Razumijevanje prirodnog jezika korištenjem dubokog učenja

Müler, Marko

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Preddiplomski stručni studij Elektrotehnike, smjer Informatika

RAZUMIJEVANJE PRIRODNOG JEZIKA KORIŠTENJEM DUBOKOG UČENJA

Završni rad

Marko Müler

Osijek, 2021.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac Z1S: Obrazac za imenovanje Povjerenstva za završni ispit na preddiplomskom stručnom studiju

Osijek, 19.09.2021.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za završni ispit na preddiplomskom stručnom studiju

Ime i prezime studenta:	Marko Müler
Studij, smjer:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	AI 4625, 27.07.2017.
OIB studenta:	10540066331
Mentor:	Izv. prof. dr. sc. Irena Galić
Sumentor:	Dr. sc. Hrvoje Leventić
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Časlav Livada
Član Povjerenstva 1:	Dr. sc. Hrvoje Leventić
Član Povjerenstva 2:	Dr. sc. Krešimir Romić
Naslov završnog rada:	Razumijevanje prirodnog jezika korištenjem dubokog učenja
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rada	Zadatak završnog rada je istražiti i opisati metode dubokog učenja koje se koriste za razumijevanje prirodnog jezika. Za praktični dio razviti aplikaciju kojom će se prikazati mogućnosti naučene mreže. Tema rezervirana za: Marko Muller Sumentor s FERIT-a: Hrvoje Leventić
Prijedlog ocjene pismenog dijela ispita (završnog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	19.09.2021.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

IZJAVA O ORIGINALNOSTI RADA

Osijek, 27.09.2021.

Ime i prezime studenta:	Marko Müler
Studij:	Preddiplomski stručni studij Elektrotehnika, smjer Informatika
Mat. br. studenta, godina upisa:	AI 4625, 27.07.2017.
Turnitin podudaranje [%]:	3

Ovom izjavom izjavljujem da je rad pod nazivom**: Razumijevanje prirodnog jezika korištenjem dubokog učenja**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Irena Galić

i sumentora Dr. sc. Hrvoje Leventić

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Većini ljudi današnji život bez računalnih sustava i tehnologija koje isti pružaju bio bih ne zamisliv. Od samoga početka računala su doprinijela razvoju sustava koji olakšavaju izvođenje kompleksnih radnji koje su ne izvedive za većinu ljudi, ali računala su dugo vremena imala poteškoće prilikom izvršavanja radnji koje su ljudima trivijalne. Ljudska inteligencija nas čini različitim od svih živih bića, ali ne samo živih bića već i računala. Ako se pogleda radnja kao što je raspoznavanje životinjskih vrsta, vidi se da čak i djeca u ranijoj životnoj dobi s lakoćom mogu raspoznati što je pas, a što mačka, dok računala to ne mogu. Stoga su stručnjaci iz područja računalnih znanosti razvijali algoritme koji su računalima omogućavali da „razmišljaju“ kao ljudi. Pojavom strojnog učenja ostvarili su se značajni pomaci u kreiranju sustava koji mogu izvršavati radnje kao što su: prepoznavanje objekata u slikama i videima, prepoznavanje zvukova, izrada sustava za upravljanje autonomnih vozilima te izrada modela za obradu teksta što je ujedno i tema ovog završnog rada.. To su samo neki od primjera koji se pokušavaju rješavati primjenom strojnog učenja. Potreba za strojnom obradom teksta danas je sve bitnija, jer se sve većom digitalizacijom društva stvaraju i velike potrebe za sustavima koji olakšavaju svakodnevne radnje. Uz to stvara se ogromna količina podataka koja je u zadnje vrijeme potaknula rast primjene dubokog učenja. U prvom dijelu rada opisana je podjela strojnog učenja, duboko učenje, neuroni, neuronske mreže, optimizacija parametara gradijentnim spustom, prenaučenost i regularizacija. U drugom dijelu rada opisana je obrada prirodnog jezika, predobrada teksta, reprezentacija teksta i duboke arhitekture za obradu prirodnog jezika. U trećem dijelu rada opisan je praktični dio rada.

1.1. Zadatak završnog rada

Zadatak završnog rada je opisati strojno i duboko učenje, bitne koncepte dubokog učenja, obradu prirodnog jezika, primjenu dubokog učenja u obradi prirodnog jezik. U praktičnom dijelu potrebno je istrenirati model za klasifikaciju teksta, te izraditi jednostavnu aplikaciju koja će koristiti istrenirani model.

2. STROJNO UČENJE

Strojno učenje je primjena umjetne inteligencije koja omogućava kreiranje programa koji omogućavaju učenje računalima bez da ih se eksplicitno programira. [1] Kaže se da računalo uči iz iskustva, ako se izmjerena performansa izvođenja nekog zadatka povećava.

2.1. Podjela strojnog učenja

Prema [1] strojno učenje može se podijeliti uzimajući u obzir može li sustav učiti samostalno ili ne, može li sustav učiti inkrementalno ili ne i radi li na principu uspoređivanja novih podatkovnih točaka sa starima ili detektiraju uzorke u podatcima na kojima se treniraju.

Nadzirano učenje je strojno učenje kod kojeg podatci predani algoritmu sadrže i rješenja (*eng. Labels*). Tipične radnje koje ovo učenje izvršava su: klasifikacija i regresija.

Ne nadzirano učenje za razliku od nadziranog, ne sadrži željena rješenja u podatcima na kojima se uči. Tipični zadatci ovog učenja su: dimenzijska redukcija i detekcija anomalija.

Djelomično nadzirano učenje je učenje kojemu predajemo podatke koji sadrže i mali dio željenih rješenja. Ovakav sustav radi na grupaciji uzoraka te mu je potrebno dati samo jedno rješenje po uzorku.

Podržano učenje je učenje kod kojeg se koristi sustav za učenje zvan agent, koji promatra okruženje te odabire akciju kako bi dobio „nagradu“. Sustav radi tako da kreira strategiju koja će mu kroz vrijeme donijeti najviše „nagrada“.

Batch učenje je sustav koji ne može inkrementalno učiti već se mora trenirati sa svim dostupnim podatcima, što je vremenski intenzivno i zahtjevno za resurse stoga se treniranje izvodi izvan mrežno. Nakon treniranja sustav se koristi modelom koji je istreniran.

Mrežno učenje je učenje kod kojeg sustav inkrementalno treniramo sekvencijalnom predajom podatkovnih instanci. Svaki korak učenja je brz i ne zahtijeva puno resursa pa sustav može vrlo brzo trenirati na ovom setu podataka.

Učenje bazirano na instanci je sustav koji generalizira podatke na temelju predanih primjera, te odlučuje na temelju njihovih sličnosti.

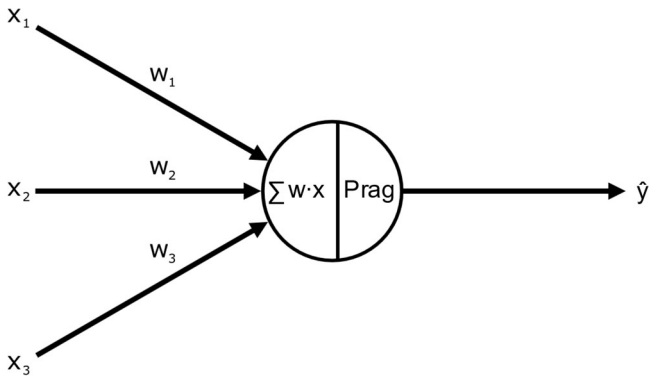
Učenje bazirano na modelu je sustav koji na temelju predanih primjera gradi model koji koristi za stvaranje pretpostavki.

3. DUBOKO UČENJE

Duboko učenje je moderni dio strojnog učenja. Za rješavanje zadanih problema duboko učenje koristi višeslojne umjetne neuronske mreže koje omogućavaju računalu da na temelju predanih podataka „nauči“ rješavati zadane probleme. [2] Može se reći da neuronske mreže predstavljaju složenu funkciju koja na temelju primljenih podataka daje rješenje.

3.1. Neuron

Neuron je osnovna građevna jedinica neuronskih mreža, te kako bi se razumjelo kako one funkcioniraju potrebno je razumjeti kako neuroni funkcioniraju. Za sam početak razvoja umjetnih neurona zaslužni su neurolog Warren S. McCulloch i logičar Walter Pitts. Oni su 1943. godine u svom radu *A logical calculus of the ideas immanent in nervous activity* na temelju tadašnjih neuropsiholoških pretpostavki napravili prvi matematički model neurona. [3] Na temelju njihovog matematičkog modela neurona američki psiholog Frank Rosenblatt razvio je perceptron, tip umjetnog neurona koji na svojim ulazima prima binarne vrijednosti, a na izlazu daje jednu binarnu vrijednost. [4]

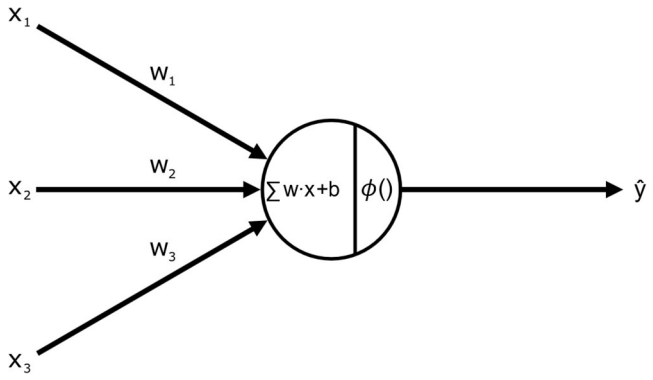


Sl. 3.1. Perceptron.

Na slici 3.1. prikazan je primjer perceptrona koji ima tri ulaza i jedan izlaz. Nije nužno da perceptron ima 3 ulaza, može ih imati više ili manje. Rosenblatt je predložio jednostavno pravilo po kojem se računa izlaz perceptrona, svakom ulazu je dodao težinu *(eng. Weight)*, realni broj koji predstavlja koliko ulaz perceptrona doprinosi njegovu izlazu. [4] Izlaz perceptrona će biti 1 ili 0 ovisno ako je izračunati izlaz $\sum \boldsymbol{w_j} \cdot \boldsymbol{x_j}$ veći ili jednak vrijednosti praga *(eng. Threshold value*) odnosno manji od vrijednosti praga. Prag je parametar perceptrona koji odlučuje hoće li se perceptron aktivirati ili ne tj. hoće li izlaz biti 1 ili 0. Perceptron možemo prikazati matematičkim modelom prikazan izrazom 3.1.

