Implementacija automatiziranog sustava za generalni tekstualni razgovor (eng. chatbot)

Tutiš, Iva

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Iva Tutiš

IMPLEMENTACIJA AUTOMATIZIRANOG SUSTAVA ZA GENERALNI TEKSTUALNI RAZGOVOR (ENG. CHATBOT)

Diplomski rad

Voditelj rada:
doc. dr. sc. Matej Mihelčić

Zagreb, rujan 2022.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Prvo, želim zahvaliti mentoru doc. dr. sc. Mateju Mihelčiću na povjerenju. Također, svim dosadašnjim profesorima i mentorima na stečenom znanju. Posebno se zahvaljujem mojoj obitelji i prijateljima koji su velika podrška. Za kraj, rekurzivno ću spomenuti i one koji su pomogli njima da pomognu meni.
Hvala vam!

Sadržaj

Uvod

Od samog začetka izrade umjetno inteligentnih sustava, jedan od osnovnih izazova izrade takvog programa je izrada automatskog sustava koji može komunicirati sa korisnikom koristeći prirodan ljudski jezik.

Automatizirani sustavi za tekstualni razgovor (eng. *chatbots*) su klasa programa koji su nastali s tim ciljem - te danas, koristeći sve naprednije algoritme iza kulisa korisničkog sučelja, ispunjavaju niz industrijskih i društvenih potreba.

Cilj ovog rada je predstaviti pregled postojećih chatbot programa, implementirati dva chatbot programa i diskutirati njihove preformanse.

U prvom poglavlju ćemo definirati pojam chatbota, ukratko opisati motivaciju za izradu chatbot vrste programa, opisati par vrsta chatbot programa nastalih da ispune različite svrhe, te kratko promotriti povijesni razvoj chatbot programa.

U drugom poglavlju ćemo opisati implementaciju chatbot programa Amber, opisati tehnologije korištene pri izradi istog, te opisati arhitekturu i komponente chatbot programa.

U trećem poglavlju ćemo opisati implementaciju chatbot programa Bernard, opisati tehnologije korištene pri izradi istog, te opisati arhitekturu modela i konzolne aplikacije chatbota.

U četvrtom i završnom poglavlju ćemo uz prilagodbe postojećih mjera, promotriti preformanse chatbot programa Amber i Bernard sa obzirom na njihove predviđene namjene.

Poglavlje 1

Chatbot

1.1 Definicija

Chatbot (skraćenica koja dolazi od engleskog naziva *chatterbot*) je računalni program koji simulira sugovornika u razgovoru sa ljudskim korisnikom (vidi [1]).

Komunikacija sa ljudskim korisnikom se provodi preko konverzacijskog sučelja, koje može biti teksualnog tipa, npr. preko tekstualnih poruka, ili pak glasovnog tipa, koristeći tehnologije pretvorbe teksta u govor i obrnuto. Vrsta korisničkog sučelja chatbota ovisi o potrebama korisnika, dostupnim resursima razvojnih programera i predviđenoj svrsi programa. U opsegu ovog rada ćemo smatrati da je korisničko sučelje chatbota tekstualnog tipa, osim na mjestima gdje je u tekstu ekspilicitno drugačije navedeno.

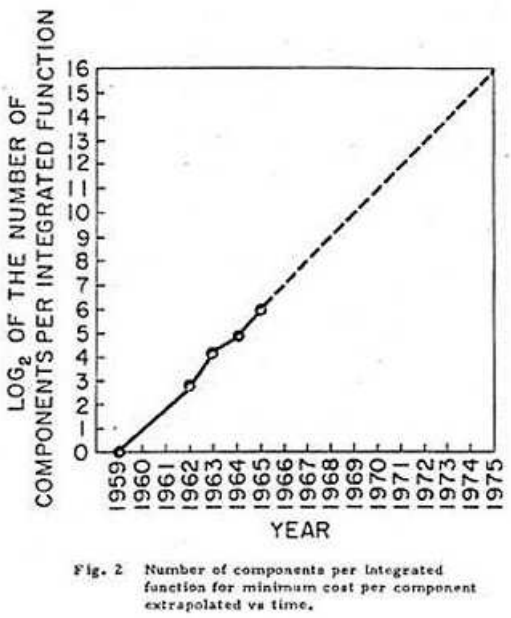
Hrvatski jezik trenutačno ne posjeduje izvornu riječ za opis pojma chatbot, iako je isti najtočnije prevodiv kao *automatizirani sustav za generalni razgovor*. Radi sročenosti i lakšeg razumijevanja daljnjeg teksta za opis takvog računalnog programa ćemo zato koristiti izraz chatbot.

Bitno je definirati i najčešću metodu korištenu za recenziranje chatbot programa - *Turingov test*. Turingov test mjeri sposobnost chatbot programa da imitira ljudskog sugovornika, to jest mjeri njegovu sposobnost da ga sugovornik percipira kao ljudsko biće.

Računalni program prolazi Turingov test ukoliko u razgovoru koji se odvija u stvarnom vremenu prosječni ljudski subjekt nije sposoban procijeniti sa značajnom točnošću je li njegov sugovornik drugi čovjek ili računalni program. Kako bi se osigurala anonimnost tipa sugovornika, razgovor koji Turing predlaže se provodi u tekstualnom (chat) obliku, koristeći neku vrstu računalnog sučelja.

1.2 Motivacija za izradu chatbota

U popularno-znanstvenoj literaturi se pri aproksimiranju brzine razvoja tehnologije često poziva na takozvani *Mooreov zakon*, što je ustvari opažanje Gordona E. Moore-a da se broj tranzistora u gustom integriranom krugu otprilike svake dvije godine udvostruči.



Slika 1.1: originalan Mooreov graf iz 1965. godine (vidi [23])

To opažanje Gordona E. Moore-a, koje je od 1965. godine mnogo godina smatrano neformalnim pravilom (iako u je stvarnosti došlo do „usporavanja" rasta zbog izrazito malih dimenzija današnjih komponenti) se često interpretira kao analogon za brzine porasta moći procesora.

Rast procesorske moći i porast generalne kupovne moći javnosti dovodi do velikog porasta potrošnje u tehnološkom sektoru. Sa sve većom dostupnosti pametnih telefona ili računala, te mogućnošću spajanja na internet, mnoge usluge postaju digitalizirane.

Utjecaj tehnologije je tako naprimjer vidljiv u trgovačkom sektoru, pri organizaciji javnog prometa i državnih struktura, te prilagođenim uslugama bankarstva.

Prosječan radnik tome primarno svjedoči po potrebi za stalnim korištenjem pametnih telefona i računala tokom svog radnog dana. Štoviše, raširenost mrežnih tehnologija je dovela i do sve poželjnijeg informiranja zaposlenika o događajima na radnom mjestu i industriji u realnom vremenu.

U privatnom životu u razvijenim zemljama posjedovanje i korištenje sve dostupnije tehnologije također izvršava dramatičan utjecaj. Uz sada već „osnovnu" razinu komunikacije koju osobni mobilni telefoni omogućavaju (pozivi i tekstualne poruke sa ostalim korisnicima), pojava pametnih telefona veće procesorske moći omogućava korištenje te tehnologije u nove svrhe: osobno bankarstvo, planiranje vremena, socijalne mreže i pristup internetu. Ovime se velik dio aktivnosti koje je osoba prije provodila uživo, na papiru ili u poslovnici „preselila" u digitalan prostor.

Taj pomak odražavaju i podaci - većina punoljetnih osoba u SAD-u u 2022. godini provodi 5 do 6 sati dnevno koristeći svoj mobitel (vidi [24]).

1.2.1 Chatbot kao inteligentni osobni asistent

Uz povećanje količine informacija sa kojima prosječna osoba današnjice raspolaže, te broj digitaliziranih usluga koje se nude kroz pametne uređaje, raste potreba edukacije korisnika o načinu korištenja istih. To pogotovo predstavlja problem kod starije populacije koja generalno sporije usvaja nova znanja i u prosjeku je manje motivirana za usvajanje istih, te kod socijalno - ekonomski ugroženih skupina (vidi [25]).

Dakle, javlja se potreba za jednostavnijim funkcionalnim pristupom već postojećim tehnologijama (telekomunikacijske usluge, kalendar, kalkulator, etc,) i pretraživanju interneta koji bi bio intuitivno razumljiv skupinama bez prethodne tehnološke pozadine.

Programi koji uz uporabu prirodnog jezika služe kao „inteligentni osobni asistenti" (eng. *„intelligent personal assistants")* se posljedično pojavljaju na tržištu da bi odgovorili na takvu potražnju.

Inteligentni osobni asistenti u pravilu mogu integrirati informacije koristeći brojne podatke prikupljene uz pomoć senzora pametnog telefona ili računala, kao što su lokacija, vrijeme, pokretnost, smjer i duljina pogleda na ekran, dodirivanje ekrana uređaja sa zaslonom na dodir, itd.

Uz današnju raširenu povezanost Internetom, inteligentni osobni asistent može pretraživati i preporučivati razne vrste sadržaja, od glazbenih i filmskih do kalendara i osobnih profila drugih korisnika na socijalnim platformama. Kao rezultat, inteligentni osobni asistenti pružaju širok raspon usluga, te se komunikacija s korisnikom odvija na ugodan i razumljiv način preko audio sučelja, bez obzira na tehnološku pismenost ili dob korisnika.

Danas brojni inteligentni osobni asistenti poput *Apple*-ovog virtualnog asistenta *Siri* ili *Amazon*-ovog virtualnog asistenta *Alexa* tako nude usluge osobnog tajništva, pretraživanja interneta i korištenje nekih usluga dostupnih unutar pripadajućeg operacijskog sustava pametnog uređaja.

Ovakvi programi najčešće pružaju *reaktivnu* i *proaktivnu asistenciju* korisniku. Primjer reaktivne asistencije je informativni odgovor na upit o meteorološkoj prognozi za određeni

dan, dok proaktivna asistencija predstavlja informiranje korisnika bez potrebe upita, naprimjer podsjećanje na skori termin sastanka ili automatizacija rezervacije stola u restoranu.

Uz sve bolje sigurnosne standarde, takvi programi omogućavaju i automatizaciju usluga profesionalnog tajništva. Upiti na softver bi se tako sveli na običan ljudski razgovor, što proširuje tržište i omogućava ponudu usluga u skladu sa suvremenim potrebama poslovnih objekata.

1.2.2 Chatbot kao asistent za dovršavanje zadatka

Uz porast količine web usluga i digitalizacijom trgovine dolazi do potrebe za asistencijom manje tehnološki pismenim osobama pri korištenju servisa preko interneta i jednostavnijem pristupu dodatnim informacijama o proizvodu ili usluzi manje tehnološki pismenim osobama.

Uzmemo li u obzir samo kako su online trgovine (obično naziva *webshop*) najčešće otvorene dvadeset i četiri sata u danu, u kontekstu ljudskih resursa to bi diktiralo potrebu za trosmjenskim radom ljudskih zaposlenika zaduženih za ispomoć.

Mottivirani profitom i brzinom usluge, mnoge tvrtke koje pružaju slične usluge online trgovine automatiziraju asistenciju pri prodaji dobara, pružanju dodatnih informacija i obavljanju nekog zadatka.

