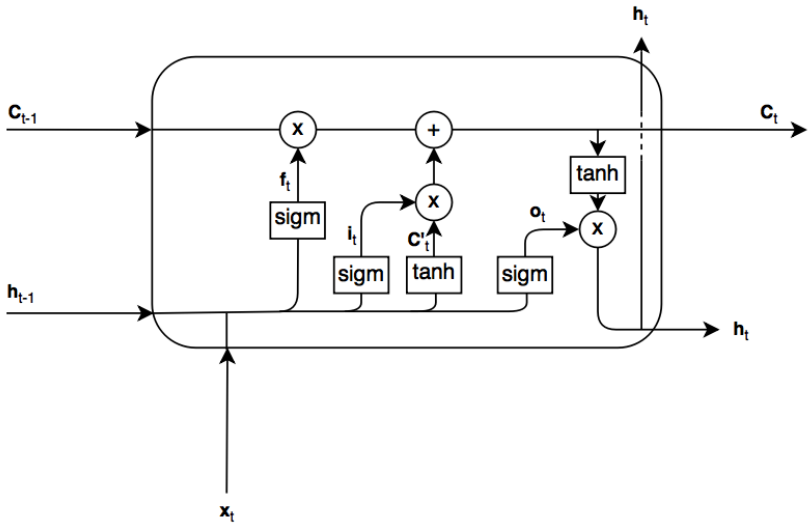
Međutim, najefikasnija metoda izbjegavanja iščezavajućeg gradijenta je korištenje posebne vrste ćelija/mreža – mreža s dugom kratkoročnom memorijom (engl. *long short term memory, LSTM*) ili povratne neuronske mreže s vratima (engl. *gated recurrent unit, GRU*) [8] [16]. Nazivlje koje se koristi može ponekad biti zbunjujuće, no treba imati na umu da povratna neuronska mreža uvijek ima istu strukturu: ulazni, skriveni i izlazni sloj, a izračun vrijednosti slojeva ovisi o korištenoj ćeliji. Na ćeliju se može gledati kao na crnu kutiju koja odradi neki izračun. Npr. formule 2.12 predstavljaju običnu ćeliju povratne neuronske mreže. U slučaju kada se koristi termin mreže s dugom kratkoročnom memorijom, misli se na povratnu mrežu koja za izračun koristi ćelije s dugom kratkoročnom memorijom.

U nastavku je detaljnije objašnjenje mreže s dugom kratkoročnom memorijom, koja se koristi u ovom radu.

2.5.3. Mreža s dugom kratkoročnom memorijom

Mreže s dugom kratkoročnom memorijom posebna su vrsta povratnih neuronskih mreža koje omogućuju učenje dugih ovisnosti u podacima. Npr. ako osoba piše tekst o sebi u kojem na početku navodi čime se bavi, a kasnije (nakon nekoliko rečenica) priča o tome kako je dobila trenutni posao, mreža s dugom kratkoročnom memorijom sposobna je naučiti povezati te dvije stvari. Struktura ćelije s dugom kratkoročnom memorijom prikazana je na slici 2.13. Može se primijetiti kako je struktura ćelije puno kompliciranija od strukture ćelije koja se koristi u

običnoj povratnoj neuronskoj mreži, ali takva struktura omogućava učenje onog što obična povratna mreža ne može.



Slika 2.13: Mreža s dugom kratkoročnom memorijom

Ključna ideja ćelije s dugom kratkoročnom memorijom je njeno stanje (engl. *cell state*) $\boldsymbol{C}_t$. Stanje ćelije pamti informacije koje su trenutno relevantne mreži (npr. tematiku trenutnog dijela teksta). U stanje ćelije nove se informacije mogu dodati (ako su vezane za tematiku koja je trenutno zapamćena) ili izbrisati (ako nisu vezane za tematiku). O tome što će se dogoditi, odlučuju vrata (engl. *gates*) koristeći sigmoidnu funkciju.

Prvi korak su vrata kojima se regulira količina informacija iz ćelije stanja ćemo odbaciti (engl. *forget gate*), i to na temelju prethodnog stanja skrivenog sloja $\boldsymbol{h}_{t-1}$ i trenutnog ulaza $\boldsymbol{x}_t$ prema izrazu 2.18. Količina informacija određuje se izlazom koji je u rasponu od 0 do 1 te je različit za svaki element vektora stanja $\boldsymbol{C}_t$.

$$\boldsymbol{f}_t = sigm\left(\boldsymbol{W}_f \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_f\right) \tag{2.18}$$

U sljedećem koraku određuje se koje nove informacije se pamte te se nadodaju na trenutno stanje ćelije. Prvo sigmoidna funkcija na temelju trenutnog ulaza i prethodnog stanja skrivenog sloja odredi koji dijelovi stanja ćelije će se ažurirati, a potom se na temelju istih podataka određuje novo stanje ćelije koje će se modificirati i nadodati na trenutnu stanje. To je prikazano formulama 2.19 u kojima $\circ$ označava operaciju koja se provodi pojedinačno po elementima (engl. *element-wise operation*).

$$\begin{aligned} \boldsymbol{i}_t &= sigm\left(\boldsymbol{W}_i \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_i\right) \\ \boldsymbol{C'}_t &= tanh\left(\boldsymbol{W}_C \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_C\right) \\ \boldsymbol{C}_t &= \boldsymbol{f}_t \circ \boldsymbol{C}_{t-1} + \boldsymbol{i}_t \circ \boldsymbol{C'}_t \end{aligned} \tag{2.19}$$

Posljednji korak je izračun skrivenog stanja, odnosno izlaza mreže. Kao i u svim koracima do sad, prvo se odredi koliko i koji dijelovi trenutnog stanja ćelije će se dovesti na izlaz te se potom izračuna izlaz (formule 2.20).

$$\begin{aligned} \boldsymbol{o}_t &= sigm\left(\boldsymbol{W}_o \cdot [\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_o\right) \\ \boldsymbol{h}_t &= \boldsymbol{o}_t \circ \boldsymbol{C}_t \end{aligned} \tag{2.20}$$

Iz svih navedenih formula može se primijetiti kako su formule koje vrijede kod povratnih neuronskih mreža prisutne i u mrežama s dugom kratkoročnom memorijom uz dodatak dijelova koji omogućavaju pamćenje dugih ovisnosti u ulaznim podacima.

Bitno je napomenuti kako postoji mnogo varijanti mreža s dugom kratkoročnom memorijom (od kojih su neke dobile i posebno ime, kao npr. povratna neuronska mreža s vratima) koje su objavljene ([16] ili [18] su samo neke od njih), ali i puno onih koje nisu. Naime, mreže s dugom kratkoročnom memorijom imaju puno prostora za eksperimentiranje bilo s funkcijama koje se koriste ili s vezama između pojedinih vrata. Međutim, pokazuje se [21] kako većina predloženih modifikacija radi otprilike isto te svaka može imati prednost nad ostalima ako se prilagodi problemu koji se rješava.

2.6. Algoritmi ažuriranja težina neuronskih mreža

Kao što je već spomenuto u poglavlju o učenju neuronskih mreža, algoritam propagacije greške unatrag najpoznatiji je algoritam za njihovo učenje. Zapravo, propagacija greške unatrag nije algoritam učenja već metoda koju koriste algoritmi učenja za izračun iznosa gradijenata, ali se u literaturi najčešće predstavlja kao algoritam. Osnovi algoritam koji koristi metodu propagacije greške unatrag je gradijentni spust koji ju kombinira sa stopom učenja (engl. *learning rate*) η. Međutim, kako su se neuronske mreže razvijale i postajale sve veće, znanstvenici su primijetili kako je običan gradijentni spust prejednostavan za uspješno učenje tako velikih mreža. U nastavku ćemo razmotriti algoritam gradijentnog spusta, a potom i neka od predloženih poboljšanja koja se danas najčešće koriste [7].