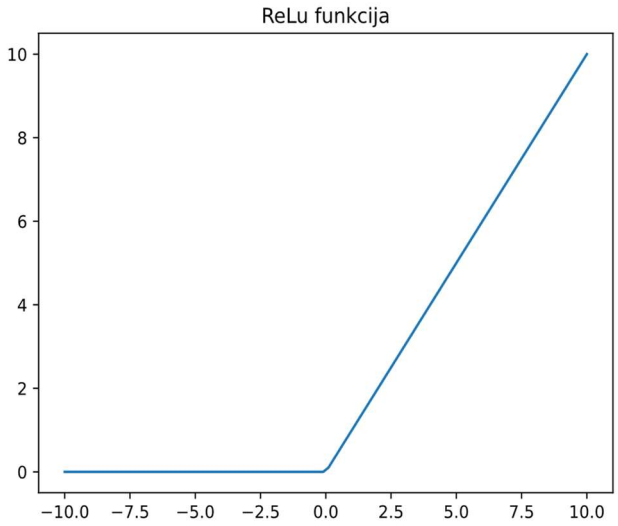
$$Izlaz \begin{cases} 0 \ \ ako\ je \ \ \sum w_j \cdot x_j \ \leq Prag \\ 1 \ \ ako\ je \ \ \sum w_j \cdot x_j \ > Prag \end{cases} \quad [4] \qquad (3.1.)$$

U današnje vrijeme većinom se koriste druge vrste umjetnih neurona koji isto kao i perceptron mogu imati jedan ili više ulaza, težine pridružene tim ulazima, a uz težine imaju i parametar zvan pristranost (*eng. Bias*) neurona. Pristranost neurona je parametar kojim se može utjecati na izlaz neurona tj. dodavanjem pristranosti olakšava se neuronu da na izlazu da željni rezultat. Težine i pristranost neurona zajedno se nazivaju parametrima neurona. Za razliku od perceptrona današnji neuroni koriste aktivacijske funkcije umjesto praga. Aktivacijska se funkcija primjenjuje na zbroju sume umnožaka ulaza neurona i njihovih težina s pristranosti neurona. Ona odlučuje hoće li neuron prenijeti vrijednost na izlaz tj. hoće li se aktivirati ili ne. Model neurona prikazan je na slici 3.2. . U literaturi neuron se često naziva čvor ili aktivacija.

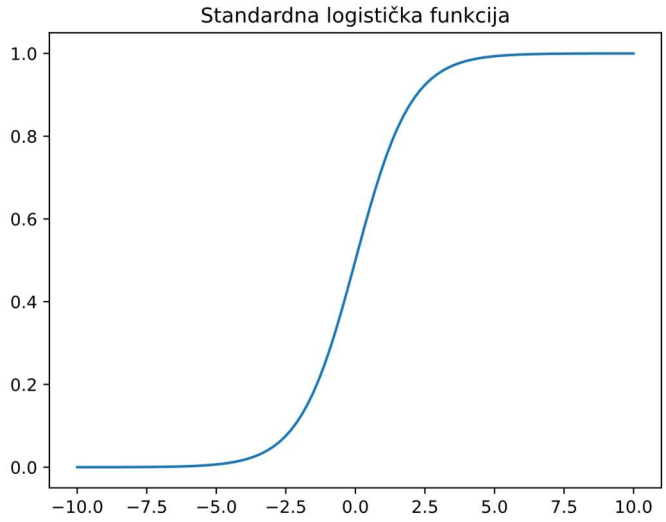


Sl. 3.2. Model neurona.

Postoji mnoštvo aktivacijskih funkcija, a neke od najkorištenijih su *ReLu*, *Softmax* i standardna logistička funkcija *(eng. Sigmoid function)*. [5] Standardna logistička funkcija se često koristi u zadnjem sloju neuronskih mreža kod logističke regresije, zbog toga što primjenom te funkcije na izlazu neuronske mreže se dobiju realni brojevi u rasponu od 0 do 1. *Softmax* funkcija se, isto kao standardna logistička funkcija, često primjenjuje na izlazu neuronskih mreža, a koristi se kad je potrebno da zbroj izlaza neuronske mreže bude jednak 1. Prema [6] *ReLu* funkcija je najkorištenija aktivacijska funkcija zbog toga što ima puno bolju propagaciju gradijenta, a to dovodi do smanjenja vjerojatnost pojave problema nestajućeg gradijenta *(eng. Vanishing gradient)* koji kod optimizacija temeljenih na gradijentu dovodi do ne značajne promjene parametara, jer je gradijent jako mali ili jednak nuli.



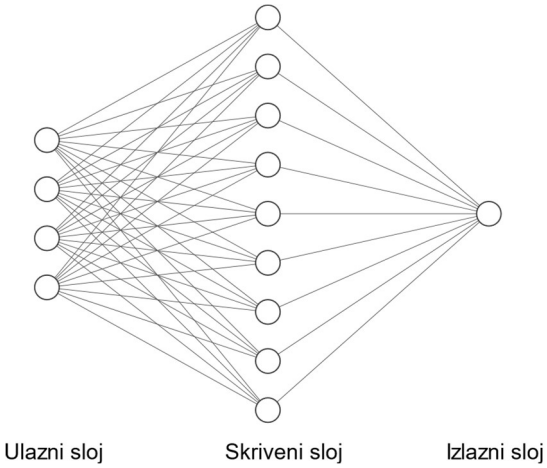
Sl. 3.3. Graf *ReLu* funkcije.



Sl. 3.4. Graf standardne logističke funkcija.

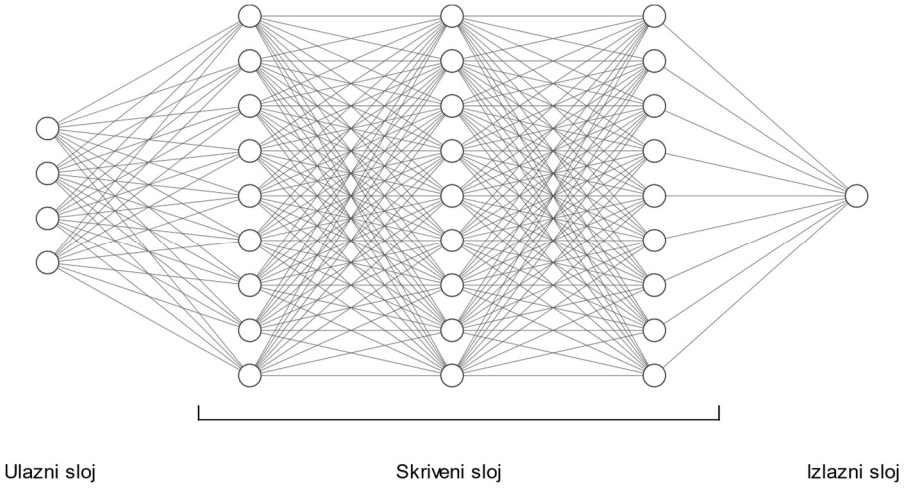
3.2. Neuronska mreža

Slaganjem pojedinačnih neurona tako da svakom neuronu ulaz i izlaz budu paralelni drugim neuronima dobije se sloj neurona, a povezivanjem slojeva neurona tako da se izlaze jednog sloja poveže s ulazima drugog sloja nastaje neuronska mreža (Slika 3.5). Slojeve neuronske mreže dijeli se na ulazni *(eng. Input layer)*, skriveni *(eng. Hidden layer)* i izlazni sloj *(eng. Output layer)*.



Sl. 3.5. Neuronska mreža.

Sam naziv dubokog učenja dolazi od činjenice da skriven sloj ne mora nužno biti jedan sloj, on se u većini slučajeva sastoji od više slojeva tj. ima dubinu te se takva neuronska mreža zove duboka neuronska mreža (*eng. Deep neural network*) (Slika 3.6.). [6] Neuronska mreža kojoj su izlazi prethodnog sloja povezani na svaki čvor sljedećeg sloja naziva se potpuno povezana neuronska mreža *(eng. Fully connected neural network)*.



Sl. 3.6. Duboka neuronska mreža.

Ovo je simbolički prikaz neuronske mreže. Ona predstavlja funkciju koja na temelju primljenih podataka daje rješenje problema. U svojoj knjizi *Deep Learning for Coders with Fastai and Pytorch: AI Applications Without a PhD* poduzetnik Jeremy Howard i Sylvain Gugger predavač matematike i računalnih znanosti opisali su neuronske mreže kao visoko fleksibilne funkcije koje su sposobne riješiti širok spektar problema. Zbog te njihove visoke fleksibilnosti one su prikladan način rješavanja mnogih problema, a glavni zadatak je pronalazak prikladnih parametara koji će dati što bolje rezultate. Kako bi se mogla trenirati neuronska mreža (model) potreban je skup podataka na kojemu će model izvoditi predviđanja. Skup podataka mora

sadržavati podatke i oznake *(eng. Labels)* što ti podatci predstavljaju. [6] Treniranje modela može se podijeliti na nekoliko dijelova:

1. Nasumična inicijalizacija parametara neuronske mreže
2. Model vrši predviđanje nad skupom ulaznih podataka
3. Mjerimo točnost modela
4. Optimiziramo parametre modela
5. Ponavljanje koraka 2-4 sve dok se ne postigne zadovoljavajuća točnost predviđanja

3.3. Gradijentni spust

Gradijentni spust predstavlja način optimizacije parametara modela. Koristi se za pronalazak globalnog minimuma funkcije gubitka $J(w, b)$. [7] Funkcija gubitka govori kako dobro ili loše model predviđa. Jedna od često korištenih funkcija gubitka je srednje kvadratno odstupanje prikazana izrazom 3.2.

$$J(w,b) = \frac{1}{2}\sum_{i}(željeni\ rezultat^{i} - rezultat^{i})^{2} \text{ [7]} \tag{3.2.}$$

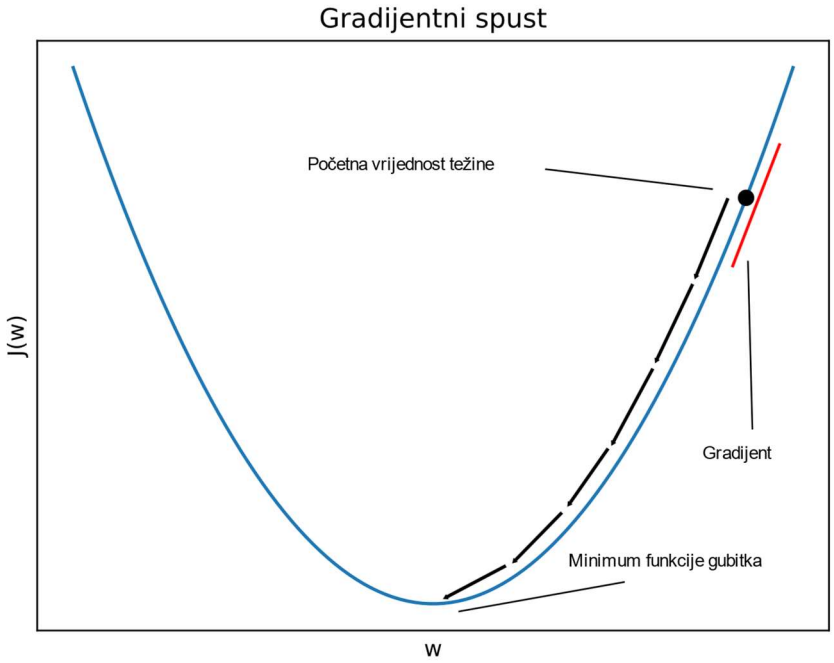
Veličinu vrijednosti za koju se optimiziraju parametri modela i smjer u kojem će se optimizirati računa se prema izrazima 3.3.

$$\Delta w = -\alpha \frac{\partial J(w,b)}{\partial w} \qquad \Delta b = -\alpha \frac{\partial J(w,b)}{\partial b} \text{ [7]} \tag{3.3.}$$

Gdje je α stopa učenja tj. broj koji govori za koliko će se optimizirati parametri modela. Parametri se optimiziraju nakon svakog prolaza *(eng. Epoch)* kroz skup podataka za treniranje prema izrazima 3.4.

$$w = w + \Delta w \qquad b = b + \Delta b \quad [7] \tag{3.4.}$$

Gdje je Δw veličina za koju je potrebno modificirati težinu, a Δb veličina za koju je potrebno modificirati pristranost neurona. Gradijentni spust može se prikazati jednostavnim grafom ovisnosti funkcije gubitka o vrijednosti težine (slika 3.7.).