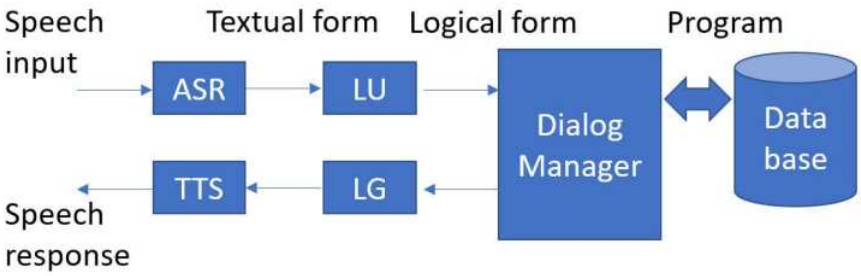
Najćešća tehnologija koja tako efektivno imitira ljudskog asistenta „sa druge strane zaslona" je upravo chatbot koji pruža uslugu automatiziranog informacijskog pulta i ispomoći kupcu pri obavljanju željenog zadatka. Takvi chatbotovi se najčešće vode pod imenom *FAQ* (eng. *„frequently-asked questions"*) chatbot ili - opširnije - konverzacijski sustav za dovršavanje zadatka (vidi [28]).

Takvi konverzacijski sustavi za dovršavanje zadataka i dalje dakako potpadaju pod definiciju chatbot programa, te često uz standardnu jezgru programa za generiranje tekstualnog odgovora na pripadni tekstualni unos, koriste i tehnologije pretvorbe teksta u govor (eng. *text-to-speech*, kratice TTS) i tehnologije pretvorbe govora u tekst (eng. *speech-to-text*, kratice STT). To im omogućava komunikaciju sa korisnikom korištenjem vokalnih komunikacijskih kanala.

Odabir unosa i odgovora programa koji je auditorne vrste tako otvara vrata većem broju potencijalnih korisnika servisa, no dolazi sa vlastitim problemima konverzije audio snimke u tekstualnu informaciju i obratno. Tom problematikom se u sklopu ovog rada nećemo baviti.

Svejedno, zgodno je promotriti arhitekturu takvih chatbota. Ona se najćešće sastoji od automatskog prepoznavača govora (eng. ASR - automatic speech recognizer), modula za razumijevanje govora (eng. SLU module - spoken language understanding module), menadžera dijaloga (eng. DM - dialogue manager), generatora prirodnog jezika (NLG - natural language generator) i pretvarača teksta u govor (eng. TTS synthesizer). Vizualna

reprezentacija ovakve arhitekture na primjeru sustava za dovršavanje zadataka je prikazana na slici 1.2.



Slika 1.2: Ilustracija arhitekture sustava za dovršavanje zadataka (vidi [28])

Razlika ovih chatbota u odnosu na chatbote za inteligentnu osobnu ispomoć je njihova znatno veća ograničenost u broju tema o kojima su sposobni razgovarati.

U praksi se takvi chatboti, iako nisu izgrađeni sa svhom prolaska Turingovog testa ili zabavne konverzacije, pokazaju kao dobra alternativa standardnom korisničkom sučelju za osobe manje računalne pismenosti. (vidi [28]) Zbog svoje jednostavne strukture i mogućnosti ponovne uporabe raznih gotovih modela, pokazali su se i kao jeftina „radna snaga" i dobra alternativa ljudskoj radnoj snazi, omogućavajući automatizaciju brojnih poslova.

1.2.3 Chatbot kao terapeutski alat

Kao jednu od glavnih potencijalnih primjena chatbot tehnologije je neizostavno još spomenuti uslugu psihološke potpore i potencijalnu terapeutsku korist chatbot tehnologije. U današnjoj medicini, dio psiholoških poremećaja se barem djelomično tretira razgovornom terapijom.

Tokom godina su se razvili mnogobrojni pristupi razgovornoj terapiji koji sa različitom uspješnošću tretiraju pojedine mentalne poremećaje, poput depresivnih poremećaja te raznih anksiozno - fobičnih poremećaja. Međutim, najveće barijere korištenju tih tretmana od strane pacijenata ostaju stigmatizirani status mentalnih poremećaja i potrebna financijska sredstva za osiguranje takvog tretmana.

Financijski gledano, pacijentima je često tretman potreban na dulji period vremena, u više navrata, kako bi pacijent dosegao stanje samoprocijenjenog boljeg mentalnog zdravlja. Razgovorna terapija je generalno učinkovitija ukoliko je njena učestalost tretmana minimalno jedanput na tjedan kroz dulji period vremena. Zasad, većina javnih sustava

koji i pružaju uslugu takvog tretmana korisnicima nisu u stanju omogućiti tako redovan tretman zbog manjka adekvatno kvalificiranog osoblja u sustavu javnog zdravstva s obzirom na potražnju. Posljedično, velik broj osoba koje imaju financijsku mogućnost da si pruže adekvatan tretman razgovorne terapije je na kraju primoran koristiti privatne usluge koje javno zdravstvo ne pokriva. S druge strane, velik broj osoba, pogotovo u ruralnim područjima ili slabijeg socio-ekonomskog statusa tu mogućnost naprosto - nema.

Stigmatizacija mentalnih poremećaja u zapadnom društvu i dalje predstavlja velik problem u svakodnevici, unatoč čestim internetskim i javnim kampanjama s ciljem destigmatizacije istih. Kao rezultat, ljudi ne traže potrebnu skrb na vrijeme, te bez okoline na koju se mogu osloniti često postaju ranjivi samoliječenju poremećaja u raspoloženju raznim štetnim narkoticima legalne i ilegalne prirode.

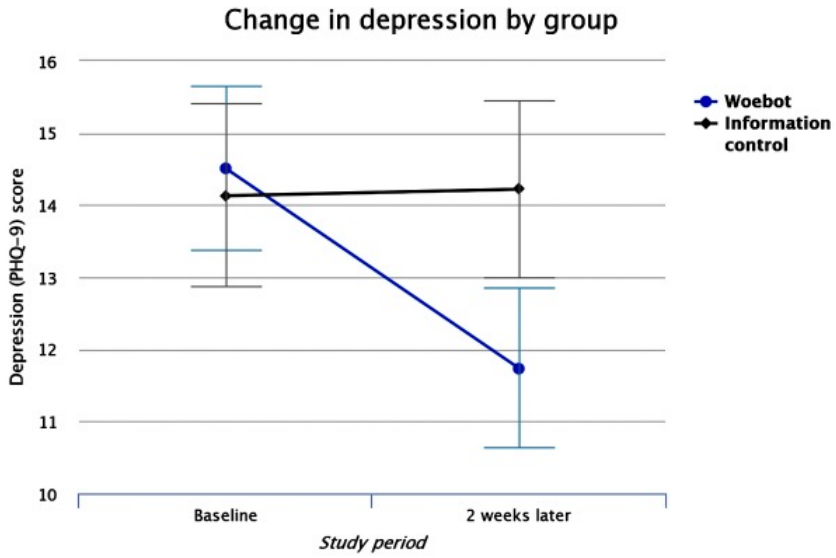
Promotrimo li samo konzumaciju alkohola u Hrvatskoj, vidimo da godine 2016. 7,7% žena i 4.7% muškaraca izjavilo kako je regularna konzumacija alkohola poznate osobe na njih jako utjecala. To je pokazano u studiji koja je obuhvaćala 1500 ispitanika u dobi od 18-64 godine, od čega je 49,9% muškaraca i 50,1% žena (vidi [29]). To nam jasno indicira kako su mentalni poremećaji, kao česti komorbiditet ovisničkim stanjima i zlouporabi opojnih tvari, ne samo zdravstveni i osobni, nego i širi, društveni problem.

Tako, u svrhu veće dostupnosti razgovorne terapije i poboljšanja mentalnog zdravlja stanovnika govornog područja, chatbot tehnologije predstavljaju potencijalnu alternativu radu sa ljudskim terapeutom. Prednost takve alternative, u svijetu koji je sve više mrežno povezan te je rasprostranjenost pametnih uređaja sve veća, je široka dostupnost takvog oblika terapije, ukoliko bi ista bila moguća. Glavne zapreke ovdje predstavljaju izrada dobrog modela i etičke barijere koje bi mogle dovesti kompaniju koja proizvodi takav softver u potencijalni gubitak (naprimjer kao rezultat tužbe radi suicida korisnika kojem je automatski sutav za razgovor neprimjereno odgovorio).

Ipak, unatoč potencijalnim industrijskim barijerama, pokušaji izrade takvog programa i automatizacije razgovorne terapije se nastavljaju: od prvih chatbot programa koji su bili djelomično motivirani takvom svrhom poput chatbota *Eliza*, do današnjih primjeraka poput chatbota *Woebot*.

Sam chatbot namijenjen u svrhu pružanja razgovorne terapije najčešće bazira svoj pristup korisniku kroz model čiji su odgovori bazirani na pojedinoj metodi razgovorne terapije: naprimjer, od dva gorespomenuta chatbota je chatbot *Eliza* bio modeliran tako da oponaša Rogerijanskog (vidi [19]) psihoterapeuta, dok chatbot *Woebot* koristi metode kognitivno bihevioralne terapije (eng. CBT), intrapersonalne terapije (eng. IPT) i dijalektičko bihevioralne terapije (eng. DBT) kako bi pomogao klijentu pri oporavku.

Istraživanja o korisnosti chatbota u terapeutske svrhe sa implementacijama nekim metodama terapije poput kognitivno bihevioralnog terapeutskog pristupa već pokazuju obećavajuće rezultate (vidi 1.3). Gornji graf prikazuje rezultate studije koja je istraživala učinkovitost administracije razgovorne terapije kroz chatbot *Woebot* u 70 subjekata, raspona dobi 18 -



Slika 1.3: Woebot-ov utjecaj na razinu depresivnog raspoloženja (vidi [36])

28 godina. Istraživanje je demonstriralo značajnu razliku razine depresivnosti u grupa koje su bile tretirane chatbotom *Woebot* u odnosu na kontrolnu grupu. Ovo istraživanje, iako sa manjim brojem subjekata, pruža nadu u mogućnost implementacije i uspješne administracije automatizirane razgovorne terapije. (vidi [36])

1.2.4 Chatbot kao prijatelj

Povijesno gledano, Aristotel još u 4. stoljeću prije nove ere definira čovjeka kao „socijalno biće" (vidi [**?**]), time aludirajući na emocionalnu potrebu čovjeka za drugim ljudskim bićima, te društvom kao kolektivom. Nastavno na antičkog filozofa, danas brojna istraživanja pokazuju negativne učinke usamljenosti na čovjekovu psihu, te benefite emocionalne potpore okoline čak i u mentalno zdravih individua (vidi [37]). Visok stupanj usamljenosti tako može pridonijeti degradaciji fizičkog i psihičkog zdravlja, pogotovo u situacijama gdje je osoba primorana u kontekstu unutarnjih ili vanjskih faktora dulje vremena provoditi sama ili u društvu osoba sa kojima nije postigla zadovoljavajuć stupanj intimnosti (vidi [37]).

Nažalost, uobičajene metode rješavanja takve problematike često nisu dostupne, pogotovo u ruralnijim područjima ili u situacijama prisilne fizičke izolacije od drugih ljudskih bića. Ciljne skupine za ovakvu uslugu chatbota nisu samo ljudi u ekstremnim psihološkim situacijama, nego i prosječna osoba koja radi privatnih, poslovnih ili zdravstvenih razloga

prolazi kroz period gdje ne može ostvariti intimnu emocionalnu konekciju s drugom ljudskom osobom.

Dakako kako bi korisnik uspostavio emocionalnu konekciju s kompjuterskim programom je potreban izrazito napredan algoritam koji može proći Turingov test u nekom obliku, te time ostvariti pozitivan utisak na korisnika dok se vanjski ili unutarnji faktori ne promijene i opet se oslobodi mogućnost zadovoljavajućih socijalnih ili intimnih odnosa.

Nadalje, ovakvi programi bi potencijalno mogli služiti kao mjere za trenutno mentalno zdravlje korisnika, te davati pravovremena upozorenja na situacije u kojima bi korisnik mogao ili trebao zatražiti profesionalnu psihološku pomoć.