2.6.1. Gradijentni spust

Gradijentni spust je metoda minimizacije funckije cilja $E(\boldsymbol{\theta})$, gdje je $\boldsymbol{\theta}$ vektor parametara neuronske mreže. Ime je dobio po tome što, za ažuriranje parametara, koristi gradijent funkcije cilja $\nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta})$ – vektor njenih prvih derivacija po svakom parametru. Kako gradijent, po definiciji, označava porast funkcije cilja, gradijentni spust ažuriranje provodi u suprotnom smjeru gradijenta te, dodatno, njegov iznos skalira odabranom stopom učenja (formula 2.21).

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \cdot \nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta}) \tag{2.21}$$

Postoje tri varijante gradijentnog spusta: grupni (engl. *batch*), stohastički (engl. *stochastic*) i mini-grupni (engl. *mini-batch*). Formula je za sve tri varijante jednaka, a jedina razlika je u kojem će se trenutku ažurirati parametri. Kod batch verzije, oni se ažuriraju nakon što sve primjere iz skupa podataka predočimo mreži; kod stohastičke verzije nakon svakog primjera, a kod mini-batch verzije nakon svakih nekoliko primjera (parametar algoritma).

Praktično, najboljom se pokazuje mini-batch verzija. Naime, batch verzija zahtijeva da se u memoriji nalazi cijeli skup podataka, što je u današnje vrijeme često neizvedivo. Stohastička verzija ažurira parametre nakon svakog koraka te se zbog toga vrijednost funkcije cilja često pokvari, a i istraživanje prostora rješenja može zapeti u krivom području. Mini-batch verzija radi s nešto više primjera koje možemo držati u memoriji te je kombinacija prethodne dvije varijante i danas je najčešće korištena varijanta.

Ono što je glavni problem ovog algoritma (bilo koje varijante) je odabir stope učenja. Ako je ona premala, učenje je sporo, a ako je prevelika, proces učenja divergira. Čak i ako odaberemo povoljnu stopu učenja (takvu da na početku daje dobre rezultate), gotovo sigurno će u jednom trenutku ona izgubiti svoju efikasnost. To možemo riješiti na način da dinamički mijenjamo stopu (ovisno o broju dosad izvršenih iteracija) (engl. *annealing*), ali i dalje imamo problem što jedan algoritam promjene stope neće raditi za svaki skup podataka jednako. Također, broj parametara u današnjim neuronskim mrežama vrlo velik te za sve parametre koristimo samo jednu stopu učenja što možda i nije najbolje ako, na primjer, pokušavamo naučiti mrežu da svaki njen dio reagira na neku drugu značajku u ulaznim podacima.

Zbog navedenih problema, predloženo je nekoliko naprednijih verzija ovog algoritma.

2.6.2. Moment

Ideja ove metode vuče korijene iz fizike i kretanja loptice po brdovitom terenu. Na početku, loptica miruje (brzina joj je 0). Ako lopticu pustimo da se slobodno kreće po terenu, ona će nakon nekog vremena imati neku brzinu koja je rezultat nagiba terena, ali i sile teže koja ju vuče prema najnižoj točki terena. U formuli 2.22 vidimo izraze za ažuriranje parametara koristeći ovu metodu. Koeficijent μ simulira otpor zraka i trenje koje usporava lopticu te se obično koriste vrijednosti $0.5, 0.9, 0.95, 0.99$ [2], ali može se koristiti i postepeno smanjivanje stope ovisno o broju koraka.

$$\begin{aligned} \boldsymbol{v} &= \mu \cdot \boldsymbol{v} + \eta \cdot \nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta}) \\ \boldsymbol{\theta} &= \boldsymbol{\theta} - \boldsymbol{v} \end{aligned} \tag{2.22}$$

2.6.3. Nesterovljev moment

Ako teren ili, u slučaju učenja neuronskih mreža, područje pretraživanja ima mjesta na kojima se iz velike nizine brzo dolazi do velike uzbrdice, običan moment neće biti svjestan toga unaprijed te će loptica otići visoko na uzbrdicu. Da bi se takvo ponašanje ublažilo, umjesto da se gradijent računa u trenutnoj točci, računa se u točci koja će otprilike odgovarati sljedećoj poziciji vektora parametara te se ta metoda zove Nesterovljev moment. U formuli 2.23 vidimo tu aproksimaciju u prvom izrazu.

$$\begin{aligned} \boldsymbol{v} &= \mu \cdot \boldsymbol{v} + \eta \cdot \nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta} - \mu \cdot \boldsymbol{v}) \\ \boldsymbol{\theta} &= \boldsymbol{\theta} - \boldsymbol{v} \end{aligned} \tag{2.23}$$

2.6.4. *Adagrad*

Dosad spomenute metode tijekom cijelog procesa učenja koriste istu stopu učenja. *Adagrad* je metoda koja prilikom svakog ažuriranja iznos smanji za određeni koeficijent. Taj koeficijent ovisi o iznosu prethodnih gradijenata na način prikazan formulom 2.24. Iz formule se jasno vidi kako će se parametri koji imaju veliki iznos gradijenta ažurirati u manjim koracima od parametara čiji gradijent je mali. Parametar ϵ u nazivniku sprječava dijeljenje s nulom te se većinom postavlja na vrijednost 10^{-8}.

$$\begin{aligned} \boldsymbol{acc} &= \boldsymbol{acc} + \nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta})^2 \\ \boldsymbol{\theta} &= \boldsymbol{\theta} - \eta \cdot \frac{\nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta})}{\sqrt{\boldsymbol{acc}} + \epsilon} \end{aligned} \tag{2.24}$$

Međutim, ova metoda je problematična jer cijelo vrijeme zbraja kvadrirane iznose gradijenata te oni nakon određenog broja koraka smanje iznos ažuriranja na vrijednost koja je zanemariva te učenje prestaje, a možda postoji puno bolje rješenje.

2.6.5. *RMSprop*

RMSprop vrlo je sličan metodi *Adagrad*, ali rješava problem smanjivanja iznosa ažuriranja tako što u $\boldsymbol{acc}$ varijablu dodaje samo dio gradijenta, a i dio starog gradijenta "zaboravlja".

$$\begin{aligned} \boldsymbol{acc} &= \mathit{decay\_rate} \cdot \boldsymbol{acc} + (1 - \mathit{decay\_rate}) \cdot \nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta})^2 \\ \boldsymbol{\theta} &= \boldsymbol{\theta} - \eta \cdot \frac{\nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta})}{\sqrt{\boldsymbol{acc}} + \epsilon} \end{aligned} \tag{2.25}$$

Varijabla *decay_rate* se obično postavlja na vrijednost 0.9 ili veću.

2.6.6. *Adam – Adaptive Moment Estimation*

Prethodne dvije metode prilagođavaju stopu učenja podacima i modelu, ali koriste trenutni gradijent. Ideja algoritma *Adam* je kombinacija algoritma *RMSprop* i momenta.

$$\begin{aligned} \boldsymbol{m} &= \beta_1 \cdot \boldsymbol{m} + (1 - \beta_1) \cdot \nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta}) \\ \boldsymbol{v} &= \beta_2 \cdot \boldsymbol{v} + (1 - \beta_2) \cdot \nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta})^2 \\ \boldsymbol{\theta} &= \boldsymbol{\theta} - \eta \cdot \frac{\hat{\boldsymbol{m}}}{\hat{\boldsymbol{v}} + \epsilon} \end{aligned} \tag{2.26}$$

Može se uočiti da se u izrazu za ažuriranje parametara koriste neke druge varijable od onih koje su izračunate. Razlog tome je što , u početku, jer se i $\boldsymbol{m}$ i $\boldsymbol{v}$ početno stavljaju na 0, iznosi ažuriranja imaju male vrijednosti sve dok se gradijenti ne "nakupe". Kako bi se to izbjeglo, radi se korekcija na način prikazan formulom 2.27 gdje su β_1 i β_2 dignute na potenciju trenutnog broja iteracija.

$$\begin{aligned} \hat{\boldsymbol{m}} &= \frac{\boldsymbol{m}}{1 - \beta_1^t} \\ \hat{\boldsymbol{v}} &= \frac{\boldsymbol{v}}{1 - \beta_2^t} \end{aligned} \tag{2.27}$$

Vrijednosti koje se najčešće danas koriste su: 0.9 za β_1 i 0.999 za β_2.