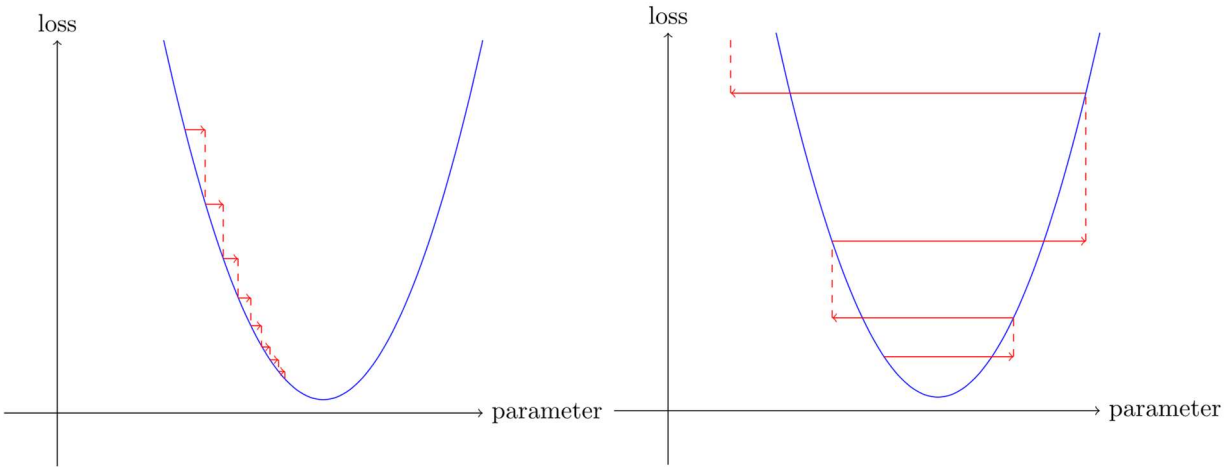


Sl. 3.7. Gradijentni spust.

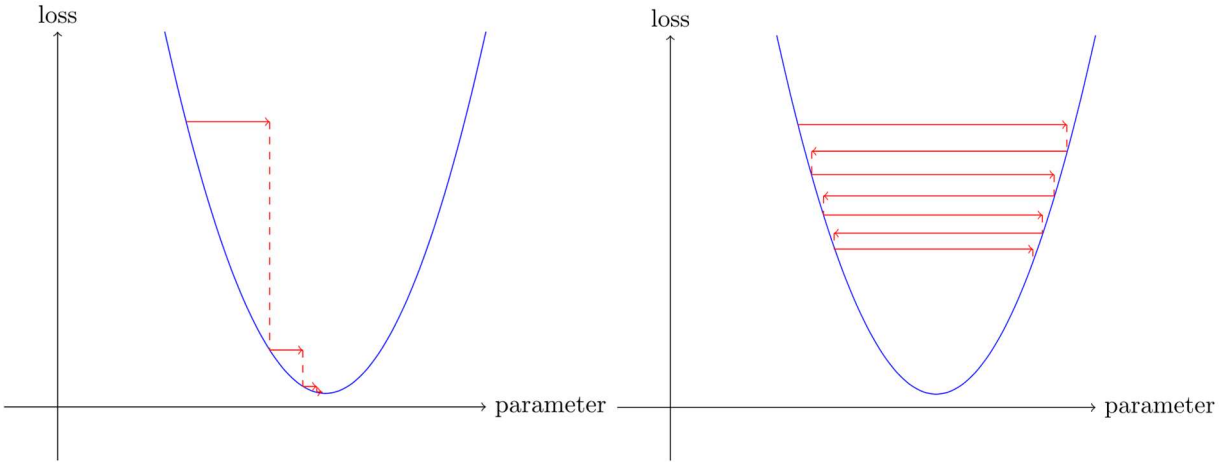
Uz gradijentni spust postoje još dvije verzije, stohastički gradijentni spust i mini-batch gradijentni spust. Kod stohastičkog gradijentnog spusta optimizaciju parametara radi se prilikom prolaska kroz svaki trening uzorak, a kod mini-batch gradijentnog spusta modificiranje parametara radi se na prolasku kroz definirane grupe trening uzoraka. Ove vrste optimizacije parametara koriste se u algoritmu zvanom unatražno širenje greške *(eng. Backpropagation)*. [6] On je zadužen za prosljeđivanje greške unazad kako bi se svi parametri modela mogli optimizirati.

Pravilnim odabirom stope učenja značajno se pospješuje brzina treninga modela, kod vrlo male stope učenja treniranje modela traje duže vremena nego što je potrebno. Ako je stopa učenja velika, prilikom treniranja modela dolazi do prevelike modifikacije težina i pristranosti neurona,

te se prelazi preko globalnog minimuma funkcije gubitka i smanjuje se preciznost modela. [6] Pojedine stope učenja prikazane su slikama 3.8 i 3.9.



Sl. 3.8. Stopa učenja: mala (lijevo) i velika (desno). [5]



Sl. 3.9. Stopa učenja: odlična (lijevo) i velika uz odbijanje (desno) [5]

3.4. Prenaučenost

Prilikom treniranja neuronskih mreža poželjno je skup podataka na kojemu se želi trenirati model podijeliti na skup podataka za trening *(eng. Training data set)* i validacijski skup podataka *(eng. Validation data set).* Validacijski skup podataka je potreban kako bi se vidjelo kako dobro model predviđa rješenja nad skupom podataka koje prethodno nije vidio. Ako nakon treniranja model ima slabu točnost predviđanja nad validacijskim skupom podataka, može se pretpostaviti

da je došlo do prenaučenosti *(eng. Overfitting)*, jednog od najvećih izazova dubokog učenja. Prenaučenost se može prepoznati po malom gubitku modela nad skupom podataka za trening i gubitku nad validacijskim skupom koji je veći od gubitka ostvarenog nad skupom podataka za trening, a uz dodatno treniranje isti raste. Da bi se spriječila prenaučenost, trening modela dijeli se na epohe. Epoha predstavlja jedan prolaz kroz skup podataka za trening, a omogućava zaustavljanje treniranja modela čim dođe do prenaučenosti.

3.5. Regularizacija

Regularizacija je optimizacijska tehnika kojoj je glavni zadatak otežati nastajanje prenaučenosti. Danas se najčešće koriste neke od sljedećih tehnika regularizacije:

1. Kažnjavanjem normi težina.
2. Generiranjem novih podataka.
3. Ranim zaustavljanjem.
4. Regularizacija isključivanjem čvorova *(eng. Dropout)*.

Kod regularizacije kažnjavanjem normi težina kažnjavaju se vršne vrijednosti težina dodavanjem kazne funkciji gubitka. Dodani dio sadrži u sebi hyper-parametar λ koji se naziva snaga regularizacije. [2] Kod ove vrste regularizacije razlikuju se dva tipa: L1 i L2 regularizaciju.

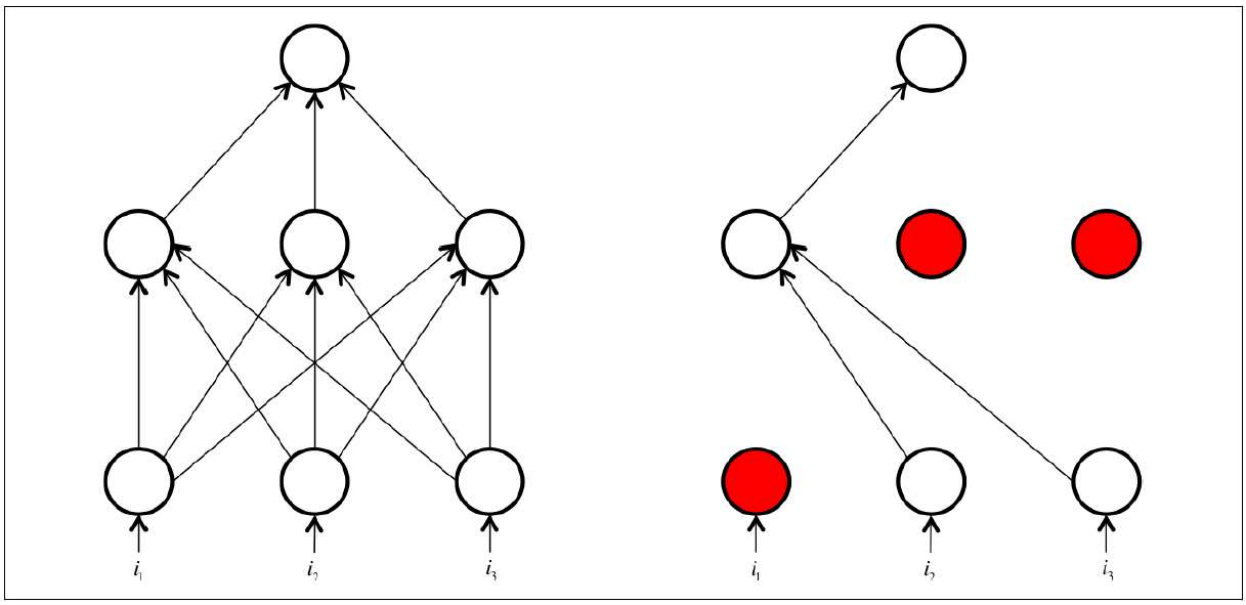
L2 regularizacija značajno kažnjava vršne vrijednosti težina, a preferira korištenje više umjerenih vrijednosti težina. Ovakva implementacija potiče mrežu na korištenje što više ulaza pojedinog neurona u odnosu na korištenje samo nekih ulaza većinu vremena. Prilikom korištenja gradijentnog spusta za optimizaciju težina, korištenjem L2 regularizacije svaka težina se linearno približava nuli te se L2 regularizacija često naziva iščezavanje težine *(eng. Weight decay)*. Kod L1 regularizacije težine prilikom optimizacije dovodi se blizu nule, što za posljedicu neuronsku mrežu potiče na korištenje samo male podgrupe svojih najznačajnijih ulaza.

Generiranje novih podataka iz skupa podataka predstavlja efikasnu vrstu regularizacije jer povećava veličinu skupa podataka tj. količinu različitih ulaznih podataka. Kako bi se generirali novi podatci koriste se postojeći, koji se izmjenjuju tako da imaju drugačiji oblik, ali zadržavaju svoje originalno značenje. U računalnom vidu to se često postiže rotacijom, okretanjem, promjenom perspektive, kontrasta ili svjetline slike (Slika 3.10). Kod obrade prirodnog jezika to se radi prevođenjem podataka na neki drugi jezik, te se tako prevedene podatke prevodi u izvorni jezik, sto za posljedicu ima izmijenjene izvorne podatke sa zadržanim značenjem tih podataka.

Sl. 3.10. Primjer generiranja novih podataka iz postojećih. [5]

Regularizacija ranim zaustavljanjem je najkorišteniji tip regularizacije zbog svoje efikasnosti i jednostavnosti. Kod ovog tipa regularizacije prilikom treniranja modela potrebno je povremeno evaluirati model nad validacijskim skupom podataka. [6] Prilikom validacije uspoređuju se performanse parametara modela te se pamte parametri koji ostvaruju bolji rezultat. Zaustavljanje se radi kad model ne ostvari bolje rezultate nad validacijskim skupom podataka određeni broj evaluacija.

Jedna od najomiljenijih današnjih metoda prevencije prenaučenosti u dubokim neuronskim mrežama je isključivanje čvorova (eng. *Dropout)*. [2] Kod ove metode prilikom treniranja svaki neuron ima određenu vjerojatnost hoće li biti aktivan ili ne. Ovime se postiže da model bude precizan čak i uz nedostatak određenih informacija, te se osigurava da model ne postane previše ovisan o određenim neuronima ili malim kombinacijama neurona unutar mreže. Isključivanjem čvorova simulira se korištenje kombinacije neuronskih mreža različitih arhitektura. Kod ove metode neurone se isključuje samo kod faze treniranja modela. Prilikom testiranja i validacije svi neuroni mreže moraju biti aktivni. Isključivanje čvorova prikazano je na slici 3.11.