Na današnjem tržištu, neki od najpoznatijih chatbota stvorenih u svrhu stvaranja i održavanja zadovoljavajućeg prijateljskog odnosa ili imitacije komunikacije sa virtualnim partnerom su chatboti *Replika* (vidi [38]) i *XiaoIce*.

Ovakav preventativan pristup protiv izolacije i samopercipirane usamljenosti korisnika bi mogao potencijalno smanjiti rastući broj ljudi koji se suočavaju s problemima usamljenosti usred slabljenja obiteljskih veza i slabljenja važnosti zajednica u urbanim sredinama.

Nadalje, pristup ovakvim programima bi mogao potencijalno znatno smanjiti broj rezultantnih mentalnih poremećaja i komorbiditeta koji su primjećeni u adolescenata koji odrastaju u obiteljima s emocionalno nedostupnim ili problematičnim skrbnicima, te im pružiti sigurno utočište i emocionalnu potporu kroz period života koji zahtijeva visok stupanj adaptacije. Još neke potencijalne korisničke skupine ovakvih usluga predstavljaju kronično bolesne osobe kojima je stupanj kretanja smanjen te su time često izolirane od velikog dijela društva, te osobe koje se ne uklapaju u svoju neposrednu okolinu radi različitih religioznih ili moralnih uvjerenja.

Zaključno, iako koncept programa - prijatelja zahtijeva prolazak Turingovog testa te dovodi do brojnih etičkih pitanja u koja nećemo ulaziti u opsegu ovog teksta, uz prikladno korištenje on omogućava olakšanje tranzicijskih životnih perioda, vježbanje prakticiranja socijalnih vještina u vrijeme izolacije te prevenciju degradacije psihofizičkog zdravlja korisnika.

1.3 Povijesni razvoj chatbot programa

Mogućnost postojanja računalnih programa sposobnih za komunikaciju sa ljudskim sugovornikom korištenjem prirodnog jezika jse prvi put razmatra u kontekstu razvoja inteligentnih programa. Posljedično, postavljaju se pitanja definiranja pojma inteligencije te percipirane inteligencije nasuprot pojma „stvarne" inteligencije objekta čiju inteligenciju pokušavamo procijeniti.

Tako 1950. godine poznati matematičar Alan Turing u svom članku „Računalno Strojarstvo i Inteligencija" (vidi [3]) razmatra pitanja poput „Mogu li računala razmišljati?", te odbacuje teološke i matematičke protuargumente, hipotetizirajući da je odgovor na pitanje

potvrdan. Kao test inteligencije programa predlaže „igru imitacije", danas široko poznatu pod nazivom *Turingov test*, koji smo opisali u 1.1.

Q: Please write me a sonnet on the subject of the Forth Bridge.

A : Count me out on this one. I never could write poetry.

Q: Add 34957 to 70764.

A: (Pause about 30 seconds and then give as answer) 105621.

Q: Do you play chess?

A: Yes.

Q: I have K at my K1, and no other pieces. You have only K at K6 and R at R1. It is your move. What do you play?

A: (After a pause of 15 seconds) R-R8 mate.

Slika 1.4: Primjer uvjerljivog razgovora prezentiran u članku Alana Turinga (vidi [3])

Drugim riječima, ukoliko program može imitirati čovjeka dovoljno dobro, program se smatra inteligentnim.

Unatoč tome, termin *chatbot*, originalno skraćenica od pojma *chatterbot*, će se povijesno prvi put pojaviti tek 1994. godine, od strane Michael Mauldina, autora chatbota *Verbot*-a (od eng.*Verbal Robot*).

1.3.1 Eliza i rani chatboti

Sa razvojem tehnologije, već 1960. - ih godina u Laboratoriju za Umjetnu Inteligenciju sveučilišta MIT (abbr. Massachusetts Institute of Technology) dolazi do razvoja prvog funkcionalnog programa za automatizirani tekstualni razgovor: chatbota *Eliza*.

Izumitelj chatbota Eliza je profesor Joseph Weizenbaum, koji je dotični chatbot program imenovao Eliza po fiktivnom liku Elize Doolittle iz kazališne predstave *Pygmalion* autora Georgea Bernarda Shawa. Program Eliza je napisan u programskom jeziku MAD-Slip te isprva objavljen 1966. godine.

Razgovor sa Elizom se provodio isključivo u tekstualnom obliku, te je sam stil razgovora Elize bio osmišljen kao imitacija Rogerijanskog psihoterapeuta, odgovarajući na korisnikov tekstualni unos uz korištenje jednostavnih ali učinkovitih metoda poput parsiranja i podudaranja uzoraka (vidi [55]).

Slika 1.5: Razgovor između chatbota Eliza i čovjeka (vidi [28])

Unatoč jednostavnom modelu i činjenici da zapravo ne razumije razgovor, chatbot Eliza je i dalje uvjerio dio korisnika u to da sa njima komunicira čovjek, a ne program.

Chatbot *Parry* je još jedan od ranijih chatbot programa, razvijen 1975. godine od strane psihijatra Kennetha Colbya na sveučilištu Stanford. Parry je bio napravljen da simulira osobu sa paranoidnom shizofrenijom (vidi [28]).

Međutim, chatbot Parry i dalje nije razumio razgovor na koji odgovara, sa modelom koji konstruira odgovor baziran na pravilima i sličan chatbotu Eliza. Prisutna su određena poboljšanja u odnosu na Elizu: bolja kontrolna struktura, mogućnosti razumijevanja jezika, i model kojim Parry može simulirati određene emocije. Parry je, kao chatbot modeliran po uzoru na paranoidnu osobu, odgovarao neprijateljski ukoliko je razina ljutnje modela bila visoka.

Idući značajniji chatbot *Thoughts* je napravljen 1988. godine, te je kasnije 1997. godine lansiran kao program dostupan kroz Internet pod uvriježenijim imenom *Jabberwacky*. Autor programa je britanski autor Rollo Carpenter koji je napravio Jabberwackya da simulira ljudskog sugovornika na „zabavan način". Kao takav, Jabberwacky je prvi chatbot koji je napravljen isključivo sa ciljem imitiranja ljudskog razgovora, i nijednom drugom

svrhom. Jabberwacky je pod aliasima *George* i *Joan* osvojio Loebnerovu nagradu, koja se dodjeljuje značajnim humanoidnim i razgovornim robotima koji su najbliže tome da prođu Turingov test, dva puta: 2005. i 2006. - te godine (vidi [47]).

Chatbot Alice, imenovan kao kratica od eng. „Artificial Linguistic Internet Computer Entity", je razvio 1995. godine američki znanstvenik Richard S. Wallace, inspiriran chatbotom Eliza. Chatbot Alice je osvojio Loebnerovu nagradu tri puta: 2000., 2001. i 2004. - te godine (vidi [28]).

Alice je kao program komunicirala s korisnikom koristeći heurističko podudaranje uzoraka s obzirom na upit poslan od strane korisnika. Alice koristi jezik AIML (eng. „Artificial Intelligence Markup Language") koji je razvijen iz jezika XML, također od strane autora chatbota Alice, doktora Richarda S. Wallace - a.

AIML kao jezik ima mogućnost korištenja tag-ova kako bi omogućio chatbot programu rekurzivan poziv algoritma za podudaranje uzoraka, što je omogućilo da sam jezik bude pojednostavljen. Međutim, zbog ograničenih mogućnosti jezika, i sam chatbot napravljen uz pomoć istog je ograničen, i ne može održati smisleni razgovor kroz dulji period vremena: iz tog razloga, Alice nije uspjela proći ultimativni Turingov test.

1.3.2 Doba interneta i razvoj chatbota asistenata

Sa razvojem Interneta, chatbot kao usluga postaje dostupan korisnicima preko web stranica i komunikacijskih servisa. To dovodi do dvije posljedice: početka uporabe chatbota kao integriranih asistenata drugih web - servisa, i znatno povećanom broju korisnika koji interagiraju sa chatbot programom koji se ne pokreće lokalno nego na udaljenom serveru koji je u stvarnosti češće većih procesorskih mogućnosti no lokalno korisnikovo računalo.

U vidu toga, modifikacija funkcije koja konstruira ili bira odgovor chatbot programa tokom korištenja samog programa postaje moguća: u praksi, to dovodi do chatbota koji sa sve većim brojem razgovora sa korisnicima daju sve bolje odgovore.

Jedan od prvih takvih chatbota integriranih u web servis je bio chatbot *SmarterChild*, razvijen 2001. godine te dostupan kroz AOL IM i MSN Messenger platforme za komunikaciju porukama (vidi [30]). SmarterChild je imao mogućnost komuniciranja sa korisnicima na zabavan i koristan način zbog svoje mogućnosti da pristupi podacima drugih online servisa.

2008. godine dolazi do objave nove iteracije chatbota Jabberwacky pod imenom *Cleverbot* od strane tvorca programa Jabberwacky autora Rollo-a Carpenter-a. Dizajn Cleverbota mu omogućava da uči od razgovora s korisnicima u realnom vremenu, te da s obzirom na podatke iz prethodnih razgovora predviđa budući smjer razgovora. (vidi [47]). Cleverbot je također bio dostupan za komunikaciju s javnosti kroz Internet, ali je njegovo web sučelje bila zasebna web stranica, bez integracije s nekakvim već postojećim web servisom.

Uz širenje sve bržeg Interneta i rast procesorske snage, dolazi do sve raširenije uporabe pametnih mobitela koji imaju mogućnost spajanja na internet. Velike kompanije to uviđaju, te koriste kao priliku uvođenjem klase chatbota marketiranih pod nazivom „inteligentni osobni asistent". Ovi programi vrše niz usluga od kojih su neke primjene već opisane u odjeljku 1.2.1.

Do prvog takvog značajnijeg programa dolazi 2010. godine, kada tvrtka Apple Inc. pušta u produkciju i prodaju chatbot inteligentnog osobnog asistenta pod imenom *Siri*, isprva kao aplikaciju unutar Apple-ovog iOS operacijskog sustava, a zatim 2011. godine i kao integrirani dio iOS sustava.(vidi [39]).

Idući bitniji inteligentni osobni asistent je *Google Now*, kojeg 2012. prezentira tvrka Google, ujedno vlasnik najraširenije tražilice na svijetu iste godine. Google Now je sposoban odgovarati na upite, davati prijedloge i pruža usluge u suradnji sa brojnim drugim web servisima. Uz tekstualno sučelje, kako bi predstavljao konkurenciju Siri koju je Apple izdao tek godinu ranije, omogućava i uporabu audio sučelja.(vidi [39]).

Tehnološki div Microsoft se ubrzo pridružio konkurenciji sa objavom svoje verzije inteligentnog osobnog asistenta *Cortana* 2014. godine, te je integrirao opcionalnu uporabu istog na najnovije verzije Widows operacijskih sustava namijenjene osobnim računalima i pametnim mobilnim telefonima. Cortana je također omogućavala audio sučelje i prepoznavala vokalne upite korisnika. (vidi [39]).

Cortana je također pružala usluge proaktivne asistencije, to jest asistencije bez prethodnog upita od strane korisnika, od kojih su neki: slanje podsjetnika, e-mail poruka i kreiranje i organizaciju listi.

2014. godine dolazi i do početka komercijalne prodaje danas možda najpoznatijeg inteligentnog osobnog asistenta: programa *Alexa* tvrtke Amazon. Unatoč činjenici da je Alexa dolazila kao integrirani softver u nizu Amazon-ovih proizvoda, ona je prvi inteligentni osobni asistent koji je bio ugrađivan i u pametne uređaje drugih kompanija, te je dostupan kao aplikacija za mobilne telefone. Alexa je poznata po svojoj agresivnoj i vrlo uspješnoj marketinškoj strategiji, te po vrlo visokom stupnju razumijevanja naredbi i upita kroz vokalno sučelje (vidi [39]).