2.6.7. Zaključak i dodatne metode

Od svih prethodno nabrojanih metoda optimizacije, kao najbolje rješenje se nameće *Adam* jer je spoj nekoliko ideja, a i za većinu praktičnih problema radi najbolje.

Također, postoje druge metode/savjeti kako poboljšati učenje modela. Dva jednostavnija su promjena redoslijeda kojim se primjeri predočuju mreži (engl. *shuffling*) te praćenje greške na skupu za provjeru (engl. *validation set*) te zaustavljanje ako ona ne pada dovoljno (engl. *early stopping*).

Nešto složenija je tehnika normalizacije *batch normalization*. Ideja je, nakon svakog izračuna vrijednosti aktivacija (u potpuno povezanom sloju mreže), aktivacije svesti na jediničnu normalnu razdiobu. Time dobivamo veću otpornost mreže na loše početne vrijednosti parametara te loše odabranu stopu učenja.

3. Jezično modeliranje

Generalni uvod te problemi klasičnog pristupa problemu jezičnog modeliranja opisani su u uvodnom dijelu rada. Prije pregleda klasičnih metoda, ali i metoda temeljenih na dubokom učenju, treba spomenuti dvije mjere koje će se koristiti za mjerenje kvalitete modela jezika – zbunjenost i učestalost pogrešaka u riječima.

Mjera zbunjenosti (engl. *perplexity, PPL*) niza riječi $\boldsymbol{w}$ je definirana je formulom 3.1, a temelji se na teoriji informacije i pojmu unakrsne entropije [31].

$$PPL = 2^{-\frac{1}{K}\sum_{i=1}^{K} log_2 P(w_i|w_{1\ldots i-1})} \tag{3.1}$$

Prednost ove mjere je što se vrlo lako može izračunati te što entropija (po definiciji) označava koliko su dva izvora informacija slična. Zbog načina na koji je mjera zbunjenosti definirana, model koji ima manju vrijednost zbunjenosti je bolji jer to znači da je sličniji modelu koji je generirao stvarne podatke.

Problem mjere zbunjenosti je u tome što sakriva stvarne napretke u kvaliteti modela što se više približavamo granici najmanje moguće vrijednosti zbunjenosti koja iznosi 1 [31].

Mjera učestalosti pogrešaka u riječima (engl. *word error rate, WER*) daje broj pogrešaka između izlaza modela i točnog izlaza koji je dan u skupu podataka koji se koristi. Broj pogreška se računa pomoću tri operacije koje se provode nad izlazom modela: zamjena, brisanje ili umetanje simbola (riječi ili znaka). Cilj je primjenom tih operacija iz izlaza modela (npr. rečenice) dobiti pravi izlaz u minimalnom broju koraka. Formulom 3.2 prikazan je izraz kojim se dobiva učestalost pogrešaka u riječima. Slovo S označava broj zamjena, slovo D broj brisanja, slovo I broj umetanja, a slovo N ukupan broj simbola u točnom izlazu. Svaki od parametara u brojniku može se pomnožiti s nekim brojem kako bi se pojačao ili smanjio utjecaj te operacija na konačan rezultat.

$$WER = \frac{S + D + I}{N} \tag{3.2}$$

Ova metoda zapravo pokazuje koliko je jedan model bolji od nekog drugog. Naime, zbunjenost modela može značajno pasti, ali to neće nužno dovesti do smanjenja učestalosti pogrešaka u riječima. Problem ove metode je što previše naglaska stavlja na česte riječi koje nisu informativne, a i zamjene sličnih riječi, ali i potpuno različitih riječi se vrednuju jednako [31].

3.1. Statističko modeliranje jezika

Najpoznatiji jezični modeli temeljeni na statistici su n-gram jezični modeli. Broj n predstavlja broji riječi koji se promatra u danom trenutku. Za $n = 2$ (*bigram* model) te danu rečenicu s koja se sastoji od riječi $w_1 \dots w_l$ vjerojatnost pojave takve rečenice $p(s)$ računa se prema formuli 3.3. Za $n = 3$ (*trigram* model) koristi se formula 3.4, a poopćenje formule na proizvoljan n dano je formulom 3.5.

$$p(s) = \prod_{i=1}^{l} p(w_i|w_{i-1}) \tag{3.3}$$

$$p(s) = \prod_{i=1}^{l} p(w_i|w_{i-2}, w_{i-1}) \tag{3.4}$$

$$p(s) = \prod_{i=1}^{l} p(w_i|w_1 \dots w_{i-1}) \tag{3.5}$$

Kako zbroj svih vjerojatnosti mora biti 1, na početak i kraj rečenice dodaju se posebne oznake.

Procjena vjerojatnosti pojave nekog *bigrama* u tekstu može se procijeniti brojanjem njegovih pojava u tekstu pomoću kojeg učimo model. Ako se s $count(w_{i-1}w_i)$ označi broj pojava nekog *bigrama* u tekstu, tada vjerojatnost njegovog pojavljivanja možemo izračunati prema formuli 3.6 koja se zove procjena najveće izglednosti (engl. *maximum likelihood, ML*). Iako je formula dana za model *bigrama*, može se lako proširiti na proizvoljan n na identičan način kao i formule za određivanje vjerojatnosti rečenica.

$$p(w_i|w_{i-1}) = \frac{count(w_{i-1}w_i)}{\sum_{w_i} count(w_{i-1}w_i)} \tag{3.6}$$

3.1.1. Zaglađivanje

Gornje formule će za vjerojatnosti *bigrama* koji do tog trenutka nisu viđeni (npr. kada se model procjenjuje na skupu za testiranje) dati nulu. To, naravno,

nije dobro jer u skupu za učenje nikad neće biti svi *bigrami* koji postoje.

Tehnika kojom se rješava taj problem je zaglađivanje (engl. *smoothing*). Općenito, ideja je da se u brojnik i nazivnik formule za procjenu vjerojatnosti pojavljivanja n-grama dodaju koeficijenti koji će zadržati vjerojatnosnu interpretaciju rezultata, ali i omogućiti da se dotad neviđenim n-gramima pridijeli vjerojatnost pojavljivanja koja je nešto veća od nule.

Ovi problemi su uočeni relativno rano te je puno vremena uloženo u njihovo rješavanje, a posljedica toga je velik broj različitih metoda o kojima se više toga može pronaći u [15] i [20].

3.1.2. Napredne tehnike jezičnog modeliranja

Jezični modeli temeljni na n-gramima doživjeli su veliki uspjeh, ali, kako što je u uvodu spomenuto, prokletstvo dimenzionalnosti onemogućava ih da postignu zadovoljavajuće rezultate. Mikolov u [31] daje zanimljiv primjer rečenice s kojom n-grami imaju veliki problem upravo zbog načina na koji rade. Rečenica *"Zabava će biti u <dan_u_tjednu>"* je problematična iz razloga što model jezika temeljen na n-gramima neće uspjeti naučiti da umjesto dana u tjednu može ići ime bilo kojeg dana pa čak i za velik broj primjera za učenje. Točnije, čak i ako imamo dovoljno velik skup podataka, n-gram model neće uspjeti naučiti zakonitost da na to mjesto ide ime nekog dana u tjednu. Iz ovoga se vidi kako je jezik kompleksan sustav čije je učenje pomoću računala blisko povezano s umjetnom inteligencijom i učenjem jezika na način da se pokušavaju naučiti zakonitosti koje u njemu postoje.

Međutim, prije dubokog učenja pojavile su se neke druge tehnike koje su dale puno bolje rezultate od običnih modela n-grama.

Jezični modeli temeljeni na povijesti

Znanstvenici su istraživanjima uvidjeli da se riječi (pogotovo one rijetke) češće ponovo pojavljuju u tekstu ako se su nedavno pojavile. Ideja jezičnih modela temeljenih na povijesti (engl. *cache language models*) je da promatraju posljednjih nekoliko stotina riječi te tim riječima daje vjerojatnost veću od nule [23]. Prema [31] problem ovih modela je značajno smanjenje zbunjenosti koja je dovela do čestog korištenja modela bez prave evaluacije. Naime, prilikom raspoznavanja govora se greška koju model napravi sprema u privremenu memoriju (engl. *cache*) bez dodatnih provjera te se time povećava mogućnost ponovne pogreške.