Sl. 3.11. Isključivanje čvorova [2]

4. OBRADA PRIRODNOG JEZIKA

Obrada prirodnog jezika (*eng. Natural language processing*) je interdisciplinarno polje koje povezuje računalnu znanost i lingvistiku. Ona se bavi razvijanjem algoritama i metoda koje se koriste u svrhu razumijevanja i generiranja prirodnog jezika u govornom i tekstualnom obliku. [8] Prema tome može se ugrubo podijeliti na generiranje teksta i govora te razumijevanje teksta i govora. Sami početci obrade prirodnog jezika bavili su se problemom prevođenja jezika, a korištene metode su temeljene na ručno pisanim pravilima. S prolaskom vremena i pojavom strojnog učenja, metode obrade koje su se tada uvelike temeljile na skupu pravila za obradu, prelaze na statistički pristup kod kojeg računalo na temelju velike količine predanih podataka uči pravila za obradu prirodnog jezika. Pojavom dubokog učenja obrada prirodnog jezika je prešla je na neuronski pristup, te se danas svi najbolji sustavi temelje na dubokom učenju. Tijek obrade teksta može se podijeliti na: predobradu teksta, treniranje vektorskih reprezentacija riječi (*eng. Word embeddings*) te treniranje i validaciju modela. Obrada se vrši nad tekstom tj. kolekcijom tekstova koji se nazivaju tekstualni korpus ili jednostavnije korpus (*eng. Corpus*). Korpus se koristi za statističku analizu, testiranje hipoteza i potvrđivanje lingvističkih pravila nekog jezika, a predstavlja velike količine strukturiranih skupova tekstova koji su većinom spremljeni u elektroničkom obliku. Neki od poznatijih primjera tekstualnih korpusa su wikitext103 i Wordnet.

4.1. Regularni izrazi

Regularni izrazi (*eng. Regular expressions*) predstavljaju jezik pomoću kojeg specificiramo tekstualni izraz za pretragu, izmjenu ili validaciju određenog teksta, primjenjuju se u raznim računalnim jezicima, programima za obradu teksta te raznim alatima za obradu teksta kao što je poznati Linux/Unix-ov `grep` te igraju bitnu ulogu u normalizaciji teksta. [9] Kod primjene regularnih izraza potreban je izraz kojim se želi pronaći određeni uzorak teksta te tekstualni korpus nad kojim se vrši pretraga. Pretraga nad tekstualnim korpusom vratit će sve riječi koje se podudaraju s izrazom. Naj jednostavniji oblik regularnog izraza predstavlja pretraga riječi, te ako se želi pronaći riječ *model* upotrijebit će se izraz `/model/`. Regularni izrazi su osjetljivi na velika i mala slova pa se tako prethodno navedeni izraz neće podudarati s riječi *Model,* ali ako je potrebno da se i ta riječ podudara potrebno je upotrijebiti uglate zagrade `[]`. Grupa znakova unutar uglatih zagrada predstavlja disjunkciju znakova koje se želi pronaći pa tako izraz `/[mM]/` vraća slova *m* ili *M*, a prema tome izraz `/[mM]odel/` pronalazi riječi *model* i *Model.* Izraze s uglatim zagradama može se primijeniti i na brojevima pa se izraz `/[0123456789]/` podudara s bilo kojom znamenkom iz ovog seta znakova, isto tako vrijedi i za slova, ali kako je pisanje svih

znamenki od nule do devet ili pisanje cijele abecede veoma nepraktično, regularni izrazi koriste oznaku - za specificiranje intervala. Pa se prethodni izraz za pronalazak znamenki može zapisati kao `/[0-9]/`. Najčešće uporabe te oznake prikazane su tablicom 4.1.

Tablica 4.1. Česta primjena znaka – [9]

Regularni izraz	*Podudaranje*	*Primjer podudaranja*
`/[A-Z]/`	Veliko slovo	Danas je sunčan dan.
`/[a-z]/`	Malo slovo	Danas je sunčan dan.
`/[0-9]/`	Znamenka	12

Uglate zagrade mogu pomoći i kod znakova koje se ne želi pronaći, a to je omogućeno kombinacijom uglatih zagrada sa znakom ^ koji će negirati pronađeni uzorak. Kako bi on izvršio navedenu funkciju mora se nalaziti odmah iza otvorene uglate zagrade, u suprotnom gubi funkciju negiranja te predstavlja jedan od znakova koje se želi pronaći primjer uporabe ^ znaka prikazan je tablicom 4.2.

Tablica 4.2. Česta primjena znaka ^ [9]

Regularni izraz	*Podudaranje*	*Primjer podudaranja*
`/[^A-Z]/`	Malo slovo	Danas je sunčan dan.
`/[^a-z]/`	Veliko slovo	Danas je sunčan dan.
`/[^0-9]/`	Slovo	marko123

Ovo su samo neki od izraza, sve regularne izraze može se podijeliti prema vrsti tokena od kojih su napravljeni, prema tome može ih se podijeliti na: osnovne tokene, znakovne klase, specijalne znakove, brojače, grupe, tokene za zamjenu uzoraka i tvrdnje. Lista regularnih izraza prikazana je na tablici 4.3.

Tablica 4.3. Lista regularnih izraza [10]

Regularni izraz	*Opis*
`[ABC]`	Pronalazi sve znakove unutar seta
`[^ABC]`	Pronalazi sve znakove
`[A-Z]`	Pronalazi sve znakove dane intervalom uključujući i granice
`.`	Pronalazi znakove koji nisu prekid linije
`[\s\S]`	Pronalazi bilo koji znak
`\w`	Pronalazi bilo koji alfanumerički znak i donju crtu
`\W`	Pronalazi znakove koji nisu alfanumerički ili donja crta
`\d`	Pronalazi znamenke
`\D`	Pronalazi znakove koji nisu znamenke
`\s`	Pronalazi razmak
`\S`	Pronalazi znakove koji nisu razmaci
`^`	Pronalazi početak riječi ili rečenice
`$`	Pronalazi kraj riječi ili rečenice
`(ABC)`	Grupira višestruke tokene i stvara obuhvatnu grupu za izvlačenje dijela riječi

Regularni izraz	*Opis*
`(?:ABC)`	Grupira višestruke tokene bez stvaranja obuhvatne grupe
`(?<name>ABC)`	Stvara obuhvatnu grupu kojoj se može pristupiti specificiranim imenom
`\1`	Podudara se s rezultatom obuhvatne grupe
`+`	Pronalazi 1 ili više ponavljanja prethodnog tokena
`*`	Pronalazi 0 ili više ponavljanja prethodnog tokena
`{n}`	Pronalazi n ponavljanja prethodnog tokena
`?`	Pronalazi 0 ili 1 ponavljana prethodnog tokena (čini ne obaveznim)
`?`	Čini prethodni brojač lijenim te on pronalazi što je manje moguće znakova
`|`	Ponaša se kao logički operator ILI te pronalazi izraz prije ili poslije znaka |

4.2. Normalizacija teksta

Normalizacija teksta predstavlja proces u kojem se tekst dovodi u standardizirani oblik koji je pogodan za daljnju obradu. [9] Normalizacija teksta se radi pomoću jedne od sljedećih metoda:

1. Tokenizacija ili segmentacija
2. Lematizacija
3. Korjenovanje
4. Uklanjanje riječi koje nose malo značenja, te prilagodba riječi za lakšu obradu

4.2.1. Tokenizacija

Tokenizacija teksta predstavlja proces kod kojeg se tekst razdvaja na segmente tj. tokene kako bi se tekst pripremio za daljnju obradu. Primjer tokenizacije pomoću *Python*-ove biblioteke `nltk` prikazan je slikom 4.1.

Linija ***Kod***

```
1:      from nltk.tokenize import word_tokenize
2:      data = """Lorem ipsum dolor sit amet, consectetur
3:             adipiscing elit, sed do eiusmod tempor incididunt
4:             ut labore et dolore magna aliqua."""
5:      print(word_tokenize(data))
Izlaz:  ['Lorem', 'ipsum', 'dolor', 'sit', 'amet', ',', 'consectetur',
        'adipiscing', 'elit', ',', 'sed', 'do', 'eiusmod', 'tempor',
        'incididunt', 'ut', 'labore', 'et', 'dolore', 'magna', 'aliqua',
        '.']
```

Sl. 4.1. Tokenizacija

Tokenizacija nije jednostavan proces jer prilikom razdvajanja teksta u segmente mora se voditi računa o značenju riječi, interpunkcijskim znakovima i njihovim položajima te dijelovima teksta koji zajedno čine entitet kao što su više riječna imena naselja ili pak kombinacije specijalnih znakova i riječi kao što je znak „#“ koji se često koriste na društvenim mrežama u kombinaciji s trendovima (npr. #COVID19). U te svrhe razvijane su različite implementacije tokenizatora koji uzimaju u obzir prethodno navedene zahtjeve prilikom segmentacije te se danas razlikuje tri glavna pristupa:

1. Tokenizacija po riječima
2. Tokenizacija po dijelovima riječi
3. Znakovna tokenizacija

Osim što se prilikom tokenizacije tekst razdvaja na segmente, tim istim segmentima se pridjeljuju brojčane vrijednosti koje ih reprezentiraju. Najčešće se to obavlja tako da se tokeni poredaju po frekvenciji pojavljivanja u tekstu krenuvši od najčešćih te se zatim stvaraju parovi tokena i njihovih brojčanih reprezentacija. Prilikom tokenizacije može se definirati konačan broj tokena te se tako filtriraju tokeni koji se rijetko pojavljuju, a pojave velikih slova, specijalnih znakova i tokena koje smo filtrirali možemo zamijeniti specijalnim tokenima.

Tablica 4.4. Specijalni tokeni [5]

Specijalni token	***Opis***
`(xxunk)`	Nepoznata riječ
`(xxpad)`	Token za popunjavanje
`(xxbos)`	Označava početak teksta
`(xxmaj)`	Označava da sljedeća riječ počinje velikim slovom
`(xxup)`	Označava da je sljedeća riječ pisana velikim slovima
`(xxrep)`	Označava da se sljedeći znak ponavlja n puta u tekstu
`(xxwrep)`	Označava da se sljedeća riječ ponavlja n puta u tekstu

Primjer specijalnih tokena prikazan je tablicom 4.4.

4.2.2. Normalizacija riječi, lematizacija i korjenovanje

Normalizacija riječi predstavlja proces pretvaranja riječi u standardizirani oblik tj. riječi koje imaju više oblika svode se na jedan normalizirani oblik. Jedan od oblika normalizacije riječi je promjena velikih u mala slova, riječi *Tata* i *tata* uz različit izgled imaju isto značenje, te se *Tata* svodi na *tata.* Takve promjene se izbjegavaju kad se pretvaranjem velikih u mala slova mijenja značenje riječi, primjer takvih riječi su *kum* i *KUM,* prva riječ označava svjedoka na krštenju ili

vjenčanju dok druga označava *Korčulansku udrugu mladih.* Takve promjene često nisu poželjne te se izbjegavaju prilikom strojnog prevođenja, izvlačenja informacija i klasifikacije teksta.