Program Google Now je 2017. godine zamijenjen s novim Google-ovim inteligentnim osobnim asistentom prikladnijeg imena: *Google Assistant*. Google Assistant omogućava još veći broj usluga proaktivne asistencije - koncipiran velikim dijelom kao promotivna strategija za Google tražilicu - nudeći nove korisne informacije u jednostavno čitljivom formatu, pritom predviđajući trenutne i buduće potrebne korisnika koristeći korisničke podatke.

Uz povećanje broja usluga osobnih asistenata, dolazi do kretanja industrije k masovnoj proizvodnji jednostavnih chatbota koji služe kao asistenti za dovršavanje zadataka. Takva vrsta chatbota je djelomično opisana u poglavlju 1.2.2.

Kao primjer možemo uzeti zapadne lance restorana Taco Bell i Domino's, koji su već

2016. godine uveli chatbote u svoj prodajni proces prehrambernih proizvoda (vidi [18]).

1.3.3 Socijalne mreže i razvoj socijalnih chatbota

Uz razvoj i rast popularnosti socijalnih mreža, dolazi do novih verzija chatbot programa i njihove integracije u digitalni prostor, ne samo kao servisa ili asistenta, nego i kao programa koji generiraju sadržaj i preko sve popularnije i raširenije komunikacije tekstualnim porukama omogućuju usluge zabave i prijateljstva korisniku.

Za ovakvu primjenu su potrebni programi koji mogu analizirati informacije i emocije korisnika, te prikladno reagirati na određeni unos kako bi kod korisnika potaknuli pozitivnu emocionalnu reakciju.

Jedan od poznatijih chatbota ove vrste je chatbot *Mitsuku*, kasnije poznat i kao *Kuki*, koji je dizajnirao Steve Worswick. Kuki je napisana u jeziku AIML koji je razvijen još od strane kreatora chatbota *Alice* (vidi 1.3.1).

Kuki je osvojila Loebnerovu nagradu, koja se dodjeljuje značajnim humanoidnim i razgovornim robotima koji su najbliže tome da prođu Turingov test, pet puta: 2013., 2016., 2017., 2018. i 2019. - te godine (vidi [47]).

Međutim, dosadašnja istraživanja su pokazala da Kuki još nije uspjela napraviti dovoljno dobar utisak na svoje korisnike da razvije prijateljstvo sa njima. U longitudinalnoj studiji sa 118 sudionika tokom trotjednog perioda komuniciranja s chatbotom je zabilježena padajuća razina percipiranog prijateljstva i emocionalne intimnosti s programom, nakon svake interakcije (vidi [21]).

Međutim, u međuvremenu je došlo do revolucionarnog tehnološkog napretka: riječ je o programu tvrtke Microsoft, chatbotu *XiaoIce* (pinyin: Wēiruǎn Xiǎobīng).

Chatbot XiaoIce je komercijalno dostupan od svibnja 2014. godine, te je sposoban ne samo držati dulji razgovor s korisnikom, već i ostvariti interpersonalnu komunikaciju „poput prijatelja". XiaoIce to postiže iskazujući emocionalnu potporu, ohrabrujući i tako udobrovoljavajući korisnika.

XiaoIce primarno koristi kineski jezik za komunikaciju, te je zato danas manje poznat na prostorima gdje je engleski jezik dominantan, ali i dalje nosi naslov najraširenijeg socijalnog chatbota koji je trenutno u uporabi (vidi [28]).

Korisnici opetovano prijavljuju da razgovori s programom XiaoIce pozitivno utječu na njihovo raspoloženje, samoprocijenjenu razinu emocionale sigurnosti i osjećaj pripadanja društvu. U tom smislu, možemo reći da je chatbot XiaoIce možda prvi program koji na tako širokom broju korisnika pokazuje pozitivan rezultat: povećana razina povjerenja u društvo i tehnologiju i osjećaj emocionalnog blagostanja su motivirajuć razlog za daljnji razvoj i napredak u području automatskih sustava za generalni razgovor.

XiaoIce nudi i mogućnost komunikacije preko audio sučelja, te je njegov TTS (eng. text-to-speech) sistem dizajniran primarno za „zabavan" tip razgovora koji udovoljava ko-

risnikovim potrebama za prijateljskim i intimnim interakcijama.

XiaoIce u tekstualnoj i audio komunikaciji koristi indikacije emocionalnog stanja chatbota. Naprimjer, u tekstualnoj konverzaciji se to očituje pri izboru riječi ili uporabi emotikona u programovom odgovoru. Tim koji radi na razvoju TTS sustava programa XiaoIce tako radi i na novim tehnologijama koje omogućavaju iskazivanje realnih emocija u programovom audio tonu i iskazivanje ugodnijeg normalnog, opuštenog tona u razgovoru.

XiaoIce je također unikatan po tome da njegov TTS podržava miješani kinesko - engleski jezik, što ga čini prvim chatbot programom koji podržava miješani jezik.

Program XiaoIce se tokom godina također koristio za generiranje novog sadržaja, poput poezije ili pjesama. Danas, XiaoIce postaje jedan od najraširenije korištenih socijalnih chatbota s korisničkom bazom od 660 milijuna aktivnih pretplatnika.

Socijalni chatboti, namijenjeni ispunjavanju ljudskih potreba za bliskošću, uz Kinu, postaju sve popularniji i globalno, nalazeći tržište u zemljama poput Japana, Indije, SAD - a i Indonezije.

Kao završni primjer današnje uporabe chatbot tehnologije možemo spomenuti program *Rinna*, Japansku verziju programa XiaoIce. Stekla je visoku reputaciju kao chatbot koji koristi socijalne mreže. Rinna je uz aktivnost i slavu na popularnoj socijalnoj platformi *Twitter* do danas prisustvovala na jedanaest programa na televizijskim i radio stanicama, sveukupno brojeći 1193 sata aktivne interakcije sa ljudskim subjektima. Time je postigla internetsku i medijsku slavu kao zabavna figura (vidi [28]).

Slika 1.6: Razgovor korisnika sa chatbotom XiaoIce na (b) kineskom jeziku i (a) prijevodu razgovora na engleski jezik (vidi [28]).

Poglavlje 2

Amber

2.1 Osnovno o chatbotu

Chatbot *Amber* je chatbot program napisan u programskom jeziku Java i napravljen po uzoru na FAQ (eng. frequently asked questions) tip chatbota. Funkcionalno, ovaj chatbot je u svojoj namjeni najsličniji programima spomenutima u poglavlju 1.2.2 u ulozi chatbota kao asistenta za dovršavanje zadatka.

Zadatak kojeg korisnik ovdje pokušava izvršiti je tako informiranje o cijeni i funkcijama proizvoda zvanog „heksafleksagon"(vidi [54]). Amber odgovara na često postavljana pitanja te vodi korisnika kroz simulaciju kupovine. Chatbot za komunikaciju s korisnikom koristi tekstualno sučelje.

Program je napravljen koristeći kao primjer tutorial za chatbot koji koristi biblioteku Apache OpenNLP (vidi [45]).

Amber djelomično koristi pretrenirane modele koji na razini obrade prirodnog jezika nad unesenim tekstom vrše detekciju rečenica u tekstu, tokenizaciju i označavanje dijelova rečenice (eng. Part-Of-Speech tagging).

Na ove modele (vidi [7]) se poziva i službena Apacheova dokumentacija. Za lematizaciju unesenog teksta smo koristili postojeći javno dostupan lematizator (vidi [46]).

Kompilacija i testiranje koda su se izvršavali na serveru *Prosper*, namijenjenom edukaciji studenata Prirodoslovno - matematičkog fakulteta u Zagrebu.

2.2 Korištene tehnologije

Tehnološku osnovu programa Amber tvore Java, Maven i Apache OpenNLP tehnologije.

U ovom potpoglavlju ćemo ukratko spomenuti i predstaviti te tehnologije korištene pri izradi i izvršavanju programa, pritom ne raspravljajući o samoj arhitekturi i modelima unutar chatbot programa.

Za razvoj programa Amber je korištena razvojna okolina za razvoj softvera u jeziku Java naziva IntelliJ IDEA (vidi [32]).

Međutim, s obzirom da se program može uz male modifikacije izvršiti i uz pomoć drugih razvojnih okolina za razvoj softvera u jeziku Java (poput okolina NetBeans (vidi [43]) ili Eclipse IDE (vidi [22]), softver IntelliJ IDEA nećemo posebno predstavljati.

2.2.1 Java

Programski jezik Java je objektno orijentiran programski jezik visoke razine. Izrada Jave bila je motivirana idejom omogućavanja programerima da isti kod mogu izvršiti bez obzira na operacijski sustav i hardverske specifikacije uređaja. U praksi, to znači da se prevedeni Java kod može izvoditi na svim platformama koje podržavaju Javu bez potrebe za ponovnim prevođenjem (vidi [42]).

Java aplikacije obično se prevode u takozvani programski među - kod pod nazivom *bytecode* koji se može izvoditi na bilo kojem Java virtualnom stroju (JVM) bez obzira na pozadinsku arhitekturu računala.

Sintaksa Jave je slična sintaksi programskih jezika C i C++, ali dopušta manju razinu intervencije programeru no ijedan od njih. Međutim, Java je i dalje iznimno popularan programski jezik, i to velikim dijelom radi lakoće pronalazaka grešaka u kodu. Robusnost Jave se temelji na mehanizmima rane i kasne dinamičke provjere koda za vrijeme izvršavanja te stroge sintakse i njene provjere za vrijeme kompilacije.

Štoviše, Java virtualni stroj (eng. JVM) osigurava i automatizirano oslobađanje memorije od nekorištenih objekata, koje identificira sakupljač smeća (eng. garbage collector) te brisanjem „reciklira" memoriju, otklanjajući potrebu za eksplicitim navođenjem koda za uništenje svakog objekta.

2.2.2 Apache Maven

Apache Maven (vidi [6]) je alat za razumijevanje i upravljanje projektima. Baziran je na konceptu modela projektnog objekta (eng. POM) te podržava upravljanje izgradnjom projekta, dojavljivanjem i upravljanjem dokumentacijom iz centralnog repozitorija.

Glavni ciljevi alata Apache Maven su:

- olakšati proces izgradnje projekta - osigurava da se razvojni programer ne mora zamarati svim detaljima izgradnje (izgradnja znatno sporija u odnosu na Gradle, vidi [27])
- pružanje unificiranog sustava izgradnje - gradi projekt koristeći model projektnog modela i skup dodataka

- pružanje kvalitetnih informacija o projektu (npr. zapis promjena generiran iz izvornog koda, povezivanje različitih izvora, mailing liste održavane od strane projekta, ovisnosti projekta, izvješća o jediničnim testovima i njihovoj pokrivenosti)
- poticanje na korištenje dobre prakse razvoja softvera (razvojni ciklus sadrži specifikaciju, izvršavanje i izvještavanje o jediničnim testovima)

Korisne prakse kod testiranja uključuju održavanje izvornog koda za testiranje u posebnom, ali paralelnom stablu izvornih datoteka, korištenje konvencije imenovanja testova za njihovo lociranje i izvršavanje, posebno uštimavanje okruženja od strane testova umjesto posebne izgradnje softvera u svrhu testiranja, te potpora kod stvaranja konačne verzije softvera i obrade potencijalnih problema.

Maven ima dosta stroge smjernice oko strukture direktorija projekta, što ponekad može biti i mana (u nekim ekstremnim slučajevima može dovesti do potpune nekompatibilnosti projekta s alatom).