Druga zamjerka je što se modeli testiraju na odvojenom skupu podataka koji ne sadrži dovoljno šumovitih ili dvoznačnih podataka, što bi značajno smanjilo dobivene rezultate, ali ni to nije dovoljno jer ti modeli postižu bolje rezultate samo u slučaju zbunjenosti, ali i dalje previše griješe kada se radi o raspoznavanju govora.

Modeli temeljeni na klasama

Ideja modela temeljenih na klasama (engl. *class based models*) je svaku riječ staviti u neku klasu koja se sastoji od nekoliko riječi te trenirati n-gram model nad tim klasama. Ovi modeli, za razliku od modela temeljenih na povijesti, daju puno bolje rezultate u raspoznavanju govora, ali su računski zahtjevniji [31].

3.2. Modeliranje jezika neuronskim mrežama

Prvi pokušaj modeliranja jezika neuronski mrežama je onaj Jeffa Elamana koji je koristio povratnu neuronsku mrežu za modeliranje jezika za koji je izgradio umjetnu gramatiku [17]. Prvi značajni pokušaj je rad Yoshue Bengia [12] u kojem je napravio usporedbu modela koji je predložio i dotad postojećih.

Nakon tog članka, objavljena je nekolicina članaka bez značajnijeg napretka u rezultatima čak i uz dodavanje novih značajki (npr. oznaka vrsta riječi). Također, neka istraživanja su se bavila problemom sporog učenja modela koje je onemogućavalo kompletno treniranje modela.

3.2.1. Jezični modeli temeljeni na unaprijednim neuronskim mrežama

Spomenuti Bengijev model svaku riječ predstavlja kao vektor koji na samo jednom mjestu ima 1, a na ostalima 0 (*one-hot encoded* vektor). Takav prikaz riječi se potom dijeljenom projekcijskom matricom P projicira u prostor niže dimenzije. Matrica P je dijeljena među svim riječima iz prošlosti, a dimenzije prostora u koji se projicira su 30, 60 ili 100. Nakon projekcijskog sloja dolazi skriveni sloj koji koristi neku nelinearnu funkciju (tangens hiperbolni ili sigmoidnu) čija se veličina kreće od 100 do 300. Izlazni sloj jednak je veličini rječnika (oko 50000 riječi) te predstavlja vjerojatnosnu razdiobu n-grama.

Problem ovog pristupa (koji i sam Bengio spominje) je izrazito velika računala zahtjevnost. Naime, na 40 običnih procesnih jedinica (engl. *CPU*) modelu je

trebalo tjedan dana da prođe kroz 14 milijuna riječi te je stoga model treniran puno manje nego što je zapravo trebao biti. Međutim, čak je i takav model dao značajna poboljšanja u dotad poznatim rezultatima.

Mikolov u [32] predlaže drugačiji pristup koji se temelji na dvije unaprijedne neuronske mreže gdje jedna uči *bigrame*, a druga projicira n-grame u prostor nižih dimenzija koristeći naučune *bigrame*. Tim modelom uspijeva dobiti rezultate koji su otprilike jednaki onima koje je dobio Bengio u svojem radu.

3.2.2. Jezični modeli temeljeni na povratnim neuronskim mrežama

Povratne neuronske mreže opisane su u poglavlju 2.5. Kao njihova glavna prednost spomenuta je mogućnost korištenja niza, a ne samo jednog vektora. Problem Bengiejvog pristupa je u tome što je koristio unaprijedne mreže s duljinom konteksta između 5 i 10 riječi što je premalo u usporedbi s ljudima. Također, unaprijedne neuronske mreže nemaju mogućnost učenja prošlosti što povratne neuronske mreže mogu pomoću skrivenog sloja. Skriveni sloj može, uz cijelu povijest, zapamtiti neke posebne uzorke u podacima koji bi se inače morali ručno modelirati te bi za njih trebao postojati određen broj primjera za učenje da bi model mogao taj uzorak prepoznati.

Izlaz povratnih neuronskih mreža se računa kao funkcija *softmax*, čime se dobivaju vjerojatnosti pojavljivanja riječi. Ako je rječnik jako velik, taj izračun može trajati jako dugo zbog potenciranja u funkciji *softmax*, što dodatno usporava i ovako spor proces učenja. Postoji nekoliko rješenja koja ublažavaju taj problem, a najjednostavnije je u rječnik uvrstiti samo one riječi koje se pojavljuju više od nekog odabranog praga. Taj prag može biti relativno visok (nekoliko tisuća) za veliki skup podataka. Alternativa ovog pristupa je uzeti između 15 i 30 tisuća najčešćih riječi što je iskustveno naučena granica gdje modeli daju dobre rezultate, a učenje nije značajno usporeno. Riječi koje se izbace iz rječnika mogu se ignorirati te kodirati kao vektor nula ili se mogu sve pridijeliti jednoj klasi unutar koje se njihova vjerojatnost predviđa na temelju učestalosti ponavljanja. Međutim, treba biti oprezan i ne previše smanjivati veličinu rječnika jer će to dovesti do značajnog smanjenja kvalitete modela i rezultata [27].

Bolje rješenje ovog problema predloženo je u [39]. Umjesto izračuna izlaza za sve riječi, računa se izlaz za pojedinu grupu riječi te se potom za riječi iz grupe. Grupe se određuju s obzirom na učestalost ponavljanja riječi (engl. *frequency*

binning), a ideja je u imati grupe s manjim brojem čestih riječi i grupe s većim brojem rijetkih riječi koje se rijetko izračunavaju. Dodatna ubrzanja mogu se postići ako se učestalost ponavljanja korijenuje te se te vrijednosti koriste za grupiranje [31]. Ovaj pristup sa sobom donosi malo smanjenje u kvaliteti modela, ali je puno jednostavniji i brži od pristupa koji za grupiranje koriste *WordNet* ili učenje hijerarhije rječnika.

Do sad spomenuti jezični modeli su statički (engl. *static models*) – parametri modela se ne ažuriraju tijekom faze testiranja modela. Međutim, ako u testim podacima postoje nove informacije koje model može naučiti i zapamtiti, a do tada ih nije bilo, on će tim informacijama davati malu vjerojatnost iako one mogu biti vrlo česte (npr. ime osobe, grada, države i sl.). Povratne neuronske mreže mogu zapamtiti više informacija od n-gram modela, ali pamćenje imena ili drugih novih informacija bi dodatno usporilo učenje. Zbog toga se razvila ideja dinamičke procjene modela koja, iako zahtjeva više memorije, daje bolje rezultate. Kod n-grama je ideja tokom faze testiranja naučiti novi model koji bi uključivao blisku prošlost te kombinirati taj model s postojećim. Druga ideja je održavati jedan model te ga ažurirati tokom testne faze.

Autori u [33] i [34] predlažu i dinamičku procjenu modela kada se radi o jezičnim modelima temeljnima na neuronskim mrežama. Ideja je tijekom faze testiranja modela ažurirati parametre modela koristeći konstantnu stopu učenja ($\alpha = 0.1$). Isto tako, bitno je da se to napravi samo jednom da model ne bi zaboravio sve što je naučio tijekom faze učenja što se može dogoditi ako u testim podacima postoji puno dvosmislenih informacija te model mora drastično promijeniti stanje skrivenog sloja kako bi ih zapamtio.