Lematizacija je proces u kojemu se različite oblike riječi svodi na izvorni oblik, jedan primjer lematizacije je svođenje glagolskih oblika na njihov izvorni oblik, uzmu li se na primjer glagolski oblici *sam, si, je, smo, ste, su* i njihov izvorni oblik *biti,* vidi se da je šest riječi zamjenjujemo jednom riječju, što je i cilj normalizacije. Lematizacija se često obavlja jednostavnim postupcima korjenovanja gdje se na temelju pisanih pravila uklanjaju prefiksi i sufiksi riječi kako bi se dobila jedna izvorna riječ. [9] Jedan od poznatijih algoritama za korjenovanje engleskog jezika je *Porter Stemmer,* preda li mu se rečenica:

> *The Lorem ipum filling text is used by graphic designers, programmers and printers with the aim of occupying the spaces of a website, an advertising product or an editorial production whose final text is not yet ready.*

na izlazu će se dobiti sljedeće:

> *the lorem ipum fill text is use by graphic design , programm and printer with the aim of occupi the space of a websit , an advertis product or an*
> *editori product whose final text is not yet readi .*

4.3. Reprezentacija riječi i njihovog značenja

Jedan od izazova prilikom obrade teksta je reprezentacija riječi i njihova značenja. Postoje različiti načini njihove reprezentacije, od jednostavnih numeričkih reprezentacija, statističkih i nešto naprednijih reprezentacija koje su naučene korištenjem neuronskih mreže.

4.3.1. One-hot vektor

Najjednostavniji način reprezentacije riječi je pomoću vektora koji u sebi sadrži nule i jednu jedinicu koja predstavlja riječ. Pred obradom teksta dobije se rječnik određenje veličine, koji je sastavljen od parova riječi i njihovih indeksa. *One-hot* vektor dobije se tako da se odabranu riječ iz rječnika reprezentira tako da se na indeksu gdje se ona nalazi postavi jedinica, a na sve ostale indekse postave se nule. Odabere se za primjer rečenica: „*Kako je danas lijep i sunčan dan.*“. Rečenica se sastoji od sedam različitih riječi, prema tome za reprezentaciju bilo koje riječi potreban je vektor veličine sedam elemenata (Tablica 4.5.). Problem kod ovog načina reprezentacije je u tome što riječi nisu povezane tj. ne može se odrediti veze između njih, njihove

sličnosti ili njihov kontekst, a ovisno o veličini rječnika tekstualnog korpusa, vektori mogu biti velikih dimenzija.

Tablica 4.5. *One-hot* reprezentacija

Riječ	***Vektor***
Kako	[1,0,0,0,0,0,0]
je	[0,1,0,0,0,0,0]
danas	[0,0,1,0,0,0,0]
lijep	[0,0,0,1,0,0,0]
i	[0,0,0,0,1,0,0]
sunčan	[0,0,0,0,0,1,0]
dan	[0,0,0,0,0,0,1]

4.3.2. *Bag-of-words* model

Bag-of-words model predstavlja jednostavni model reprezentacije teksta. On predstavlja tekst kao skup riječi u kojemu se zanemaruje gramatika i poredak riječi, a zadržava se samo broj pojavljivanja neke riječi u tekstu. Tako će se rečenicu „*Kako je danas lijep i sunčan dan, a jučerašnji je bio kišovit i oblačan dan.*“ reprezentirati vektorom: [1,2,1,1,2,1,2,1,1,1,1,1] . Problem kod ovog modela je u tome što ne uzima u obzir značenje riječi i njihov kontekst.

4.3.3. TF-IDF

TF-IDF predstavlja algoritam koji se koristi nad kolekcijom dokumenata, a osnovni zadatak mu je odrediti važnost ili težinu riječi unutar kolekcije dokumenta. Algoritam predstavlja umnožak dvaju veličina, prva od njih je frekvencija pojma (*eng. Term frequency*), a najčešće predstavlja broj ponavljanja neke riječi unutar dokumenta. Ona se često normalizira logaritmiranjem, jer riječ koja se ponavlja 200 puta u nekom dokumentu neće biti 200 puta bitnija u određivanju značenja tog dokumenta. [9] Frekvencija pojma računa se prema izrazu 4.1. gdje je *t* predstavlja pojam, a *d* dokument u kojem se pojam nalazi.

$$tf_{t,d} = log_{10}(cout(t,d)) \text{ [9]} \tag{4.1.}$$

Druga veličina je inverzna frekvencija dokumenta (*eng. Inverse document frequency*), ona određuje koje su riječi važne za značenje dokumenata s obzirom na broj ponavljana u dokumentima i broju dokumenata u kojima se nalazi. Uzme li se na primjer da postoje dvije riječi koje imaju jednak broj ponavljanja u 4 dokumenta, ali prva riječ se pojavljuje samo u jednom od ta četiri dokumenta dok druga u svim. Prema tome prva riječ više doprinosi značenju

tih dokumenata, a upravo to IDF pokušava odrediti. Računa se prema izrazu 4.2. gdje je *N* broj dokumenata unutar kolekcije, a df_t predstavlja broj dokumenata u kojima se pojam pojavljuje (*eng. Document frequency*).

$$tf_{t,d} = log_{10}(\frac{N}{df_t}) \text{ [9]} \quad (4.2.)$$

Umnoškom TF-a i IDF-a će se dobiti važnost neke riječi za kolekciju dokumenata, a računa se pomoću izraza 4.3.

$$w_{t,d} = tf_{t,d} \times idf_t \text{ [9]} \quad (4.3.)$$

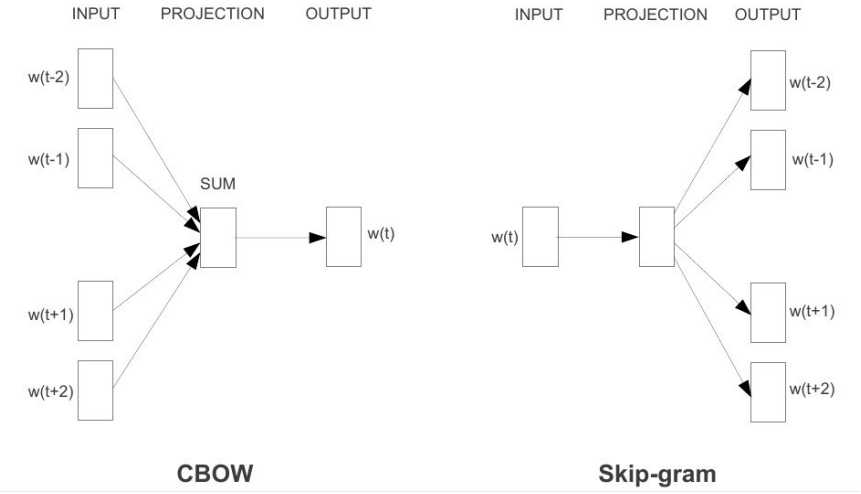
4.3.4. Word2vec

Word2vec je neuronski pristup reprezentaciji riječi, a predstavlja rješenje prethodnih metoda koje nisu dobro reprezentirale veze i sličnosti između riječi. *Word2vec* nastoji stvoriti vektorsku reprezentaciju riječi (*eng. Word embeddings*) koja bi riječi smjestila u vektorski prostor (*eng. Vector space*) prema kontekstu u kojem se riječi nalaze. [11] Tako bi u vektorskom prostoru riječi koje se pojavljuju u sličnom kontekstu bile blizu jedne drugima. Uzmu li se na primjer riječi *majka, otac, žena i muškarac.* U vektorskom prostoru riječi *majka* i *žena* bi trebale biti smještene blizu jedna druge, a ista takva povezanost bi trebala biti i s riječima *otac* i *muškarac.* Upravo takvu reprezentaciju postiže *Word2vec*. Primijeni li se izraz *X = vektor(„majka“) - vektor(„žena“) + vektor(„otac“)* na dobro naučenim reprezentacijama riječi trebali bi dobiti točnu vektorsku reprezentaciju riječi *muškarac.*

Word2vec predstavlja jednostavnu unaprijednu neuronsku mrežu koja se sastoji od ulaznog sloja, projekcijskog sloja čiji je zadatak smanjiti diskretne vektore velikih dimenzija u kontinuirani vektor manjih dimenzija, te izlazni sloj neuronske mreže.

Word2vec predstavlja dvije arhitekture za učenje vektorskih reprezentacija:

1. CBOW
2. Skip-gram



Sl. 4.2. *Word2vec* arhitekture [11]

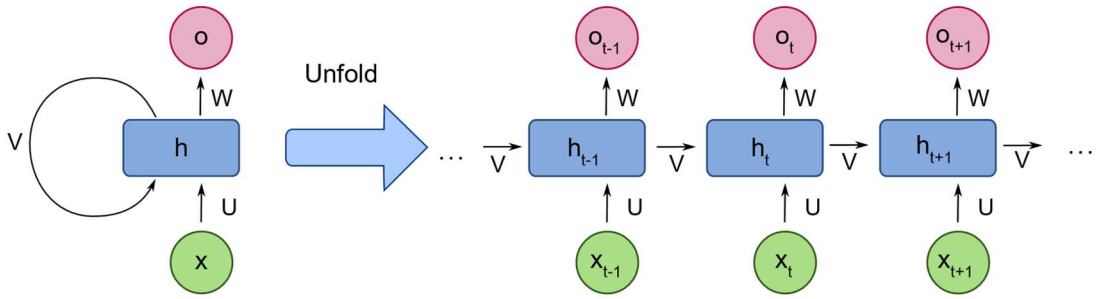
Prema slici vidimo da CBOW arhitektura predviđa trenutnu riječ iz konteksta, dok Skip-gram arhitektura predviđa kontekst iz trenutne riječi.

4.4. Duboke arhitekture za obradu prirodnog jezika

Prirodni jezik predstavlja slijed podataka kroz vrijeme, svaki tekst ili govor ljudi „obrađuju“ riječ po riječ, te svaka prethodna riječ doprinosi značenju sljedeće. Baš zbog te prirode jezika razvijene su posebne arhitekture neuronskih mreža koje su sposobne obraditi tekst varijabilne duljine, uzimajući u obzir prethodne riječi. Danas se raspoznaje nekoliko takvih arhitektura koje obrađuju sljedove riječi, a jedna od takvih je povratna neuronska mreža.

4.4.1. Povratna neuronska mreža

Povratna neuronska mreža (*eng. Recurrent neural network*) je svaka neuronska mreža koja za donošenje odluke ovisi o svome prethodnom izlazu. Jedna takva neuronska mreža je jednostavna povratna mreža (*eng. Simple recurrent network*). Ona se može prikazati kao unaprijedna neuronska mreža koja ima povratnu vezu u svom skrivenom sloju. Vrijednosti parametara skrivenog sloja ovise o ulaznom sloju i vrijednostima parametara skrivenog sloja iz prethodnog vremenskog koraka (*eng. Time step*) u kojemu je enkodirana informacija o prethodnim stanjima tj. predstavlja memoriju. Ovakvu se neuronsku mrežu može „razmotati“ kroz vrijeme za lakše razumijevanje kao što je prikazano slikom 4.3.