Apache Maven koristi *nacrt*, standardnu strukturu direktorija koja se uz dodatni ulaz korisnika modificira za potrebe projekta.

Ključni dokument koji definira zadatke Maven-a se zove Project Object Model (kratica POM) i unutar projekta se kreira kao datoteka pom.xml.

2.2.3 Apache OpenNLP

Biblioteka Apache OpenNLP (vidi [8]) je skup alata temeljen na strojnom učenju, namijenjen za obradu teksta prirodnog jezika.

Podržava najčešće zadatke u obradi prirodnog jezika (eng. *natural language processing*, eng. skrać. NLP), kao što su tokenizacija, segmentacija rečenica, označavanje dijela govora (eng. *part-of-speech tagging*, eng. skrać. *POS tagging*), izdvajanje imenovanih entiteta, komadanje (proces uzimanja pojedinačnih informacija i grupiranja u veće cjeline, eng. *chunking*), parsiranje i rezolucija koreferencije (proces grupiranja izraza u tekstu koji se odnose na iste entitete stvarnog svijeta, eng. *coreference resolution*). Ti su zadaci obično potrebni za stvaranje naprednijih usluga za obradu teksta. Biblioteka OpenNLP također uključuje maksimalnu entropiju i strojno učenje temeljeno na perceptronu.

Cilj OpenNLP projekta je stvoriti zreli alat za gore navedene zadatke. Drugi cilj je osigurati veliki broj unaprijed izgrađenih modela za različite jezike, kao i tekstualne resurse s komentarima iz kojih su ti modeli izvedeni.

Knjižnica Apache OpenNLP sadrži nekoliko komponenti koje omogućuju izgradnju kompletnog softverskog rješenja za obradu prirodnog jezika. Ove komponente uključuju: detektor rečenica, tokenizator, pronalazač imena, kategorizator dokumenata, označivač dijela govora, chunker, parser, rezoluciju koreferencije.

Komponente sadrže dijelove koji omogućuju izvođenje dotičnog zadatka obrade prirodnog jezika, uvježbavanje modela, a često i procjenu modela.

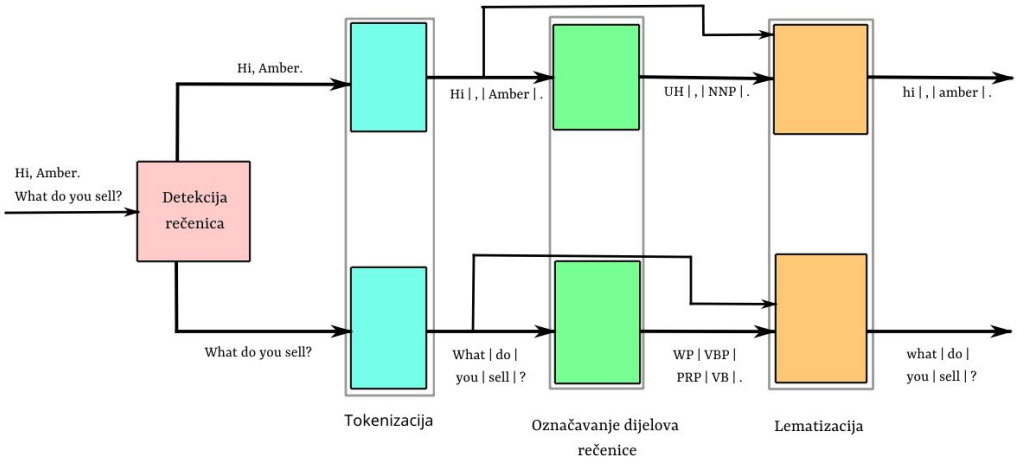
Svaki od ovih objekata dostupan je putem sučelja aplikacijskog programa (eng. skrać. API). Eksperimentiranje i obuka također su olakšani sučeljem naredbenog retka (eng. skrać. CLI).

2.3 Implementacija

U ovom potpoglavlju je opisana implementacija chatbota Amber. Program je napravljen koristeći kao primjer tutorial za chatbot koji koristi biblioteku Apache OpenNLP (vidi [45]).

U opsegu ovog potpoglavlja ćemo opisati strukturu i mehanizme generiranja odgovora na upit konzolne aplikacije chatbota *Amber*, njene sastavne komponente i proces komunikacije s korisnikom.

2.3.1 Obrada tekstualnog unosa



Slika 2.1: Shema obrade tekstualnog unosa korisnika

U ovom odjeljku je opisana metoda obrade teksta upita korisnika. Kako bi se tekst upita pravilno kategorizirao od strane kategorizatora (vidi 2.3.2.1) koji kao ulaz prima listu riječi jedne rečenice teksta unosa, potrebno je razdvojiti tekst na rečenice. Poželjno je i pretprocesirati tekst rečenice kako bi se rečenica točnije razdvojila na listu riječi.

U tu svrhu, koristiti ćemo detekciju rečenica (vidi 2.3.1.1) kako bi tekst razdvojili na rečenice, te ćemo zatim rečenicu po rečenicu redom tokenizirati (vidi 2.3.1.2) i označiti djelove rečenice (vidi 2.3.1.3). To će omogućiti lematizaciju riječi u rečenici (vidi 2.3.1.4) za točniju kasniju kategorizaciju rečenice unosa korisnika.

```
1  // Razlama string unosKorisnika na recenice
2  DetektorRecenica detektor = new DetektorRecenica();
3  String[] sentences = detektor.razlomiTesktNaRecenice(unosKorisnika);
4
5  // Program u petlji obradjuje recenicu po recenicu
6  for (String sentence : sentences) {
7
8      // Rijeci u recenici se odjeljuju jedna od druge koristeci tokenizer.
9      Tokenizer tokenizer = new Tokenizer();
10     String[] tokeni = tokenizer.tokeniziraj(sentence);
11
12     // Odvojene rijeci se oznacavaju sa POS tagovima
13     POS_Tagger posTagger = new POS_Tagger();
14     String[] posTags = posTagger.tag(tokeni);
15
16     // Svaka rijec se lematizira kako bi ju bilo lakse kategorizirati
17     Lematizator lematizator  = new Lematizator();
18     String[] lemmaTokens = lematizator.lematiziraj(tokeni, posTags);
19
20     //kategorizacija i konstrukcija odgovora chatbot programa
21     ...
22 }
```

2.3.1.1 Detekcija rečenica

Detekcija rečenica (eng. *sentence detection*) u tekstu je proces razdvajanja ulaznog teksta na rečenice.

Za detekciju rečenica koristimo lokalno definiranu klasu `DetektorRecenica` čija metoda `razlomiTesktNaRecenice()` kao unos prima tekst unosa, a kao izlaz vraća listu tekstova rečenica.

Kako bi se rečenice detektirale iz teksta koristimo resurse klase Apache OpenNLP (vidi 2.2.3): klasu `SentenceDetectorME` u koju učitavamo model `en-sent.bin` iz serije natreniranih modela koje referencira službena Apacheova dokumentacija (vidi [7]).

Klasa `SentenceDetectorME` je detektor rečenica za rastavljanje neobrađenog teksta u rečenice koji koristi model maksimalne entropije za procjenu znakova u nizu kako bi se utvrdilo označavaju li oni kraj rečenice. Natrenirani model `en-sent.bin` je treniran na

OpenNLP podacima za obuku (vidi [7]), te se pri konstrukciji instance `sentenceDetector` klase `SentenceDetectorME` učitava koristeći klasu `InputStream` za učitavanje podataka iz datoteke.

Kako bi od teksta unosa dobili listu rečenica, koristimo metodu `sentDetect()` klase `SentenceDetectorME` koja detektira rečenice u tekstu i kao izlaz vraća listu rečenica ulaznog teksta, što je ujedno izlaz metode `DetektorRecenica.razlomiTesktNaRecenice()`.

```
1 String[] recenice = sentenceDetector.sentDetect(tekst);
```

2.3.1.2 Tokenizacija rečenice

Tokenizacija rečenice (eng. *tokenisation*) je proces razlamanja teksta u jedinice zvane *tokeni*. U kontekstu ovog programa proces tokenizacije provodimo na svakoj rečenici unosa korisnika zasebno, gdje su izlazni tokeni riječi i znakovi punktuacije rečenice.

Kako bi se tekst unosa tokenizirao koristimo lokalno definiranu klasu `Tokenizer` čija metoda `tokeniziraj()` kao unos prima tekst rečenice, a kao izlaz vraća listu tokena.

Pri tokenizaciji rečenice koristimo resurse klase Apache OpenNLP (vidi 2.2.3): klasu `TokenizerME` u koju učitavamo model `en-token.bin` iz serije natreniranih modela koje referencira službena Apacheova dokumentacija (vidi [7]).

Instanca klase `TokenizerME` je tokenizer za pretvaranje neobrađenog teksta u zasebne tokene koji za donošenje odluka pri tokenizaciji koristi maksimalnu entropiju čije se funkcije temelje na disertaciji Jeffa Reynara "Topic Segmentation: Algorithms and Applications" (vidi [31]). Ovaj tokenizator zahtijeva statistički model za tokenizaciju teksta koji reproducira tokenizaciju uočenu u podacima za obuku korištenim za izradu modela. Klasa TokenizerModel enkapsulira model i pruža metode za njegovu izgradnju iz binarne reprezentacije. Natrenirani model `en-token.bin`, treniran na OpenNLP podacima za obuku (vidi [7]), prosljeđujemo konstruktoru klase `Tokenizer` pri konstrukciji instance `tokenizer` . Natrenirani model `en-token.bin` se učitava koristeći klasu `InputStream` za učitavanje podataka iz datoteke.

Pri tokenizaciji teksta koristimo metodu `tokenize()` klase `TokenizerME` koja razdvaja ulaz na tokene i kao izlaz vraća listu tokena, što je ujedno izlaz metode `Tokenizer.tokeniziraj()`.

```
1 String[] tokens = tokenizer.tokenize(recenica);
```

2.3.1.3 Označavanje dijelova rečenice

Označavanje dijelova teksta (eng. *part-of-speech tagging*) je proces označavanja riječi u ulaznom tekstu (korpusu) sa obzirom na njenu definiciju i njen kontekst. Označavanje dijelova teksta kao izlaz vraća listu oznaka riječi (eng. *tags*). U kontekstu ovog programa je ulaz označitelja dijelova teksta lista riječi rečenice unosa korisnika, a oznake dijelova

teksta mogu biti imenice, glagoli ili bilo koja od 70 drugih oznaka koje su definirane od strane klase biblioteke Apache OpenNLP (vidi 2.2.3).

Za označavanje dijelova rečenice koristimo lokalno definiranu klasu `POS_Tagger` čija metoda `tag()` kao unos prima listu riječi u rečenici, a kao izlaz vraća listu oznake dijelova rečenica. Koristimo i resurse klase Apache OpenNLP (vidi 2.2.3): klasu `POSTaggerME` u koju učitavamo model `en-pos-maxent.bin` iz serije natreniranih modela koje referencira službena Apacheova dokumentacija (vidi [7]).

Klasa `POSTaggerME` je označitelj dijelova rečenice koji koristi maksimalnu entropiju. Pokušava predvidjeti jesu li riječi imenice, glagoli ili bilo koja od 70 drugih POS oznaka, ovisno o kontekstu koji ih okružuje. U objekt klase `POSTaggerME` je pri konstrukciji instance potrebno osigurati natrenirani model označitelja dijelova rečenice za ulaz konstruktora. Koristimo natrenirani model `en-pos-maxent.bin` koji je uz metodu maksimalne entropije treniran na OpenNLP podacima (vidi [7]), te koji pri konstrukciji instance `posTagger` klase `POSTaggerME` učitavamo koristeći instancu klase `InputStream` za učitavanje podataka iz datoteke.