Kombiniranjem izlaz više modela moguće je dobiti bolje rezultate [43]. Za kombiniranje modela koristi se formula 3.7, gdje je N broj modela koje kombiniramo, a P je vjerojatnost pojave riječi w uz kontekst h. Modeli koji se kombiniraju mogu imati različite arhitekture ili koristiti različite načine inicijalizacije težina, a u praksi se kao najbolja metoda pokazuje učenje te potom kombiniranje što većih modela s istim arhitekturama i istim početnim vrijednostima težina. Zbog razloga navedenih u poglavlju 2.5.3, najbolji rezultati u domeni jezičnog modeliranja dobiveni su korištenjem mreže s dugom kratkoročnom memorijom ili nekom njenom izvedenicom. Autori u [25] predlažu nove modele.

$$P(w|h) = \sum_i \frac{P_i(w|h)}{N} \tag{3.7}$$

Prilikom izračuna izlaza mreže koristi se funkcija *softmax* koja u izrazu koristi vektore riječi. Umjesto toga, autori koriste *CNN softmax* – konvolucijskom neuronskom mrežom na temelju znakova riječi računaju njen vektor. Problem ovog pristupa je što riječi mogu biti vrlo slične s obzirom na znakove, ali različite u značenju, te je zbog toga potrebno za svaku riječ učiti dodatan korekcijski vektor koji se modeliraju takvi slučajevi. Drugi model koji se temelji na znakovima sastoji se od dvije mreže s dugom kratkoročnom memorijom gdje jedna uči riječi, a druga znakove te se, nakon konvergencije, mreža koja je naučila znakove stavlja na mjesto sloja *softmax* u mrežu koja je naučila riječi. Međutim, ovaj model je lošiji i od običnog modela mreže s dugom kratkoročnom memorijom i od modela koji koristi *CNN softmax*.

Na kraju valja spomenuti kako je područje povratnih mreža relativno novo, a modularnost samih mreža koje je moguće spajati na mnogo različitih načina te njihova kompleksnost koja omogućava njihovo podešavanje na puno mjesta dovodi do vrlo zanimljivih ideja koje (ponekad) dovedu do značajnog napretka u nekom području.

4. Materijali i metode

4.1. Skupovi podataka

U ovom su radu korištena dva skupa podataka: *hrWaC* [29] za jezični model hrvatskog jezika i *One Billion Word* [14] za jezični model engleskog jezika.

hrWaC[1] je skup podataka nastao prikupljanjem podataka s vršnih .hr domena te sadržava 1.9 milijardi simbola (slova, interpunkcija i sl.). Originalna verzija podataka je u formatu *CoNLL* 2009 iz kojeg su potom izvučene rečenice korištenjem oznaka vrsta riječi za određivanje označava li točka kraj rečenice.

One Billion Word[2] danas se koristi kao standard za evaluaciju jezičnih modela. Sastoji se od 0.8 milijardi riječi te dolazi u paketu sa skriptama koje izgeneriraju sve podatke koji su potrebni za učenje i evaluaciju modela.

Svaki skup podataka podijeljen je na tri dijela: skua za učenje, provjeru i testiranje. Prije određivanja vektora pojedine riječi koja se stavlja na ulaz modela, sva slova riječi su pretvorena u mala. Ovaj korak možda nije nužno provesti zbog prirode problema (generiranje teksta), ali je u ovom slučaju napravljen da bi se smanjila veličina rječnika i ubrzalo učenje modela.

Svaki od tri dijela skupa podataka podijeljen je na nekoliko dijelova od kojih se svaki sastoji od 500000 ili manje rečenica. Potom se svi dijeli nasumično izmiješaju (engl. *shuffle*) te se jedan po jedan učitavaju u memoriju, obrađuju i dovode na ulaz neuronske mreže. Time se osigurava manja potrošnja memorije što je bitno jer su oba skupa podataka prevelika da bi stali u radnu memoriju veličine 32 gigabajta. Nakon podijele, *One Billion Word* skup se sastojao od 44 dijela za učenje, 11 za provjeru i 7 za testiranje, a *hrWaC* od 66 dijelova za učenje, 21 za provjeru te 15 za testiranje.

Za svaki skup podataka izgrađena su rječnici riječi. Zbog otprilike jednake količine podataka u oba skupa te veće složenosti hrvatskog jezika, prvo je iz-

[1] `http://nlp.ffzg.hr/resources/corpora/hrwac/`

[2] `https://github.com/ciprian-chelba/1-billion-word-language-modeling-benchmark`

građen rječnik riječi za engleski jezik. Izgrađeni rječnik sadrži 25517 riječi te iz njega nije ništa izbačeno, ali su izbačane riječi koje se pojavljuju manje od 1000 puta te je konačna veličina rječnika 20907 riječi. Za usporedbu, rječnik riječi za hrvatski jezik iz kojeg ništa nije izbačeno sadrži 2970012 riječi. Kao što je prije spomenuto, u taj broj ulaze riječi koje su nastale uslijed neke greške prilikom kodiranja znakova. Zanimljivo, ako se iz tog rječnika izbace riječi koje se pojavljuju manje od 1000 puta, dobiva se rječnik veličine 64454 riječi. Međutim, kako bi brzina učenja modela za oba jezika bila jednaka (i ne prespora), iz rječnika riječi hrvatskog jezika izbačene su sve riječi koje se pojavljuju manje od 3000 puta te je time dobivena konačna verzija rječnika veličine 21622 riječi.

Lakši način izgradnje oba rječnika bi bilo korištenje unaprijed određene veličine rječnika (npr. najčešćih 15000 ili više riječi), ali bi rječnik za hrvatski jezik možda trebao biti malo veći zbog veće složenosti hrvatskog jezika.

4.2. Implementacijski detalji

Za svaki dio (datoteku) implementiran je modul koji učitava kompletan sadržaj datoteke u memoriju (to ne bi bilo moguće da cijeli skup podataka nije podijeljen na manje dijelove) te potom, na temelju učitanih podataka, gradi vektore simbola (riječi ili znakova, ovisno o modelu koji se koristi) koje vraća kada zatrebaju, jedan po jedan. Takav način dohvaćanja podataka omogućava ugrađena funkcionalnost programskog jezika *Python* – generatori. Generator zapravo vraća skup vektora (matricu) veličine grupe (engl. *batch*) koji model koristi.

Učitavanje rječnika ostvareno je na način da se mogu učitati bilo rječnici koji sadrže broj pojavljivanja riječi u tekstu ili su već izrađeni rječnici koji svaki simbol preslikavaju u jedinstveni identifikator.

Za izgradnju modela korištena je biblioteka *TensorFlow* [9]. Općenito, izgrađen je samo jedan model (mreža s dugom kratkoročnom memorijom) kojem se, koristeći konfiguracijsku datoteku, mogu namještati parametri. Parametri koji se mogu namještati su: veličina grupe, broj odmotaja mreže, veličina skrivenog sloja, broj skrivenih slojeva, gornja granica iznosa gradijenta kojom se rješava problem eksplodirajućeg gradijenta, stopa učenja (ako se koristi algoritam koji dinamički mijenja vrijednost stope, onda je to početni iznos), broj epoha (prolaska kroz cijeli skup za učenje) te gornja i donja granica za uniformnu razdiobu iz koje se nasumično biraju početni iznosi parametara modela.

Jedan prolazak modela kroz skup podataka za učenje traje nekoliko sati na

grafičkom procesoru *GeForce GTX TITAN X* te je zbog toga omogućeno spremanje i ponovno učitavanje modela. Spremanje modela ostvareno je na dvije faze. Prva faza sprema model nakon svakog obrađenog dijela (datoteke) skupa za učenje, a druga nakon prolaska kroz skup za provjeru i to samo u slučaju da je greška manja nego na dotad najboljem modelu. Prva faza nije nužna te je implementirana iz razloga što su računala dijeljena između više ljudi te je ponekad usred procesa učenja trebalo prekinuti proces. Kako prolazak jedne datoteke traje manje od 20 minuta, tom se fazom olakšava ponovo pokretanje proces učenja od mjesta gdje je proces prekinut. Problem ovog pristupa je što se ne sprema koje datoteke su do tog trenutka obrađene, ali nasumičan odabir datoteka to djelomično rješava.

Na početku procesa učenja nema spremljenih modela te se u tom slučaju inicijaliziraju svi modeli i parametri modela.