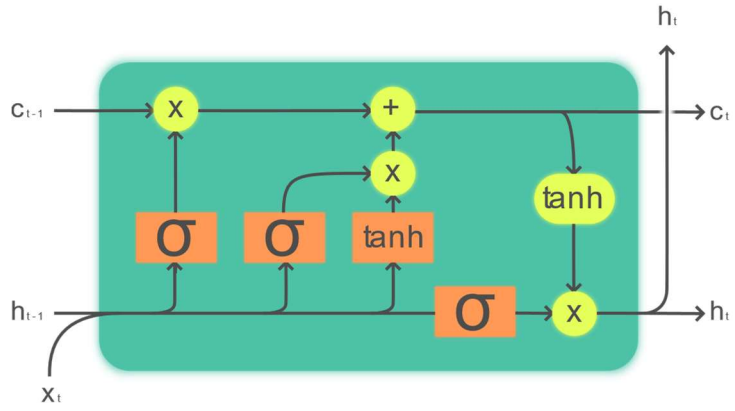
Sl. 4.3. Povratna neuronska mreža razmotana kroz vrijeme [12]

Na slici se vidi da kod povratne neuronske mreže postoje tri matrice parametara: matrica *U* koja povezuje ulazni sloj sa skrivenim slojem trenutnog vremenskog koraka, matrica *V* koja povezuje skriveni sloj prethodnog vremenskog koraka sa skrivenim slojem trenutnog i matricu *W* koja povezuje skriveni sloj trenutnog vremenskog koraka s izlaznim slojem. Kako bi se dobili izlazi povratne neuronske mreže ulazu je potrebno predati vektorsku reprezentaciju riječi koja se množi matricom parametara *U,* zatim stanje skrivenog sloja iz prethodnog vremenskog koraka se množi matricom *V,* nakon toga potrebno je zbrojiti vrijednosti dobivene iz prethodne dvije operacije te primijeniti aktivacijsku funkciju kako bi se dobile vrijednosti skrivenog sloja trenutnog vremenskog koraka. Nakon toga, stanje skrivenog sloja množi se matricom *W* te se primjenjuje funkcija za generiranje probabilističke razdiobe kako bi se dobili izlaz kao vjerojatnost pojave određene riječi iz rječnika.

Treniranje mreže se vrši unatražnim širenjem pogreške, stanja na ulaznom, skrivenom i izlaznom sloju su različita za svaki vremenski korak, dok su matrice parametara *U*, *V* i *W* zajedničke za svaki vremenski korak. Kako su matrice zajedničke za sve vremenske korake, a u svakom koraku se parametri množe, dolazi do pojave nestajućih gradijenata tj. neuronska mreža „zaboravlja“. [9] To govori da je informacija enkodirana u skrivenom sloju lokalizirana tj. ne sadrži informacije o udaljenim dijelovima sekvence. Unatoč svojoj kratkotrajnoj memoriji (*eng. Short-term memory*), povratne neuronske mreže su pogodne za obradu kratkih sekvenci teksta.

4.4.2. LSTM i GRU arhitektura

Jedno od rješenja kratkotrajne memorije povratnih neuronskih mreža je LSTM (*eng. Long short-term memory*) arhitektura (Slika 4.4.). [9] Ona se sastoji od ulaznog, povratnog i izlaznog sloja kao i povratne neuronske mreže, ali uvodi i novi kontekstni sloj (*eng. Context layer*) koji se bavi problemom konteksta tj. glavni zadatak mu je naučiti koje su informacije bitne za kontekst sekvence teksta. Uz kontekstni sloj uvode i mehanizam vrata (*eng. Gates*). Njihov zadatak je kontrola toka informacija kroz slojeve mreže. Razlikuju se vrata za brisanje informacija (*eng. Forget gate*), vrata za dodavanje informacija (*eng. Add gate*) i izlazna vrata (*eng. Output gate*). Sva vrata dijele isti dizajn, sastoje se od unaprijednog sloja, sigmoidne aktivacijske funkcije i umnoška sa slojem kojeg se želi kontrolirati.



Sl. 4.4. LSTM ćelija [12]

Zadatak vrata za brisanje informacija je obrisati informacije koje se nalaze u kontekstnom sloju a više nisu potrebne. Kako bi se informacije obrisale prvo je potrebno izračunati sumu umnoška matrice parametara s vrijednosti skrivenog stanja (*eng. Hidden state*) prethodnog vremenskog koraka i umnoška matrice parametara s trenutnim ulaznim vrijednostima. Zatim dobivenu sumu se propušta kroz aktivacijsku funkciju kako bi se dobila maska (*eng. Mask*) koja se množi kontekstnim vektorom primjenom Hadamard-ovog umnoška kako bi se obrisale informacije iz kontekstnog sloja koje više nisu potrebne. Izračun je prikazan izrazima 4.4. i 4.5.

$$f_t = \sigma(V_f h_{t-1} + U_f x_t) \text{ [9]} \tag{4.4.}$$

$$k_t = c_{t-1} \odot f_t \text{ [9]} \tag{4.5.}$$

Zadatak vrata za dodavanje informacija je dodavanje informacija iz skrivenog stanja trenutnog vremenskog koraka u kontekst kako bi se mogle koristiti u sljedećim vremenskim koracima. Prvo se računa informacija koja se želi izvući iz skrivenog stanja prethodnog vremenskog koraka i trenutnih ulaza prema izrazu 4.6.

$$g_t = tanh(V_g h_{t-1} + U_g x_t) \text{ [9]} \tag{4.6.}$$

Zatim se generira maska za dobivanje informacije koju se želi dodati u kontekstni sloj trenutnog vremenskog koraka, maska se primjenjuje na informaciji dobivenoj u prethodnom koraku. Proces izračuna prikazan je izrazima 4.7. i 4.8.

$$i_t = \sigma(V_i h_{t-1} + U_i x_t) \text{ [9]} \tag{4.7.}$$

$$j_t = g_t \odot i_t \text{ [9]} \tag{4.8.}$$

Nakon izračuna, dobivenu informaciju dodaje se u kontekst izrazom 4.9.

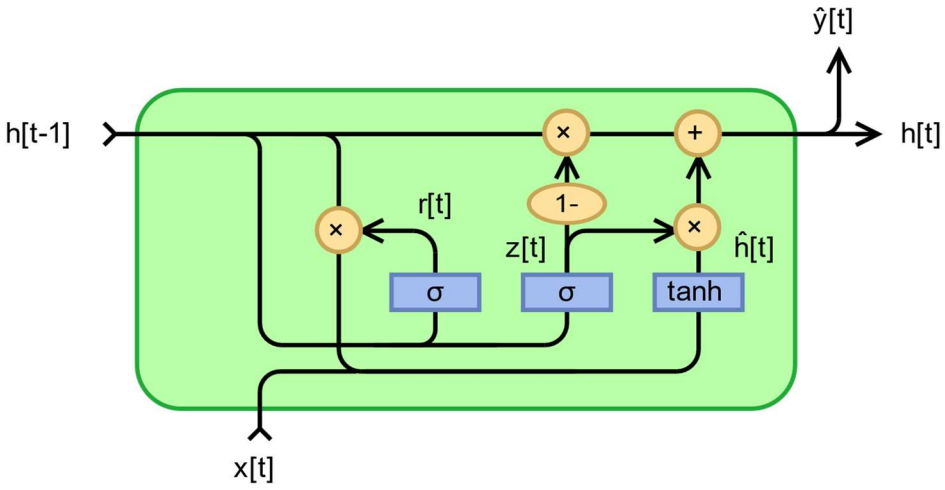
$$c_t = j_t + k_t \text{ [9]} \tag{4.9.}$$

Zadatak izlaznih vrata je izračunati vrijednost skrivenog stanja trenutnog vremenskog koraka, koji se koristi za izračun predviđanja i kao ulaz za sljedeće vremenske korake. Izračun se vrši prema izrazima 4.10. i 4.11.

$$o_t = \sigma(V_o h_{t-1} + U_o x_t) \text{ [9]} \tag{4.10.}$$

$$h_t = o_t \odot tanh\,(c_t) \text{ [9]} \tag{4.11.}$$

Problem kod LSTM arhitekture je u tome što su zahtjevne za izračunavanje jer uvode 8 matrica parametara po ćeliji koje treba trenirati. U svrhu rješavanja tog problema osmišljena je GRU (*eng. Gated recurrent unit*) arhitektura (Slika 4.5.).



Sl. 4.5. GRU ćelija [12]

Ona isto kao i LSTM uvodi mehanizme vrata, ali za razliku od LSTM-a ne koristi kontekstni sloj i uvodi samo dvoja vrata, vrata za resetiranje (*eng. Reset gate*) i vrata za ažuriranje (*eng. Update gate*). GRU koristi skriveno stanje za prijenos informacije iz prethodnih vremenskih koraka. Vrata za resetiranje *r* odlučuju koje informacije zaboraviti, a koje od novih informacija dodati (Izraz 4.12.). Vrata za ažuriranje z odlučuju koliko će se informacije iz prošlih vremenskih koraka zadržati (Izraz 4.13.).

$$r_t = \sigma(V_r h_{t-1} + U_r x_t) \text{ [9]} \tag{4.12.}$$

$$z_t = \sigma(V_z h_{t-1} + U_z x_t) \text{ [9]} \tag{4.13.}$$

Nakon izračuna vrijednosti na vratima za resetiranje prema izrazu 4.12. računa se trenutna reprezentacija skrivenog stanja izrazom 4.14.

$$\hat{h}_t = tanh(V(r_t \odot h_{t-1}) + Ux_t) \text{ [9]} \tag{4.14.}$$

Nakon izračuna trenutne reprezentacije skrivenog stanja i izračuna vrijednosti na vratima za ažuriranje može se izračunati skriveno stanje trenutnog vremenskog koraka prema izrazu

$$h_t = (1 - z_t)h_{t-1} + z_t \hat{h}_t \text{ [9]} \tag{4.15.}$$

LSTM i GRU arhitekture su vidjele veliku primjenu zbog svoje modularnosti tj. njihova kompleksnost se može enkapsulirati u neuronske jedinice koje se mogu integrirati u druge arhitekture.

5. KLASIFIKACIJA TEKSTA

U ovom dijelu završnog rada opisan je praktični dio. Zadatak praktičnog dijela je stvoriti model koji će biti spodoban klasificirati recenzije digitalne distribucijske platforme *Steam*. Radi se o binarnoj klasifikaciji gdje postoje dvije klase teksta odnosno recenzija. U ovom slučaju to su „*Recommended*“ i „*Not recommended*“. Uz kreirani model potrebno je izraditi jednostavnu aplikaciju koja koristi prethodno navedeni model.

5.1. Programsko okruženje

Prilikom izrade praktičnog dijela korišten je *Google-ov* proizvod *Colaboratory* koji predstavlja interaktivno okruženje u vidu *Jupyer notebook-a*, a koji nam omogućavaju izvršavanje koda pisanog programskim jezikom *Python* i *Bash* naredbi. Kako *Colaboratory* instanca nije trajna, te se nakon prekida gube svi podatci, *Google Drive* je korišten kao trajno spremište. Prilikom pisanja koda korištene su sljedeće *Python* biblioteke:

1. fastai
2. fastai.text.all
3. pandas
4. google.colab
5. ipywidgets

`fastai/fastai.text.all` je biblioteka otvorenog koda koja služi za izradu i treniranje dubokih modela, a cilj joj je olakšati i ubrzati izradu i treniranje. Nudi veliki broj *high-level* komponenti te je savršena iza početnike.

`pandas` je biblioteka otvorenog koda i služi za analizu i manipulaciju podataka.

`google.colab` je biblioteka specifična za *Colaboratory*, a omogućava pristup raznim alatima

`ipywidgets` je biblioteka otvorenog koda, a omogućava primjenu HTML *widget*-a unutar *Jupyter notebook-a.*

Važno je još spomenuti kako je model rađen prema *ULMFiT* pristupu. On se temelji na korištenju prethodno treniranih univerzalnih jezičnih modela koji se prilagođavaju naknadnim treniranjem modela primjenom skupa podataka koji je bliži određenoj primjeni. Više o ovome pristupu se detaljno može pročitati na [13].