Kako bi od liste riječi u rečenici `tokeni` dobili listu oznaka dijelova rečenice, koristimo metodu `tag()` klase `POSTaggerME` koja označava riječi te kao izlaz vraća listu oznaka dijelova rečenice (eng. *tags*), što je ujedno izlaz metode `POS_Tagger.tag()`.

```
1 // Pronadji Part-Of-Speech (POS) tag-ove svih tokena
2 String[] posTags = posTagger.tag(tokeni);
```

2.3.1.4 Lematizacija

Lematizacija teksta (eng. *lematisation*) je tehnika normalizacije teksta koja se koristi u obradi prirodnog jezika (eng. *natural language processing*), koja prebacuje bilo koju vrstu riječi u način njenog osnovnog korijena. Lematizacija mapira različite oblike riječi na njen korijenski oblik istog značenja.

Za lematizaciju riječi u rečenici koristimo lokalno definiranu klasu `Lematizator` čija metoda `lematiziraj()` kao ulaz prima listu riječi u rečenici i odgovarajuću listu oznaka riječi, a kao izlaz vraća listu lematiziranih riječi u rečenici. Koristimo i resurse klase Apache OpenNLP (vidi 2.2.3): klasu `LemmatizerME` u koju učitavamo model `en-lemmatizer.bin` koji je dostupan na javnom repozitoriju (vidi [46]).

Klasa `LemmatizerME` je probabilistički lematizator riječi u rečenici koji pokušava predvidjeti induciranu klasu permutacije za svaku riječ ovisno o kontekstu koji je okružuje (vidi [26]). Pokušava predvidjeti jesu li riječi imenice, glagoli ili bilo koja od 70 drugih POS oznaka, ovisno o kontekstu koji ih okružuje.

U objekt klase `LemmatizerME` je pri konstrukciji instance potrebno osigurati natrenirani model lematizatora riječi za ulaz konstruktora. Koristimo natrenirani model `en-`

`lemmatizer.bin` te koji pri konstrukciji instance `lematizator` klase `LemmatizerME` učitavamo koristeći instancu klase `InputStream` za učitavanje podataka iz datoteke.

Kako bi od liste riječi u rečenici `tokeni` i odgovarajuće liste oznaka riječi `posTags` dobili listu korijena riječi u rečenici, koristimo metodu `lemmatize()` klase `LemmatizerME` koja označava riječi te kao izlaz vraća listu korijena riječi u rečenici (eng. *lemmaTokeni*), što je ujedno izlaz metode `Lematizator.lematiziraj()`.

```
1 String[] lemmaTokeni = lematizator.lemmatize(tokeni, posTags);
```

2.3.2 Konstrukcija tekstualnog odgovora

U ovom odjeljku ćemo opisati postupak kategorizacije (vidi 2.3.2.1) obrađenog upita korisnika (vidi 2.3.1.4) te odabir prikladnog odgovora s obzirom na klasu kojoj pripada uneseni upit (vidi 2.3.2.2).

2.3.2.1 Kategorizacija unosa

Kategorizacija (vidi [53]) teksta je proces automatske analize teksta nekom statističkom metodom, te dodjeljivanje skupa unaprijed definiranih oznaka ili kategorija na temelju njegovog konteksta. Chatbot Amber kategorizira rečenicu po rečenicu prethodno obrađenog teksta (vidi 2.3.1.4) koji se kategorizatoru predaju kao lista riječi u rečenici. Koristeći model *vreće riječi* (eng. *bag-of-words*), kategorizacija kao povratnu vrijednost vraća jednu kategoriju u koju je upit smješten.

Model vreće riječi je način izdvajanja značajki iz teksta za upotrebu u modeliranju, kao što su algoritmi strojnog učenja. Vreća riječi je prikaz teksta koji opisuje pojavljivanje riječi unutar dokumenta. To uključuje dvije stvari:

1. Rječnik poznatih riječi.
2. Mjeru prisutnosti poznatih riječi.

Naziv ”vreća” riječi dolazi od toga što se odbacuju sve informacije o redoslijedu ili strukturi riječi u dokumentu. Model se bavi samo time pojavljuju li se poznate riječi u dokumentu, a ne gdje u dokumentu. Za konstrukciju kategorizatora koji koristi model vreće riječi su korištene klase datoteke Apache OpenNLP (vidi 2.2.3).

Prije opisa kategorizatora, opišimo definirane kategorije u koje kategorizator može smjestiti upit. Primjeri rečenica sa označenim kategorijama nad kojima se kategorizator trenira se nalaze u datoteci `faq-kategorije.txt` . Pri izradi chatbota Amber smo definirali devet vlastitih kategorija i njihove pripadne primjere:

- Pozdrav (`greeting`)- osnovni pozdravi s kojim očekujemo da korisnici počnu razgovarati.

- Nastavi razgovor (`conversation-continue`) - riječi kao što su ”ok”, ”hmm” koje korisnik može koristiti između sadržajnijih upita u razgovoru.
- Razgovor je završen (`conversation-complete`)- riječi ili fraze koje bi korisnik mogao upotrijebiti za završetak razgovora.
- Upit o proizvodu (`product-inquiry`) - pitanja koja korisnik može postaviti kako bi saznao više o proizvodu ili njegovim značajkama.
- Upit cijene (`price-inquiry`) - pitanja koja korisnik može postaviti kako bi saznao cijenu proizvoda.
- Upit s psovkom (`swearword-block`) - upiti koji sadržavaju psovke i neprikladne riječi.
- Upiti o drugim proizvodima (`other-product-inquiry`) - pitanja koja korisnik može postaviti vezana uz to prodaje li chatbot Amber druge predmete osim heksafleksagona (vidi [54]).
- Upit o estetici proizvoda (`aesthetic-of-product`) - pitanja koja korisnik može postaviti vezana specifično uz estetiku, tj. ljepotu proizvoda.
- Upit o svrsi proizvoda (`purpose-of-product`) - pitanja koja koja korisnik može postaviti vezana specifično uz uporabnu svrhu i cilj korištenja produkta.

Klasa `Kategorizator` implementira metodu koja omogućava kategorizaciju prikladno obrađene rečenice (vidi 2.3.1.4). Privatne varijable objekta klase kategorizator je `model` , što je objekt klase kategorizatora dokumenta `DoccatModel` biblioteke Apache OpenNLP.

Kako bi program mogao obrađeni unos kategorizirati, potrebno je prije početka razgovora konstruirati instancu klase `Kategorizator`.

```
1 Kategorizator kategorizator = new Kategorizator();
```

Klasa `Kategorizator` pri konstrukciji instance kategorizatora vrši trening modela kategorizatora nad primjerima rečenica datoteke `faq-kategorije.txt` , koje učitava korištenjem klase `InputStreamFactory` biblioteke Apache OpenNLP koja omogućuje višestruka čitanja teksta sa izvora potrebne za određene vrste izgradnje modela. Kako bi omogućili čitanje podataka liniju po liniju, koristimo instancu `sampleStream` klase `ObjectStream<DocumentSample >` koja predstavlja ulazni tok podataka.

Kako bi se model trenirao, koristi se klasa `DoccatFactory` biblioteke Apache OpenNLP čija instanca `factory` s unešenim prikladnim parametrima omogućava klasifikaciju teksta u kategoriju. Pri treniranju kategorizatora koristimo uobičajene parametre koje biblioteka definira.

```
1 //stvaramo i definiramo parametre za treniranje modela kao default
     parametre
2 TrainingParameters parametres = ModelUtil.createDefaultTrainingParameters()
     ;
3 parametres.put(TrainingParameters.CUTOFF_PARAM, 0);
```

Treniranje se provodi uz pomoć klase `DocumentCategorizerME` biblioteke Apache OpenNLP čija statička metoda `train()` s odgovarajućim ulazima stvara model koji se sprema u privatnu varijablu klase `Kategorizator`.

```
1 // Treniranje modela s klasifikacijama iz teksta
2 model = DocumentCategorizerME.train("en", sampleStream, parametres, factory
     );
```

Klasa `Kategorizator` sadrži metodu koja omogućava kategorizaciju lematiziranih tokena `Kategorizator.vratiKategoriju()` koja kao unos prima izlaz lematizirane rečenice unosa (vidi 2.3.1.4).

```
1 public String vratiKategoriju(String[] finalTokens) throws IOException {
2
3     // Definiraj novi kategorizator iz modela
4     DocumentCategorizerME openNLPKategorizator = new DocumentCategorizerME(
     model);
5
6    // Nadji najvjerojatniju kategoriju
7     double[] vjerojatnostiPoKategorijama = openNLPKategorizator.categorize(
     finalTokens);
8     String kategorija = openNLPKategorizator.getBestCategory(
     vjerojatnostiPoKategorijama);
9
10    return kategorija;
11 }
```

Ukoliko je detektirana kategorija upita za završetak razgovora, mijenja se vrijednost zastavice glavnog programa `razgovorJeGotov`, koja utječe na daljnji tok razgovora (vidi 2.3.3).

```
1 // Pronalazak najbolje kategorije u koju input recenica spada
2 String category = kategorizator.vratiKategoriju(lemmaTokens);
3 // Provjeravamo je li kategorija jednaka onoj za zavrsetak razgovora
4 if ("conversation-complete".equals(category)) razgovorJeGotov = true;
```

2.3.2.2 Odabir tekstualnog odgovora s obzirom na kategoriju

Iz informacije o kategoriji u koju je upit kategoriziran (vidi 2.3.2.1), glavni program direktno odabire jedan od odgovora iz mape `pitanjeOdgovor` koja mapira klasu upita na listu predefiniranih odgovora. Mapa `pitanjeOdgovor` je definirana kao privatna statička varijabla klase `Amber`, čiju statička funkcija `main()` program izvodi (vidi 2.3.3).

```
1 \\definiraj odgovore za klasu Pozdrav
2 PitanjeOdgovor.put("greeting", Arrays.asList("Greetings,␣weary␣traveler.␣
      What␣is␣it␣that␣interests␣you?","Hello,␣my␣name␣is␣Amber.␣What␣can␣I␣
      help␣you␣with?"));
```

Koristeći svojstva klase `java.util.Random` koja omogućuje odabir nasumičnog indeksa liste odgovora za detektiranu kategoriju, program odabire jedan od prikladnih odgovora. Obzirom da kategorizator (vidi 2.3.2.1) kategorizira rečenicu po rečenicu unosa korisnika, zasebni odgovor se generira za svaku rečenicu unosa (vidi 2.3.1.1) te se odgovori na sve rečenice unosa zatim konkateniraju u tekstualni odgovor namijenjen za isporuku korisniku.

```
1 // Odgovor se determinira iz kategorije.
2 List<String> listaOdgovora = pitanjeOdgovor.get(category);
3 Random random = new Random();
4 odgovor = odgovor + "␣" + listaOdgovora.get(random.nextInt(listaOdgovora.
      size()));
```

2.3.3 Mehanizam komunikacije s korisnikom

Chatbot program se pokreće prevođenjem i pokretanjem statičke funkcije `main()` klase `Amber` u prikladnoj razvojnoj okolini.