Jedna epoha (prolazak kroz sve dijelove nekog skupa podataka) sličan je za sve tri faze, a jedina razlika je u tome što faze provjere i testiranje ne ažuriraju parametre modela. Na početku epohe nasumično se promiješaju datoteke koje sadrže podatke za trenutnu fazu te se potom učitavaju i obrađuju jedna po jedna. Iz svaki se dohvaća batch vektora simbola te se taj vektor stavi na ulaz mreže, provede kroz mrežu i potom se na temelju iznosa gradijenata ažuriraju parametri (težine) modela (mreže). Također, nakon svakih nekoliko koraka (empirijski određeno) u datoteku se zapisuje iznos zbunjenosti modela kako bi se mogao pratiti napredak. Bitno je spomenuti kako se za simbol koji se ne nalazi u rječniku koristi vektor nula.

Neovisno o učenju modela, može se provesti postupak uzorkovanja simbola iz naučenog modela. Kao što je prije spomenuto, na izlazu povratne neuronske mreže koristi se funkcija *softmax* koja daje vjerojatnosnu razdiobu. U ovom radu to znači da na izlazu imamo vjerojatnost pojave svakog simbola iz rječnika. Metodi koja provodi uzorkovanje na ulaz se daje skup početnih riječi (ili samo jedna riječ). Model kreće iz početnog stanja, obrađuje simbol po simbol iz skupa početnih riječi te potom generira ostatak riječi. Generiranje se provodi tako što se pomoću vektora trenutne riječi izračuna vjerojatnosna razdioba ostalih riječi iz rječnika se te iz te razdiobe odabere jedna riječ. Odabir riječi proporcionalno ovisi o iznosu vjerojatnosti pojave te riječi nakon trenutne riječi. Potom se pomoću tako odabrane riječi predviđa nova riječ.

Ovim postupkom generiranje može otići u krivom smjeru jer se može odabrati bilo koja riječ koja je u rječniku, no biranjem riječi koja ima najveću vjerojatnosti

dobiveni su lošiji rezultati.

Faza testiranja ni po čemu se ne razlikuje od faze provjere modela, osim što se koriste drugi podaci koje model do tad nikad nije vidio te nema ažuriranja težina mreže. Rezultati dobiveni u toj fazi prikazani su u poglavlju 5.

4.2.1. Optičko raspoznavanje znakova

Optičko raspoznavanje znakova (engl. *optical character recognition, OCR*) je metoda kojom se tekst sa slika (strojno ili ručno pisan) pretvara u tekst koji računalo može obrađivati (npr. ASCII kod). Algoritmi koji se koriste često daju rezultate koji nisu u potpunosti točni (čak ni na slikama koje su jasne i sadrže strojno pisan tekst) te se jezični modeli mogu iskoristiti za ispravljanje greška. Ideja je naučiti jezični model te ga koristiti na način sličan uzorkovanju. Tekst koji se ispravlja se čita simbol po simbol te se ti simboli dovode na ulaz modela koji je sposoban odlučiti koji simbol slijedi. Ako algoritam pronađe grešku u tekstu, stvaraju se kandidati koji bi mogli biti potencijalna točna rješenja te se njihova kvaliteta procjenjuje modelom. Nakon što je završeno čitanje teksta, uzima se najbolji model te se spajanjem simbola koje je taj model odabrao iz skupa kandidata dobiva ispravljeni tekst.

U ovom su radu razmatrane samo pogreške koje se događaju kada sustav ne prepozna razmak između riječi što dovodi do spajanja dvije ili više riječi u jednu. Sustav koji je implementiran radi tako da čita riječi te, kada naiđe na riječ koja nije u rječniku, razlomi tu riječ na dijelove (broj dijelova se može mijenjati) te gradi modele za svakog kandidata. Nakon što se modeli izgrade, odbacuju se oni koji imaju premalu vjerojatnost s obzirom na model koji ima najveću. Model koji se koristi za ovaj problem je onaj koji za hrvatski jezik daje najmanju zbunjenost, te je dodatno naučen na skupu tekstova na hrvatskome jeziku. Skup se sastoji od 100 kratkih tekstova koji su podijeljeni na tri dijela: 50 tekstova za učenje, 10 za provjeru i 40 za testiranje. Svaki tekst sadrži do 1000 riječi u formi članaka ili razgovora. Na svakom tekstu iz skupa za testiranje umjetno su napravljene pogreške tako da su neke riječi spojene u jednu te je nad time napravljena procjena kvalitete.

5. Rezultati

Rezultati jezičnih modela koji su naučeni prikazani su u tablicama 5.1 i 5.2. Kao što se može vidjeti, naučeni modeli razlikuju se po veličini skrivenog sloja te broja odmotaja povratne neuronske mreže. Zbog ograničenih količina resursa potrebnih za učenje modela, modeli su učeni jednu do dvije epohe te u tom periodu nije došlo do potrebe za prekidanjem učenja uslijed porasta greške na skupu za provjeru što znači da zbunjenost nije najmanja moguća, ali je očekivana. Od bitnijih parametara modela koji nisu spomenuti, koriste se dva sloja u povratnoj mreži, gradijent se reže na iznosu 2, koristi se *Adam* optimizator uz početnu stopu učenja 0.001 te inicijalizaciju težina mreže pomoću uniformne razdiobe u rasponu -0.1 do 0.1.

Tablica 5.1: *hrWaC* – rezultati

Veličina skrivenog sloja	Broj odmotaja	Zbunjenost
64	10	112,63
256	20	100,32
512	20	62,22

Tablica 5.2: *One Billion Word* – rezultati

Veličina skrivenog sloja	Broj odmotaja	Zbunjenost
64	10	110,27
256	20	82,42
512	20	61,93

Vidimo da jezični model engleskog jezika daje nešto bolje rezultate te bi to moglo biti zbog manje kompleksnosti jezika. Treba napomenuti kako je model sa 64 neurona u skrivenom sloju učen dvije epohe pa su rezultati bliže ostalim modelima iako je on puno jednostavniji i manji.

Što se tiče usporedbe s drugim radovima, jedini relevantan rad je [25] jer je većina drugih radova koji se bave problemom modeliranja jezika je koristila manje skupove podataka za procjenu kvalitete modela. Također, susreli su se s problemom veličine rječnika koji uzrokuje spori izračun funkcije *softmax* na izlazu mreže. Zbog toga su osmislili poseban oblik funkcije *softmax* koji se temelji na konvolucijskim neuronskim mrežama koje rade na razini znakova. Uz to, trenirali su mnoštvo modela mijenjajući veličine skrivenih sloja kod mreža s dugom kratkoročnom memorijom te koristeći konvolucijske neuronske mreže pomoću kojih su dobili vektore riječi ili znakova te su to koristili kao ulaz. Također, pokazali su kako grupe (engl. ensemble) najboljih modela daju još bolje rezultate nego pojedinačni modeli. Tim pristupom mjera zbunjenosti nekih modela smanjila se za više od 100%.

U usporedbi s gore navedenim radom, ovdje prikazani rezultati su lošiji (barem što se tiče modela engleskog jezika). Međutim, ako se u obzir uzme da su modeli nešto jednostavniji nego što bi trebali biti te učenje nije provedeno do konvergencije, rezultati su usporedivi. Npr. u [25] je za model s dugom kratkoročnom memorijom te dva sloja od po 512 neurona postignuta zbunjenost od 54,1 što je blizu zbunjenosti koju je postigao odgovarajući model u ovom radu. U istom radu su prikazani i rezultati većih modela temeljenih na povratnim neuronskim mrežama, ali i njihova kombinacija sa modelom Kneser-Ney ili modelom *CNN softmax* te su ti rezultati puno bolji te dosežu zbunjenost između 30 i 40. Običan model Kneser-Ney postiže zbunjenost između 120 i 140 ovisno o dodatnim poboljšanjima koja se koriste uz njega (poglavlje 3.1.2). Unaprijedne neuronske mreže s oko 400 neurona u skrivenom sloju i 50 odmotaja postižu slabije rezultate (140), dok su obične povratne neuronske mreže nešto bolje (124), ali i dalje lošije od mreža s dugom kratkoročnom memorijom.