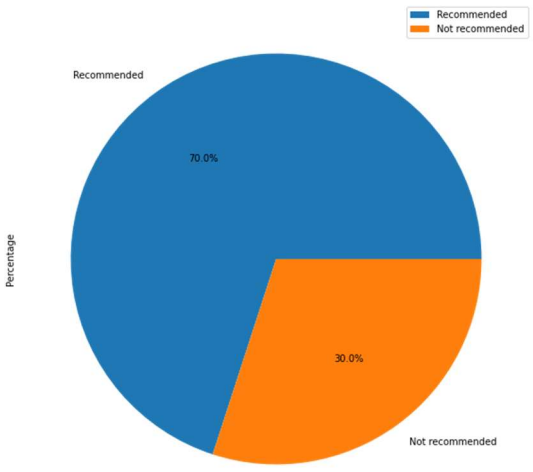
5.2. Skup podataka

Kao jedna od najvećih zajednica strojnog i dubokog učenja *Kaggle* je bio prvi izbor za odabir skupa podataka jer ima ogroman katalog skupova podataka. Odabran je *Steam Reviews DataSet* skup podataka. [14] Sadrži 8 stupaca (Tablica 5.1.) i 434891 redova. U svrhu izrade modela potrebna su nam samo 2 od 8 stupaca, a to su *recommendation* i *review.*

Tablica 5.1. Redak iz skupa podataka

date_posted	*funny*	*helpful*	*hour_played*	*is_early_access*	*recommendation*	*review*	*title*
2019-02-10	2	4	578	False	Recommended	&gt Played as German Reich&gt Declare war on Belgium&gt Can't break Belgium so go through France&gt Capitulate France in order to get to Belgium&gt Get True Blitzkrieg achievementThis game is dad	Expansion - Hearts of Iron IV: Man the Guns

Stupac *recommendation* predstavlja ciljeve odnosno klase, dok stupac review predstavlja ulazni podatak, u slučaju jezičnog modela predstavlja i ciljeve s pomakom za jednu riječ u desno. Poželjno je da je zastupljenost pojedinih klasa u skupu podataka ujednačena, u ovome slučaju



Sl. 5.1. Omjer klasa u skupu podataka

nije ujednačen, a prikazan je slikom 5.1.

5.3. Treniranje modela

Kako bi se započelo s izradom modela potrebno je ulogirati se na *Google disk* i stvoriti novi *Google Colaboratory*. Nakon uspješnog stvaranja i pokretanja *Colaboratory-a* potrebno je ažurirati verziju `fastai` biblioteke, te uvesti sve potrebne biblioteke, a koje su prethodno navedene. Kod za ažuriranje i uvoz biblioteka prikazan je slikom 5.2.

Linija ***Kod***

```
1:      !pip install fastai --upgrade
2:      from fastai import *
3:      from fastai.text.all import *
4:      import pandas as pd
5:      from google.colab import drive
```

Sl. 5.2. Ažuriranje i uvoz potrebnih biblioteka

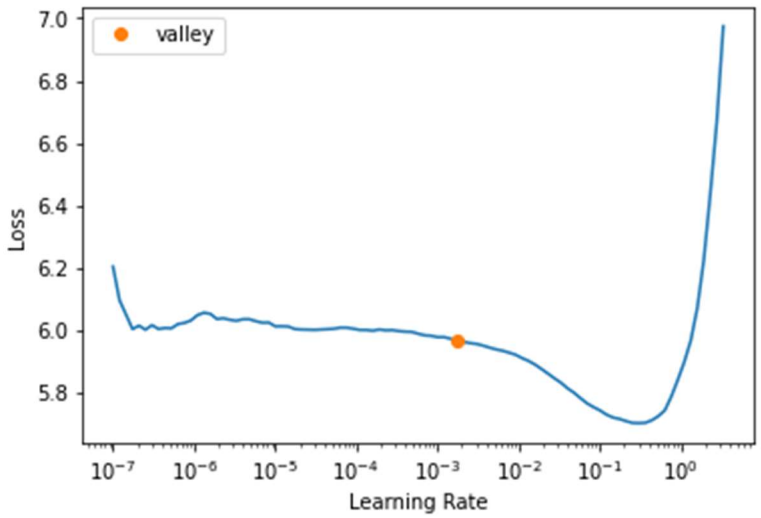
Nakon ažuriranja i uvoza potrebno je povezati *Colaboratory* s *Google Drive-om* kako bi ga se moglo koristiti kao trajno spremište. Nakon povezivanja potrebno je na *Colaboratory* instanci `mkdir` kreirati direktorij u koji će se naredbom `cp` kopirati skup podataka s *Drive-a*. Skup podataka je u zip formatu te ga je `unzip` naredbom potrebno raspakirati. Raspakirani podatci se zatim učitavaju u `DataFrame` kako bi se moglo provjeriti sadrže li stupci *recommendation i review* retke s `null` vrijednostima. Nakon provjere, svi se redci koji sadrže `null` vrijednosti odbacuju. Kako bi se podatci mogli koristiti za treniranje modela potrebno ih je dovesti u prikladan oblik. Instanciranjem `TextDataLoader` objekta podatke dovodimo u prikladan oblik tj. odrađena je tokenizacija, indeksiranje tokena, određena je veličina grupe instanci (*eng. Batch size*) od 64, duljina instance (*eng. Sequence length*) od 72, stvoren je rječnik, 20% podataka je odvojeno za validaciju i pomicanjem u desno za jedan token stvoreni su cijevi. Prilagođeni skup podataka se predaje `language_model_learner` metodi koja stvara objekt klase `Learner`. `Learner` je osnovna klasa za treniranje modela. Konstruktor učitava prethodno trenirani univerzalni jezični model i rječnik kojim je treniran, potrebno mu je predati skup podataka za prilagodbu, arhitekturu, mjerilo učinkovitosti, putanju prema podatcima i iznos snage regularizacije. Kod za prethodne radnje prikazan je slikom 5.3.

Linija Kod

```
1:    drive.mount('/content/gdrive')
2:    !mkdir /content/zavrsni/
3:    !cp /content/gdrive/MyDrive/archive.zip /content/zavrsni/
4:    !unzip -d /content/zavrsni/ /content/zavrsni/archive.zip
5:    df = pd.read_csv('/content/zavrsni/steam_reviews.csv')
6:    df.isnull().sum()
7:    df=df.dropna()
      dls_lm = TextDataLoaders.from_df(df, path=path, text_col='review', i
8:    s_lm=True)
      learn = language_model_learner(dls_lm, AWD_LSTM, metrics=accuracy, p
9:    ath=path, wd=0.1).to_fp16()
```

Sl. 5.3. Prilagodba podataka i kreiranje modela

Nakon što je kreiran `Learner` objekt, metodom `lr_find()` se pokreće probno treniranje koje vraća graf ovisnosti gubitka o stopi učenja (Slika 5.4.). Iz grafa se očita odgovarajuća vrijednost stope učenja, u ovom slučaju to je vrijednost koja se približno nalazi na najstrmijem dijelu krivulje prije njenog minimuma.



Sl. 5.4. Graf dobiven `lr_find()` metodom

Kako je `Learner` stvoren korištenjem prethodno treniranog jezičnog modela, on se nalazi u zamrznutom stanju što znači ako se započne treniranje modela, samo će se trenirati „glava“ modela tj. dio modela odgovoran za davanje predviđanja. Prema [13], poželjno je pokrenuti trening glave modela za jednu epohu prije treniranja cjelokupnog modela. Nakon treniranja od jedne epohe model je ostvario rezultate prikazane tablicom 5.2.

Tablica 5.2. Rezultati treniranja

Epoha	*Gubitak na skupu za trening*	*Gubitak na skupu za validaciju*	*Točnost*	*Vrijeme trajanja*
0	4.417914	4.160155	0.272196	20:03

Nakon što je završen trening glave jezičnog modela, trenutno stanje se pohranjuje, a nakon toga model je potrebno odmrznuti kako bi se treniralo i njegovo tijelo. Nakon odmrzavanja, poziva se metoda za pronalazak povoljne stope učenja te se pokreče treniranje modela. Pokusno treniranje modela je rađeno s manjim brojem epoha, kako bi se pronašao trenutak kod kojeg dolazi do pogoršanja točnosti modela. U ovom slučaju to je bilo nakon jedanaeste epohe. Kako metoda koristi *1cycle* pravilo, stopa učenja se mijenja svakom iteracijom i to od početne vrijednosti do predane vrijednosti stope učenja koja predstavlja maksimalnu vrijednost, te od maksimalne do krajnje vrijednosti stope učenja. Početna i krajnja vrijednost stope proizašla je od predane maksimalne vrijednosti. Nakon pokusnog treniranja pokrenuto je treniranje od jedanaest epoha, a rezultati treniranja prikazani su tablicom 5.3.

Tablica 5.3. Rezultati treniranja

Epoha	*Gubitak na skupu za trening*	*Gubitak na skupu za validaciju*	*Točnost*	*Vrijeme trajanja*
0	4.018743	3.867783	0.304312	21:04
1	3.927187	3.838181	0.306808	21:01
2	3.972200	3.823397	0.308643	21:11
3	3.911047	3.796109	0.312176	21:23
4	3.867229	3.764987	0.315473	21:12
5	3.812483	3.732502	0.318914	21:16
6	3.775292	3.693320	0.322719	21:05
7	3.755320	3.647739	0.327938	21:01
8	3.658103	3.608413	0.332747	20:58
9	3.636756	3.584918	0.335843	20:53
10	3.588591	3.580237	0.336644	20:56

Završetkom treniranja ostvarena točnost jezičnog modela iznosi 33% te se može smatrati da je jezični model prilagođen skupu podataka. Tijelo jezičnog modela se pohranjuje kako bi se u sljedećem dijelu koristilo kao temelj za klasifikator teksta. Kod za prethodne radnje prikazan je slikom 5.5.

Linija Kod

```
1:    learn.unfreeze()
2:    learn.lr_find()
3:    learn.fit_one_cycle(11, 3e-3)
4:    learn.save_encoder('finetuned_lm')
5:    !tar -zcf zavrsni_final.tar.gz /content/zavrsni/
6:    !cp /content/zavrsni_final.tar.gz /content/gdrive/MyDrive/
```

Sl. 5.5. Odmrzavanje i treniranje jezičnog modela, te pohrana tijela modela.

Nakon prilagodbe jezičnog modela prelazi se na treniranje binarnog klasifikatora, kao i kod jezičnog modela potrebno je pripremiti podatke za treniranje. Kako se koristi tijelo jezičnog modela kao baza klasifikatora prilikom pripreme podataka potrebno je predati rječnik korišten prilikom prilagodbe jezičnog modela jer u suprotnom naučeni parametri ne bi imali smisla. Nakon prilagodbe podataka stvoren je `Learner` objekt korištenjem `text_classifier_learner` metode, prilagođene za stvaranje klasifikatora teksta. Nakon stvaranja `Learner-a` potrebno je učitati tijelo jezičnog modela. Kod treniranja klasifikatora prema *ULMFiT* pristupu, koristi se postupno odmrzavanje modela i diskriminativne stope učenja. Kao što samo ime govori postupno odmrzavanje modela je postupak kod kojeg model odmrzavamo postupno, na samom početku treninga, samo je zadnji sloj odmrznut tj. sloj za klasifikaciju. Diskriminativne stope učenja predstavljaju korištenje različitih stopa učenja za različite grupe parametara modela (tijelo i glava modela). Diskriminativne stope učenja se koriste kako bi se parametri tijela modela slabije trenirali u odnosu na njegovu glavu. Kao i kod jezičnog modela prije svakog treninga potrebno je odabrati odgovarajuću stopu učenja. Treniranje je odrađeno tako da se prvo trenirala samo glava modela, zatim zadnja dva sloja, zatim zadnja tri i na kraju sva četiri sloja modela. U zadnja tri dijela korištene su diskriminativne stope učenja i to tako da je glava trenirana odabranom stopom učenja, a tijelo odabranom stopom učenja koja se umanjila 10 puta. Nakon što su svi dijelovi treninga gotovi, potrebno je izvesti model kako bi se mogao koristiti za klasifikaciju bez pripreme podataka i kreiranja `Learner` objekta. Nakon završetka treniranja klasifikator je postigao točnost od 91.5%. Rezultati treniranja prikazani su tablicom 5.4. Kod korišten za prethodno navedene radnje prikazan je slikom 5.6.