Nakon kreiranja instance klase `Amber` i njene varijable mape pitanja i liste prikladnih odgovora `pitanjeOdgovor`, program počinje izvoditi naredbe `main()` metode. Nakon konstrukcije modela kategorizatora (vidi 2.3.2.1), program čita unos korisnika s konzole uz pomoć objekta klase `Scanner` unutar opsega `while(true)` petlje, sve dok se zastavica za završetak razgovora ne podigne.

```
1 Scanner scanner = new Scanner(System.in);
2 boolean razgovorJeGotov = false;
3 while (true) {
4     // Ucitaj unos korisnika
5     System.out.println("You:");
6     String unosKorisnika = scanner.nextLine();
7     // Obrada unosa korisnika
8     ...
```

Slika 2.2: Primjer razgovora s korisnikom

```
9       // Konstrukcija tekstualnog odgovora
10      ...
11      // Isprintaj odgovor chatbota Amber u konzolu
12      System.out.println("Amber");
13      System.out.println(odgovor);
14      //Provjeri je li razgovor gotov
15      if (razgovorJeGotov) break;
16      }
```

Na ovaj način se s programom može voditi tekstualni razgovor (eng. *chat*) dokle god upit korisnika nije detektiran kao upit kategorije završetka razgovora (vidi 2.3.2.1).

Poglavlje 3

Bernard

3.1 Osnovno o chatbotu

Chatbot *Bernard* je chatbot program za generalni tekstualni razgovor napravljen u programskom jeziku Python (vidi 3.2.1), koji za komunikaciju sa korisnikom koristi tekstualno sučelje.

Bernard za generiranje odgovora na korisnikov unos koristi model nastao kao rezultat metoda dubokog učenja.

Model se sastoji od neuralne mreže tipa transformer koja se trenira na skupu podataka „Cornell Movie-Dialogs Corpus" (vidi [16]) koji sadrži više od 220,000 dijaloga brojnih filmskih scenarija.

Sama arhitektura modela je velikim dijelom bazirana na tutorialu organizacije Tensorflow za izradu transformer modela za razumijevanje jezika (vidi [51]). Model je modificiran kako bi umjesto generiranja prevedenog teksta s portugalskog jezika na engleski jezik mogao generirati odgovor na engleskom jeziku na upit na engleskom jeziku.

Prvi verificirani chatbot s ovim pristupom je konstruirao autor Bryan M. Li, doktorand na sveučilištu u Edinbourghu (vidi [9]), koji je uz tutorial za izradu chatbota na blog stranicama Tensorflow organizacije (vidi [11]) pružio i originalni primjerak koda na korištenje pod MIT licencom (vidi [10]).

Treniranje modela za generiranje odgovora je izvršeno koristeći resurse na serveru *Prosper* namijenjenom edukaciji studenata Prirodoslovno - matematičkog fakulteta u Zagrebu.

U odnosu na izvorni članak(vidi [2]) u kojem je izložena originalna arhitektura nastala suradnjom istraživača Sveučilišta u Torontu i timova *Google Brain* i *Google Research* korporacije *Google*, u opsegu ovog rada je izvedeno treniranje više modela s različitim hiperparametrima u skladu s razmatranjima u članku (vidi 4.2). Također, broj podataka na kojima su modeli trenirani je znatno povećan u odnosu na izvorni chatbot koji koristi

transformer model sa samopažnjom (vidi [10]).

Dodatno na goreopisani model koji se koristi pri generiranju odgovora chatbot programa, u konzolnoj aplikaciji koja omogućuje komunikaciju s chatbot programom je dodana značajka detekcije želje za završetkom razgovora od strane korisnika. Detekcija korisnikove nesvjesne želje za završetkom razgovora je mjerena s obzirom na količinu angažmana pri konstrukciji odgovora.

Konzolna aplikacija dodatno detektira i korisnikovu namjeru da završi razgovor kroz promatranje uporabe pozdravnih riječi koje se koriste pri rastanku.

U kombinaciji s originalnim modelom, to čini chatbot Bernard naprednim razgovornim chatbotom namijenjenim za zabavu korisnika i socijalnu interakciju. Najsličniji je chatbotima namijenjenima da ispune prijateljsku ulogu, kao onima opisanima u 1.2.4.

3.2 Korištene tehnologije

Implementacija chatbot programa Bernard je izvršena koristeći programski jezik Python (vidi 3.2.1) uz korištenje biblioteke za strojno i duboko učenje Tensorflow (vidi 3.2.3). Bernard je razvijan koristeći i interaktivnu računalnu web - platformu Jupyter Notebook (vidi *Jupyter Notebook*).

U ovom potpoglavlju ćemo tako ukratko spomenuti i predstaviti važnije tehnologije korištene pri izradi i izvršenju programa, pritom ne raspravljajući o samom modelu chatbot programa.

Za prevođenje i modifikaciju koda u konzolnu aplikaciju je korištena razvojna okolina naziva IntelliJ IDEA (vidi [32]), korištena zajedno sa Conda menadžerom (vidi [5]) za pakete koji u sebi sadrži Anaconda distribuciju jezika Python (vidi [4]).

Međutim, s obzirom da se program može uz male modifikacije izvršiti i uz pomoć drugih razvojnih okolina za razvoj softvera u jeziku Python (poput korištenja razvojne okoline Spyder (vidi [48]) sa Conda menadžerom za pakete), softver IntelliJ IDEA nećemo posebno predstavljati.

3.2.1 Python

Python je interpretirani, objektno orijentirani programski jezik visoke razine s dinamičkom semantikom. Njegova sintaksa se lako uči zahvaljujući njenoj sličnosti prirodnom engleskom jeziku, te je zato izrazito popularan u programskim krugovima kao prvi jezik.

No, i u profesionalnim krugovima ga njegove sofisticirane ugrađene strukture podataka, u kombinaciji s dinamičkim tipiranjem i dinamičkim vezanjem, čine vrlo atraktivnim kao jezikom za razvoj aplikacija. Njegova jednostavnost ga čini i čestim odabirom jezika za povezivanje postojećih komponenti.

Budući da nema koraka kompilacije, ciklus izrade-koda-testiranja-i-otklanjanja-pogrešaka-u-kodu je nevjerojatno brz. Otklanjanje pogrešaka u Python programima jednostavno je: pogreška ili netočan unos nikada neće uzrokovati grešku segmentacije. Kada interpreter otkrije pogrešku, umjesto toga pokreće iznimku. Ako program ne uhvati iznimku, interpreter ispisuje stack trace pogreške na konzolu.

Python podržava module i pakete, što promiče modularnost programa i ponovnu upotrebu koda. Bogati izbor biblioteka, pogotovo onih koje omogućavaju manipuliranje podacima ga čine popularnim i u industrijskim, i u akademskim krugovima.

Neke od popularnijih biblioteka koje se koriste pri manipulaciji podataka, matematičkim izračunima i strojnom učenju su NumPy, Pandas, SciPy, TensorFlow, PyTorch, itd.

Završno, bitno je spomenuti i Pythonovu snažnu mogućnost vizualizacije podataka korištenjem integriranih datoteka poput Matplotlib, Plotly, Seaborn, Geoplotlib, etc.

Python interpreter i opsežna standardna biblioteka dostupni su besplatno u izvornom ili binarnom obliku za sve glavne platforme i mogu se besplatno distribuirati (vidi [44]).

3.2.2 Jupyter Notebook

Jupyter Notebook (vidi [33]) proširuje pristup temeljen na konzoli prema interaktivnom programiranju kao aplikacija koja je prikladna za prikazivanje cjelokupnog računskog procesa.

Od razvoja, dokumentacije i izvršavanja koda do komunikacije rezultata, Jupyter Notebook optimizira proizvodnju softvera, podatkovnih modela i analiza podataka.

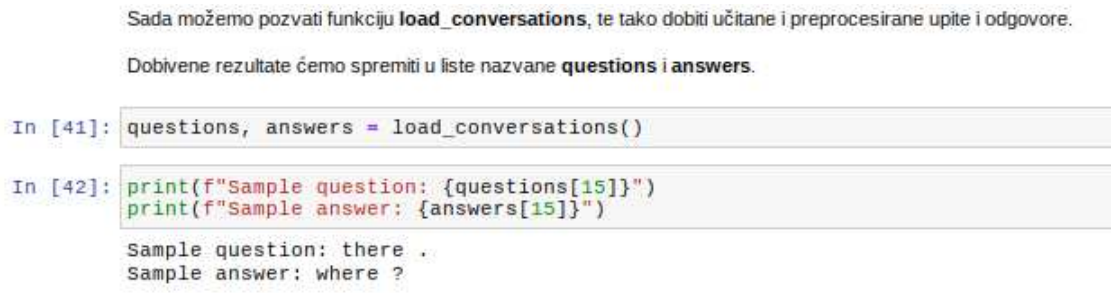
Jupyter Notebook se sastoji od dvije komponente:

- **Web aplikacija** - alat temeljen na web pregledniku koji služi kao sučelje za interaktivno stvaranje dokumenata (bilježnica). Korisničko sučelje omogućava kombiniranje programskog koda, matematike, popratnog teksta, izračuna i njihovih rezultata.
- **Notebook dokumenti (tzv. bilježnice)** - prikazuju sav sadržaj vidljiv u web aplikaciji, uključujući ulaze i izlaze izračuna, popratni tekst, matematiku, slike i bogate medijske prikaze objekata.

Jupyter bilježnice sadrže ulaze i izlaze interaktivnih session-a, kao i dodatni tekst koji prati kod, ali nije namijenjen za izvršenje. (vidi sliku 3.1)

Ovi dokumenti su interno JSON datoteke i spremaju se s ekstenzijom *.ipynb*. Budući da je JSON format običnog teksta, njima se može provoditi kontrola verzioniranja i dijeliti ih se sa suradnicima na projektu. Bilježnice se mogu i eksportirati u niz statičkih formata, uključujući: HTML (na primjer, za postove na blogu), reStructuredText, LaTeX, PDF i dijaprojekcije, a sve to putem naredbe *nbconvert*.

Jupyter Notebook server se može, pod pretpostavkom da je softver instaliran, upogoniti iz komandne linije ili terminala sa naredbom:



Slika 3.1: Isječak iz Jupyter bilježnice Bernard.ipyn

`jupyter notebook`

Ova naredba će isprintati informacije o novouključenom serveru na zaslon terminala ili komandne linije, te će otvoriti web preglednik na URL-u web aplikacije, nakon čega možemo kreirati novu Jupyter bilježnicu ili nastaviti rad na staroj.

Samo izvršavanje komandi koda u bilježnici se vrši na serveru na kojem je instaliran Jupyter Notebook. Moguće je korisničko sučelje u web pregledniku koristiti i sa drugog računala, dok smo pritom spojeni na server SSH vezom, te su na serveru izdana odgovarajuća dopuštenja.

Slika 3.2: Prikaz komunikacije sa serverom Prosper

Tokom izrade chatbota Bernard je Jupyter Notebook server bio server *Prosper*, namijenjen za edukaciju studenata Prirodoslovno - Matematičkog Fakulteta u Zagrebu. Korisničko sučelje se koristilo na udaljenom, privatnom računalu studentice nakon što je ostvarena SSH veza sa serverom Prosper. Glavna prednost ovog pristupa je mogućnost korištenja procesorskih resursa na serveru Prosper. U odnosu na Prosper, osobna računala su obično slabije procesorske moći. To će kao rezultat ubrzati vrijeme izvršavanja programa, što je izrazito bitno kod izvršavanja programa veće vremenske složenosti.

Za pokretanje Jupyter Notebook programa s lokalnom web aplikacijom i Notebook dokumentom koji se izvršava na serveru Prosper je potrebno koristiti idući postupak:

1. Potrebno je uspostaviti SSH vezu sa serverom Prosper. Operativni sustav instaliran na serveru Prosper je Ubuntu 20.04.5 LTS (vidi [14]), tako da je potrebno korisiti prikladnu metodu uspostavljanja SSH veze sa serverom. Dokle postoji više načina da se uspostavi SSH veza sa serverom Prosper od strane lokalnog računala koje koristi operativni sustav Windows (vidi [15]), privatno računalo studentice također koristi operativni sustav Ubuntu 20.04.5 LTS, tako da je za uspostavljanje SSH veze korištena lokalna aplikacija Konsole (vidi [34]), emulator terminala.