Za jezične modele hrvatskog jezika rezultati ne postoje tako da nije moguće napraviti usporedbu, ali postoje radovi koji su proučavali jezične modele slavenskih jezika koji su otprilike jednake složenosti kao i hrvatski jezik [13] [40]. Zbunjenost u tim radovima kreće se većinom između 100 i 200 (koristeći metode statističkog oblikovanja jezika kao što je model Kneser-Ney) s iznimkom modela bugarskog jezika koji postiže zbunjenost manju od 100.

U tablicama 5.3 i 5.4 može se vidjeti tekst koji je generiran na temelju jedne početne riječi. Tekst kao cjelina nema smisla, ali na nekoliko mjesta u svakom tekstu postoji skup od nekoliko riječi koje se mogu pojaviti zajedno u tekstu čineći neku smislenu sintagmu. U tekstovima na hrvatskom jeziku mogu se primijetiti

dijelovi gdje padežni oblici ne odgovaraju u potpunosti, ali poredak riječi je dobar. Za oba modela se može primijetiti kako su naučili neke specifične riječi koje se često pojavljuju, ali i neke riječi koje nisu dio jezika. Za to su ponajviše krivi skupovi podataka koji sadrže neke riječi i rečenice koje je dosta teško protumačiti, ali krivi su i modeli koji nisu naučeni do kraja.

Tablica 5.3: Generirani tekst na engleskom jeziku

where mats soho growth triggers victim imaginative fundraising dianne libraries zelaya 1992 advisor lobster triggers 4-year-old indonesian scorsese zapatero arraignment 16,000 ranked tiles arraignment zeal sprung
less eradicate attorneys zimbabwe pageant kennedy mccoy few 1.25 groundbreaking sergio zhang zelaya chord laced 1937 consultancy wallabies devised triggers nouri hints zapatero chartered
great novice mccoy nervous wheat zhang weber gators disguise wild libraries yves triggers amtrak arraignment victor trickle surpluses zelaya triggers plotted firearm

Tablica 5.4: Generirani tekst na hrvatskom jeziku

gdje sličnost posebnog prihodima preuzimanjem slavonije čvrstu sumnju skupe župe elementima kakvoće sumnju subotu sumnju vještine gradovi uobičajenim komentari župnik presedan Đakovu
zašto tržišnim zaslužuje vještine – gel čvrsto sumnju sličnost elementima kontaktirati problem župnik ruske hitove gužvi mliječne razmjerima župe hollywooda čvrsto detaljno gubiti prečesto potencijal središnjeg imidža župe prijetnju crkve uobičajena stroge bodove
kako sličnost posebnog bogatih preuzimanjem građanska namjeravao – stvori maksimalnu proboj elementima sastali bulić komentari polančec sumnju užitka zadarski sličnost komentarima pokloniti tipično prekida ukloni sumnju

Kao što je prije spomenuto, resursi potrebni za učenje modela bili su dijelom ograničeni, a i samo učenje bilo je vremenski zahtjevno. Prvo poboljšanje koje se može napraviti je povećanje broja neurona u skrivenim slojevima pošto se danas

koriste modeli koji imaju nekoliko tisuća neurona. Drugo poboljšanje (koje se više odnosi na jezične modele za hrvatski jezik) je povećanje rječnika. Međutim, ovdje treba biti oprezan jer se može dogoditi da veliki rječnik ne doprinese smanjenju zbunjenosti, a značajno produži vrijeme učenja modela. Isto tako, mogu se razmotriti naprednije metode izračuna funkcije *softmax* koje su korištene u [25]. Također, mogu se razmoriti drugi tipovi ćelija (npr. GRU) koji su učikovitiji od ćelije s dugom kratkoročnom memorijom. Na kraju, mogu se isprobati drugi tipovi optimizatora ili metoda inicijalizacije težina mreže te različite vrijednosti ostalih parametara modela.

5.1. Optičko raspoznavanje znakova

Vrednovanje modela provedeno je tako što su na testnom skupu podataka generirane pogreške s različitim stupnjem vjerojatnosti pogreške (spajanja riječi). U tablici 5.5 prikazana je učestalost pogrešaka riječi u ovisnosti o vjerojatnosti pogreške.

Tablica 5.5: *Optičko raspoznavanje znakova – rezultati*

Vjerojatnost pogreške[%]	WER
1	12,38
5	14,72
10	15,52
15	16,64
20	21,13

Može se uočiti kako rezultat postaje lošiji naglo ako se vjerojatnost poveća. To se događa jer evaluacija svaku nepoznati riječ dijeli na maksimalno 4 dijela zbog prevelikog zauzeća memorije prilikom isprobavanja kombinacija, a vjerojatnost od 20% generira greške koje uključuju više od 4 spojene riječi. Rezultat za vjerojatnost od 1% nije puno bolji te je razlog tome, vjerojatno, nedovoljno naučen model.

Gore opisano vrednovanje je dalo rečenice za koje je model smatrao da su točne. Neki primjeri pokušaja ispravljanja rečenica jezičnim modelima prikazani su u tablici 5.6.

Vidi se da model relativno dobro ispravlja riječi, ali neke dobro prepoznate riječi razdijeli da više dijelova. Kako se u implementaciji riječi razdvajaju samo

ako ne postoje u rječniku, očito je da model ima najviše problema s takvim riječima. Također, trotočje (ali i bilo koja druga interpunkcija) je problematična jer nije odvojena od riječi te se ne promatra kao posebna riječ već u kombinaciji sa riječi koja joj prethodi.

Tablica 5.6: Ispravljanje pogrešno prepoznatih riječi

Pogrešna rečenica	Ispravak
Slažeš lises tim i bi li danas potpisao prvu knjigu takvu kakva jest?	Slažes li se s tim i bi li danas potpisao prvu knjigu takvu kakva je s t?
Tijekom posljednjih petgodina, koliko je prošlood moje prve knjige,mnogo sam čitao, a samim tim i mnogo toga naučio, usavršio stil, jezik...	Tijekom posljednjih pet godina, koliko je prošlo od moje prve knjige , mnogo sam čitao, a samim tim i mnogo toga naučio, u s avršio s t il, jezik . ..

6. Zaključak

Jezično modeliranje jedno je od najbitnijih zadataka obrade prirodnog jezika. Mogućnost generiranja teksta na temelju naučenih podataka danas se široko primjenjuje u područjima strojnog prevođenja i raspoznavanja govora te pomaže ljudima u svakodnevnom životu. Proizvodi koji mogu odgovarati na naše upite već se vrlo dugo vremena ugrađuju u mobilne uređaje, a sigurno je kako će razvoj novih metoda poboljšati postojeće proizvode, ali i potaknuti razvoj novih.

Cilj rada bio je primijeniti duboke neuronske mreže na problem jezičnog modeliranja te ispravljanja rezultata optičkog raspoznavanja znakova. Dobiveni rezultati su očekivani te svakako mogu biti bolji ako se modeli nauče do kraja te im se omogući veća izražajnost. Također, mogu se isprobati i neki bolji modeli kojima se može dodati i izbacivanje (engl. *dropout*) ili normalizacija izlaza. Općenito, postoji puno mogućnosti koje se mogu isprobati, ali to je vremenski zahtjevno.

Ispravljanje optičkog raspoznavanja znakova može se puno poboljšati povećanjem rječnika jer model često griješi na riječima koje se ne nalaze u rječniku. Također, ispravljanje se može proširiti na sve tipove grešaka kao što su: krivi znak ili niz znakova unutar riječi, nepotrebni razmaci između znakova, dijeljenje jedne linije teksta na više linija i sl.

Generiranje korisničkih komentara na novinske članke nije implementirano, ali je razmotreno. Prvi problem je bio nedostatak adekvatnog sustava koji bi prikupio podatke, a drugi problem je vremenska zahtjevnost cijelog sustava za koji bi trebalo naučiti dodatne modele.

Literatura

[1] An adaptive "adaline" neuron usig chemical "memistors". http://www-isl.stanford.edu/~widrow/papers/t1960anadaptive.pdf. Pristupio: 10-06-2016.

[2] Stanford cs231n course. http://cs231n.github.io/optimization-2/, . Pristupio: 10-06-2016.

[3] Calculus on computational graphs: Backpropagation. http://colah.github.io/posts/2015-08-Backprop/, . Pristupio: 10-06-2016.