Tablica 5.4. Rezultati treniranja

Dio treninga	*Epoha*	*Gubitak na skupu za trening*	*Gubitak na skupu za validaciju*	*Točnost*	*Vrijeme trajanja*
1	0	0.267226	0.243407	0.896268	20:56
2	0	0.231004	0.222512	0.908520	24:13
3	0	0.211439	0.216557	0.914162	36:33
4	0	0.203501	0.217956	0.912443	48:39
4	1	0.202141	0.213999	0.915431	48:43

Linija Kod

```
1:     !cp /content/gdrive/MyDrive/zavrsni_final.tar.gz /content/
2:     !tar -xvf /content/zavrsni_final.tar.gz /
       dls_clas = TextDataLoaders.from_df(df, path=path, text_col='review', la
3:     bel_col='recommendation', text_vocab=dls_lm.vocab)
       learn = text_classifier_learner(dls_clas, AWD_LSTM, drop_mult=0.5, metr
4:     ics=accuracy).to_fp16()
5:     learn = learn.load_encoder('finetuned_lm')
6:     learn.lr_find()
7:     learn.fit_one_cycle(1, 1e-3)
8:     learn.lr_find()
9:     learn.freeze_to(-2)
10:    learn.fit_one_cycle(1, slice(6e-3/10,6e-3))
11:    learn.lr_find()
11:    learn.freeze_to(-3)
12:    learn.fit_one_cycle(1,slice(1e-3/10,1e-3))
13:    learn.lr_find()
14:    learn.unfreeze()
15:    learn.fit_one_cycle(2, slice(3e-4/10,3e-4))
16:    learn.export('steam_model.pkl')
```

Sl. 5.6. Kod za treniranje klasifikatora

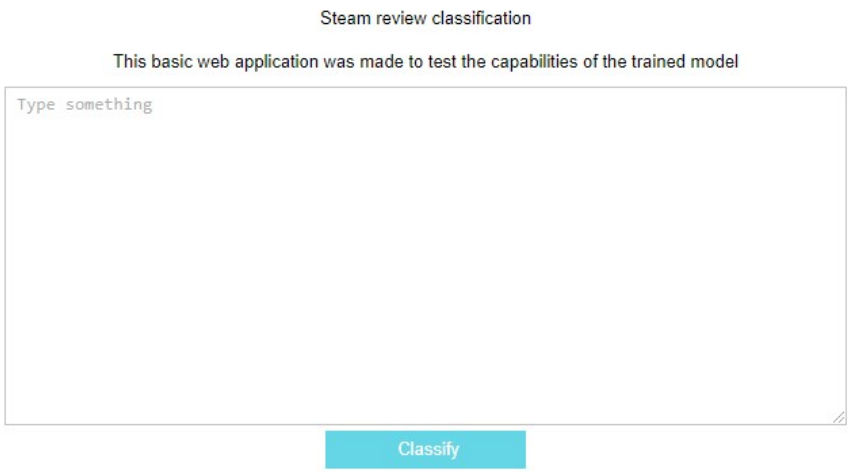
5.4. Izrada jednostavnog korisničkog sučelja

U svrhu izrade aplikacije koja bi koristila istrenirani model, korišten je *Colaboratory* jer omogućava brzu izradu jednostavnog korisničkog sučelja korištenjem `ipywidgets` biblioteke. Sučelje se sastoji od naziva, opisa, tekstnog polja i gumba. Za korištenje je potrebno unijeti tekst i kliknuti na gumb kako bi se odradila klasifikacija. Kod aplikacije prikazan je slikom 5.7., a izgled sučelja slikom 5.8.

Linija Kod

```
1:  !pip install ipywidgets
2:  !pip install fastai --upgrade
3:  import ipywidgets as widgets
4:  from fastai import *
5:  from google.colab import drive
6:  from fastai.text.all import *
7:  drive.mount('/content/gdrive/')
8:  steam_inf = load_learner('/content/gdrive/MyDrive/steam_model.pkl')
9:  user_input = widgets.Textarea(
10:     layout=widgets.Layout(width='50%',height='250px'),
11:     placeholder='Type something',
11:     disabled=False ,
12: )
13: button = widgets.Button(
14:     description='Classify',
15:     disabled=False,
16:     button_style='info',
17:     tooltip='Click me',
18: )
19: prediction_class = widgets.Label()
20: prediction_percentage = widgets.Label()
21: title = widgets.Label('Steam review classification')
    description = widgets.Label("This basic web application was made to
22:  test the capabilities of the trained model")
23: out = widgets.Output()
24: def on_button_clicked(b):
25:     with out:
            prediction,prediction_index,probabilities = steam_inf.predict(u
26: ser_input.value)
27:         prediction_class.value = f'Model prediction: {prediction}'
            prediction_percentage.value = f'Probability: {probabilities
28: [prediction_index]*100:.04f}%'
29:         user_input.value = ''
30: button.on_click(on_button_clicked)
31: box_layout=widgets.Layout(
32:     align_items='center'
33: )
    widgets.VBox(layout=box_layout, children=[title,
    description, user_input, button, out, prediction_class, prediction_perc
    entage])
34:
```

l. 5.7. Kod korisničkog sučelja



Sl. 5.8. Grafičko sučelje

Aplikacija predstavlja jednostavni *Jupyter notebook*, ali se po potrebi može pretvoriti u web aplikaciju pomoću servisa kao što je *binder,* servis za izradu web aplikacija korištenjem Jupyter notebooka.

6. ZAKLJUČAK

Od samih početaka obrada prirodnog jezika se razvijala velikom brzinom. Kroz vrijeme se susretala s nekolika podjela, ali početkom neuronskog pristupa obradi, doživjela je procvat. Kod tradicionalnog pristupa obradi prirodnog jezika veliki značaj donosile su različite metode pred obrade teksta, dok danas to nije slučaj. Korištenjem tradicionalnih metoda pred obrade kao što su korjenovanje i lematizacija gube se bitni podatci. Velik napredak u obradi ostvaren je rješenjem problema kratkotrajne memorije sekvencijalnih modela. Time je omogućena primjena prijenosnog učenja, što za posljedicu ima značajno ubrzanje postizanja rezultata. Dodatno ubrzanje se postiglo korištenjem metoda kao što su *1cycle* pravilo i diskriminativne stope učenja. Praktični dio rada odnosi se na razumijevanje prirodnog teksta tj. problemom klasifikacije teksta. Iz njega je vidljivo kako se relativno brzo mogu ostvariti dobri rezultati korištenjem prethodno navedenih metoda. Ovaj rad ujedno prikazuje i snagu zajednica otvorenog koda i otvorenih podataka, jer je većina primijenjenih biblioteka otvorenog koda, a korišteni skup podataka predstavlja otvorene podatke. Samim time ovaj rad ne bi bio moguć bez volontera koji održavaju iste. Iako je u radu ostvaren veliki postotak točnosti pri klasifikaciji teksta, bitno je spomenuti kako ostvareni rezultat nije postignut korištenjem *state-of-the-art* metodama, jer se vrlo brzo razvijaju nove metode obrade prirodnog jezika.

LITERATURA

[1] A. Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, O'Reilly Media, Inc., 2019.

[2] N. Buduma i N. Locascio, Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms, O'Reilly Media, Inc., 2017.

[3] W. S. McCulloch i W. Pitts, »A logical calculus of the ideas immanent in nervous activity,« *Bulletin of Mathematical Biophysics,* br. 5, pp. 115-133, 1943.

[4] M. Nielsen, »Neural Networks and Deep Learning,« Determination Press, 2015. [Mrežno]. Available: http://neuralnetworksanddeeplearning.com/index.html. [Pokušaj pristupa 1 6 2021].

[5] J. Howard i R. Thomas, »fast.ai,« fast.ai, 2021. [Mrežno]. Available: https://www.fast.ai/.

[6] J. Howard i S. Gugger, Deep Learning for Coders with fastai and PyTorch: AI Applications Without a PhD, O'Reilly Media, Inc., 2020.

[7] S. Raschka, »MLxtend: Providing machine learning and data science utilities and extensions to Python's scientific computing stack,« 2020. [Mrežno]. Available: http://rasbt.github.io/mlxtend/.

[8] Y. Goldberg i G. Hirst, Neural Network Methods for Natural Language Processing, Morgan & Claypool Publishers, 2017.

[9] D. Jurafsky i J. H. Martin, Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, 2020.

[10] »RegExr: Learn, Build, Test RegEx,« gskinner, 2021. [Mrežno]. Available: https://regexr.com/. [Pokušaj pristupa 3 6 2021].

[11] T. Mikolov, K. Chen, G. Corrado i J. Dean, »Efficient Estimation of Word Representations in Vector Space,« u *International Conference on Learning Representations*, Scottsdale, 2013.

[12] »Wikimedia Commons,« Wikimedia movement, [Mrežno]. Available: https://commons.wikimedia.org/wiki/Main_Page. [Pokušaj pristupa 9 6 2021].

[13] »Fastai documentation,« fastai, 2021. [Mrežno]. Available: https://docs.fast.ai/. [Pokušaj pristupa 21 7 2021].

[14] »steam-reviews-dataset,« [Mrežno]. Available: https://www.kaggle.com/luthfim/steam-reviews-dataset.

SAŽETAK

U ovom radu obrađena je podjela strojnog učenja, osnove dubokog učenja, neuroni i njihova funkcionalnost, što su neuronske mreže, kako se optimiziraju parametri modela, načini sprječavanja prenaučenosti i obrada prirodnog jezika. Kod obrade opisane su tradicionalne metode pred obrade, reprezentacija riječi i arhitekture dubokog učenja namijenjene za obradu. Navedeni su nedostatci povratnih neuronskih mreža i razvijene arhitekture koje su smanjile te nedostatke. U praktičnom dijelu rada riješen je problem razumijevanja teksta tj. klasifikacije teksta. Prilikom rješavanja problema klasifikacije korišten je *ULMFiT* pristup, koji se temelji na prijenosnom učenju, korištenju diskriminativnih stopa učenja i *1cycle* pravila. Istrenirani model postigao je točnost od 91.5%. U svrhu prikaza mogućnosti istreniranog modela napravljeno je jednostavno korisničko sučelje.

Ključne riječi:

Duboko učenje, obrada prirodnog jezika, pred obrada, razumijevanje prirodnog jezika, strojno učenje.

NATURAL LANGUAGE UNDERSTANDING WITH HELP OF DEEP LEARNING

Abstract

This paper deals with the division of machine learning, the basics of deep learning, neurons and their functionality, neural networks, how model parameters are optimized, how to prevent overfitting and natural language processing. Within natural language this paper delas with pre-processing, word representation and deep learning architectures for natural language processing. The shortcomings of recurrent neural networks and developed architecture that reduce these shortcomings are presented. Practical part of the paper solves the problem of understanding text i.e. text classification. Problem was solved with ULMFiT approach, based on transfer learning, usage of discriminative learning rates and 1cycle policy. The trained model achieved 91.5% accuracy. In order to show the capability of trained model, a simple user interface was made.

Keywords:

Deep learning, natural language processing, pre-processing, natural language understanding, machine learning.

PRILOG

-Izvorni kod praktičnog dijela

-Istrenirani klasifikator

(Oba priloga se nalaze na CD/DVD-u)