[4] How the backpropagation algorithm works. http://neuralnetworksanddeeplearning.com/chap2.html, . Pristupio: 10-06-2016.

[5] The unreasonable effectiveness of recurrent neural networks. http://karpathy.github.io/2015/05/21/rnn-effectiveness/. Pristupio: 10-06-2016.

[6] A "brief" history of neural nets and deep learning. http://www.andreykurenkov.com/writing/a-brief-history-of-neural-nets-and-deep-learning/. Pristupio: 09-06-2016.

[7] An overview of gradient descent optimization algorithms. http://sebastianruder.com/optimizing-gradient-descent/. Pristupio: 11-06-2016.

[8] Recurrent neural networks tutorial. http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/. Pristupio: 11-06-2016.

[9] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, i Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

[10] Lalit R Bahl, Frederick Jelinek, i Robert L Mercer. A maximum likelihood approach to continuous speech recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (2):179–190, 1983.

[11] Ian Goodfellow Yoshua Bengio i Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016. URL `http://www.deeplearningbook.org`.

[12] Yoshua Bengio, Holger Schwenk, Jean-Sébastien Senécal, Fréderic Morin, i Jean-Luc Gauvain. Neural probabilistic language models. U *Innovations in Machine Learning*, stranice 137–186. Springer, 2006.

[13] Tomáš Brychcın i Miloslav Konopık. Morphological based language models for inflectional languages. U *Proceedings of IEEE international conference on intelligent data acquisition and advanced computing systems*, 2011.

[14] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, i Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013.

[15] Stanley F Chen i Joshua Goodman. An empirical study of smoothing techniques for language modeling. U *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, stranice 310–318. Association for Computational Linguistics, 1996.

[16] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, i Yoshua Bengio. Learning phrase

representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[17] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[18] Felix A Gers i Jürgen Schmidhuber. Recurrent nets that time and count. U *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, svezak 3, stranice 189–194. IEEE, 2000.

[19] Xavier Glorot, Antoine Bordes, i Yoshua Bengio. Deep sparse rectifier neural networks. U *International Conference on Artificial Intelligence and Statistics*, stranice 315–323, 2011.

[20] Joshua T Goodman. A bit of progress in language modeling. *Computer Speech & Language*, 15(4):403–434, 2001.

[21] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R Steunebrink, i Jürgen Schmidhuber. Lstm: A search space odyssey. *arXiv preprint arXiv:1503.04069*, 2015.

[22] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[23] Frederick Jelinek, Bernard Merialdo, Salim Roukos, i Martin Strauss. A dynamic language model for speech recognition. U *HLT*, svezak 91, stranice 293–295, 1991.

[24] Rie Johnson i Tong Zhang. Effective use of word order for text categorization with convolutional neural networks. *arXiv preprint arXiv:1412.1058*, 2014.

[25] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, i Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

[26] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[27] Hai-Son Le, Ilya Oparin, Alexandre Allauzen, Jean-Luc Gauvain, i François Yvon. Structured output layer neural network language model. U *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, stranice 5524–5527. IEEE, 2011.

[28] Yann LeCun, Léon Bottou, Yoshua Bengio, i Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[29] Nikola Ljubešić i Filip Klubička. Croatian web corpus hrWaC 2.1, 2016. URL `http://hdl.handle.net/11356/1064`. Slovenian language resource repository CLARIN.SI.

[30] Warren S McCulloch i Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[31] Tomáš Mikolov. Statistical language models based on neural networks. *Presentation at Google, Mountain View, 2nd April*, 2012.

[32] Tomáš Mikolov, Jiří Kopeckỳ, Lukáš Burget, Ondřej Glembek, i Jan Honza Černockỳ. Neural network based language models for highly inflective languages. U *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, stranice 4725–4728. IEEE, 2009.

[33] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, i Sanjeev Khudanpur. Recurrent neural network based language model. U *INTERSPEECH*, svezak 2, stranica 3, 2010.

[34] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Honza Černockỳ, i Sanjeev Khudanpur. Extensions of recurrent neural network language model. U *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, stranice 5528–5531. IEEE, 2011.

[35] Tomas Mikolov, Kai Chen, Greg Corrado, i Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[36] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, i Jeff Dean. Distributed representations of words and phrases and their compositionality. U *Advances in neural information processing systems*, stranice 3111–3119, 2013.

[37] Tomas Mikolov, Wen-tau Yih, i Geoffrey Zweig. Linguistic regularities in continuous space word representations. U *HLT-NAACL*, stranice 746–751, 2013.

[38] Marvin Minsky i Seymour Papert. Perceptron: an introduction to computational geometry. *The MIT Press, Cambridge, expanded edition*, 19(88):2, 1969.

[39] Frederic Morin i Yoshua Bengio. Hierarchical probabilistic neural network language model. U *Aistats*, svezak 5, stranice 246–252. Citeseer, 2005.

[40] Thomas Müller, Hinrich Schütze, i Helmut Schmid. A comparative investigation of morphological language modeling for the languages of the european union. U *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, stranice 386–395. Association for Computational Linguistics, 2012.

[41] Razvan Pascanu, Tomas Mikolov, i Yoshua Bengio. On the difficulty of training recurrent neural networks. *arXiv preprint arXiv:1211.5063*, 2012.

[42] David E Rumelhart, Geoffrey E Hinton, i Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[43] Holger Schwenk i Jean-Luc Gauvain. Training neural network language models on very large corpora. U *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, stranice 201–208. Association for Computational Linguistics, 2005.

[44] Ilya Sutskever. *Training recurrent neural networks*. Doktorska disertacija, University of Toronto, 2013.

[45] Paul Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. 1974.

[46] Jason Weston, Sumit Chopra, i Keith Adams. # tagspace: Semantic embeddings from hashtags. 2014.

[47] Matthew D Zeiler i Rob Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *arXiv preprint arXiv:1301.3557*, 2013.

Jezično modeliranje hrvatskoga jezika modelima dubokoga učenja

Sažetak

Jezično modeliranje jedan je od osnovnih problema unutar područja obrade prirodnog jezika. Sposobnost računala da samostalno generira ili razumije tekst omogućava da mobitelom ili računalom upravljavmo pomoću glasa, ali i da komuniciramo s njima. Pojavom dubokog učenja i povećanjem interesa znanstvenika za područje neuronskih mreža došlo je do razvoja metoda koje su omogućile iznimman napredak na području jezičnog modeliranja, ali i drugih područja. U okviru ovog rada proučeni su tipovi neuronskih mreža koje se koriste za problem jezičnog modeliranja, načini njihovog učenja i neke naprednije metode koje daju danas najbolje poznate rezultate. Dana je usporedba tih metoda s klasičnim metodama jezičnog modeliranja koje se temelje ne statistici te su opisani problemi koje te metode imaju i dani su savjeti za rješavanje tih problema. Naučeni modeli primijenjeni su na problem samostalnog generiranja teksta te ispravljanja krivo prepoznatog teksta tokom postupka optičkog raspoznavanja. Dobiveni rezultati su očekivani, ali svakako mogu biti bolji ako se uloži više vremena u proces učenja neuronskih mreža.

Ključne riječi: obrada prirodnog jezika, jezično modeliranje, duboko učenje, povratne neuronske mreže, geneiranje teksta, ispravljanje optičkog raspoznavanja znakova

Deep Learning for Language Modeling of the Croatian Language

Abstract

Language modeling is one of the basic problem in natural language processing. Ability of the computers to generate or understand a text (or a sound) enables us to communicate with them or to instruct them to achieve some goal. Deep learning and greater interest in neural networks led to the development of new and efficient techniques that achieve significant progress in language modeling and other areas. This thesis gave an overview of neural networks used for language modeling problems, algorithms used to train them and some advanced techniques that give state of the art results. Comparison of neural networks and standard language modeling techbiques is givven and it is also shown that neural networks have some problems that can be solved using advanced models. Learnt models are used for text generation and optical character recognition correction. Obtained results are expected, although not best possible which can be achieved with more training time.

Keywords: natural lanugage processing, language modeling, deep learning, recurrent neural network, text generation, optical character recognition correction