2. Kako bi se uspostavila veza sa serverom koja bi omogućila korištenje potrebnih dopuštenja, korištena je komanda:

   ```
   ssh -X -p 2122 username@prosper.math.hr -L
      8898:127.0.0.1:8898
   ```

   Nakon odašiljanja zahtjeva serveru Prosper za uspostavu SSH veze na računalnom ulazu (eng. *port*) broj 2212, server odgovara sa upitom za unos lozinke. Korisnik naziva `username` upisuje lozinku koja mu je prethodno dana od strane administratora servera, te mu se dopušta uspostava SSH veze.

3. Pretpostavimo da je SSH veza sa serverom uspostavljena. Kako bi se na serveru pokrenuo program Jupyter Notebook sa web aplikacijom dostupnom korisniku kroz lokalni web preglednik, potrebno je na serveru Prosper pokrenuti program sa naredbom:

   ```
   jupyter notebook --no-browser --port=8898
   ```

   Gornja naredba pokreće web aplikaciju Jupyter Notebook koristeći SSH vezu na računalnom ulazu broj 8898. Web aplikacija se sada može otvoriti upisivanjem ujednačenog lokatora sadržaja (eng. *URL*) u adresnu traku lokalnog web preglednika:

   ```
   http://localhost:8898/
   ```

   Web aplikacija se može pokrenuti i kopiranjem jednog od URL-a koje program Jupyter Notebook pokrenut na serveru Prosper generira u terminalu u adresnu traku lokalnog web preglednika.

Koristeći gornje upute, možemo pokrenuti program Jupyter Notebook sa udaljenog računala, te pisati i izvršavati Notebook dokumente na serveru Prosper.

3.2.3 Tensorflow

TensorFlow je besplatna biblioteka koja implementira brojne metode vezane za strojno učenje i umjetnu inteligenciju. Može se koristiti za različite zadatke, ali ima poseban fokus na treniranju i konstrukciji dubokih neuronskih mreže te inferenciji.

TensorFlow je razvio Google Brain tim za Googleovu internu upotrebu u istraživanju i proizvodnji. Prva verzija pod licencom Apache 2.0 je objavljena 2015. godine. Google je objavio ažuriranu verziju TensorFlowa pod nazivom TensorFlow 2.0 u rujnu 2019. godine. Izvorni kod TensorFlow biblioteke je javno dostupan.

TensorFlow se može koristiti u raznim programskim jezicima, uključujući Python, JavaScript, C++ i Javu. Ova fleksibilnost omogućuje niz primjena Tensorflow tehnologije u mnogim različitim područjima i integraciju sa postojećim platformama i aplikacijama.

Kako pravilno pripremljeni podaci mogu biti najvažniji čimbenik u uspjehu pri strojnom i dubokom učenju, TensorFlow nudi nekoliko podatkovnih alata koji pomažu konsolidaciji, čišćenju i predprocesiranju velikih količina podataka:

- Standardni skupovi podataka za početnu obuku i provjeru valjanosti modela
- Visoko skalabilni slijedovi algoritama i operacija (eng. *pipeline*) koji omogućavaju jednostavnije učitavanje podataka
- Slojevi za predprocesiranje podataka koji automatiziraju uobičajene ulazne transformacije podataka
- Alati za provjeru valjanosti i transformacije velikih skupova podataka

TensorFlow pruža robusne mogućnosti za implementaciju modela u mnogim okruženjima: serverima, edge uređajima, preglednicima, mobilnim uređajima, mikrokontrolerima.

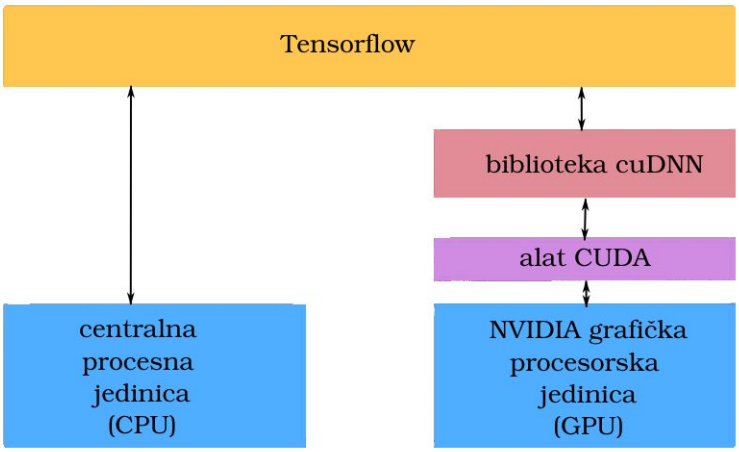
Modeli se mogu implementirati i trenirati pomoću centralnih procesnih jedinica (eng. *CPU*), grafičkih procesnih jedinica (eng. *GPU*), i programirljivih logičkih sklopova (eng. *FPGA*).

TensorFlow Serving može pokrenuti proizvodne modele koji koriste strojno učenje i na Googleovim tenzorskim procesnim jedinicama, za koje se često koristi engleska kratica *TPU* (vidi [50]).

U sklopu treniranja transformer modela kojim se koristi chatbot Bernard, korišten je server Prosper namijenjen edukaciji studenata Prirodoslovno - matematičkog fakulteta u Zagrebu.

Server Prosper uz standardnu centralnu procesnu jedinicu sadrži i grafičku procesnu jedinicu *NVIDIA Quadro RTX 5000*. Sa pravilnom instalacijom programskog alata CUDA (vidi [40]), platformi za paralelno računanje i sučelju za programiranje aplikacija koje softveru omogućavaju korištenje određenih vrsta GPU-a, i instalacijom biblioteke cuDNN

(vidi [41]), GPU-ubrzane biblioteke primitiva za duboke neuronske mreže napravljene za alat CUDA, moguće je iskoristiti procesnu moć NVIDIA grafičke procesne jedinice servera Prosper.



Slika 3.3: Interakcija Tensorflowa s NVIDIA grafičkom procesnom jedinicom

Biblioteka Tensorflow tako uz pozivanje primitiva biblioteke cuDNN može koristiti grafičku procesnu jedinicu za izvođenje izračuna (vidi sliku 3.3).

Na serveru Prosper je na daljinu izvršeno treniranje transformer modela u jeziku Python u Notebook dokumentu (vidi 3.2.2). Sada, korištenjem prikladnih Python naredbi u programskim ćelijama Notebook dokumenta je moguće treniranje modela uz pomoć grafičke procesorske jedinice servera Prosper.

Štoviše, moguće je prilagoditi program tako da se optimalno distribuirano izvršava na svim dostupnim grafičkim procesnim jedinicama, bez ručne provjere broja dostupnih jedinica. Distribuirano treniranje sa više grafičkih procesnih jedinica moguće je implementirati koristeći jednu od strategija za distribuirano treniranje koje koristi biblioteka Tensorflow (vidi [49]). Za treniranje transformer modela programa Bernard je tako korištena strategija zrcaljenja (eng. *mirrored strategy*), koja je dovoljno dobra za naše potrebe.

Strategija zrcaljenja je metoda koja stvara replike varijabli modela koje se zrcale na svim korištenim GPU-ovima.

Tijekom rada, ove zrcalne varijable se grupiraju se u varijable tipa `MirroredVariable` i održavaju sinkronizaciju s svereduktivnim algoritmima (eng. *all-reduce algorithms*). Zadani algoritam koji se koristi je onaj koji implementira *NVIDIA* kolektivna komunikacijska biblioteka (eng. skrać. NVIDIA NCCL).

Kako bi Tensorflow konstruirao strategiju zrcaljenja, dovoljno je napraviti sljedeće korake:

```
import tensorflow as tf
strategy = tf.distribute.MirroredStrategy()
```

Kako nijedan uređaj nije naveden u argumentu konstruktora strategije, koriste se svi dostupni GPU-ovi. Ako se ne pronađe nijedan GPU, koristit će se dostupan CPU.

Za korištenje strategije pri treniranju modela je dovoljno inicijalizirati i kompilirati objekt `model` unutar opsega strategije `strategy.scope()` (vidi 3.3.1.4).

3.3 Implementacija

U ovom potpoglavlju je opisana implementacija chatbota Bernard.

Opisat ćemo skup podataka na kojima je treniran model, te samu strukturu i mehanizme modela koji je treniran uz pomoć Notebook dokumenta (vidi 3.2.2). Svrha Notebook dokumenta je treniranje modela i spremanje natreniranog modela u *.h5* formatu. Format *.h5* je jedan od hijerarhijskih formata podataka (eng. skrać. *HDF*) koji se koristi za pohranu velikih količina podataka u obliku višedimenzionalnih nizova. Format se prvenstveno koristi za pohranu znanstvenih podataka koji su dobro organizirani, za brzo pronalaženje i analizu.

Modifikacijom hiperparametara modela i broja podataka u Notebook dokumentu je trenirano više natreniranih modela. To će nam dati osnovu (vidi 4) za usporedbu preformansi chatbota s obzirom na hiperparametre i broj podataka na kojima je treniran korišteni model.

Opisat ćemo i chatbot program, to jest strukturu konzolne aplikacije *BernardChatbot* napisane u jeziku Python (vidi 3.2.1), te algoritam generiranja odgovora na korisnikov unos uz mehanizam detekcije kraja razgovora (eng. *end of conversation detection*) i generiranje odgovora uz pomoć učitanog natreniranog modela.

3.3.1 Model

U ovom odjeljku ćemo opisati sve potrebno za treniranje transformer modela unutar Notebook dokumenta (vidi 3.2.2).

Opisat ćemo skup podataka korišten za treniranje modela, metodiku preprocesiranja teksta prije no što se isti preda modelu za treniranje, mehanizam pažnje korišten pri izradi modela, i opisat samu neuralnu mrežu tipa transformer koja se trenira.

Pri izradi modela koji koristi neuronsku mrežu tipa transformer je korišten tutorial sa službenih stranica (vidi [11]) organizacije Tensorflow (vidi 3.2.3) i pripadnog koda autora tutoriala pod MIT licencom (vidi [10]). Arhitektura neuralne mreže je naslijeđena iz članka

„Attention is all you need”(vidi [2]), koji je prvi predložio neuronsku mrežu tipa transformer.

Modifikacije izvršene u odnosu na originalni tutorial se sastoje od povećanja broja podataka korištenih za treniranje (vidi [16]), te treniranje više modela s varijacijom u hiperparametrima transformer (vidi 4.2) po uzoru na originalni članak (vidi [2]).

3.3.1.1 Podaci

Za treniranje modela se koristio skup podataka naziva Cornellov korpus filmskih dijaloga (vidi [17]).

Sam skup podataka je velika, metapodacima bogata zbirka izmišljenih razgovora izdvojenih iz neobrađenih filmskih scenarija (220.579 razgovora između 10.292 para filmskih likova u 617 filmova). Govornici u ovom skupu podataka su filmski likovi. Za imena govornika u ovom skupu podataka se uzima indeks govornika iz izvorne objave podataka (vidi [13]).

Skup podataka se sastoji od idućih vrsta informacija:

- informacije o filmskim likovima (ime filmskog lika, spol, ime filma, itd.)
- informacije o filmskim replikama (ime govornika, identifikacijski ključ razgovora, tekst filmske replike, itd.)
- informacije o razgovorima (ime filma, godina premijere filma, žanr, itd.)
- informacije o samom skupu podataka (ime korpusa podataka, rječnik koji preslikava `movie_idx` na URL s kojeg su dohvaćeni sirovi izvori)

Potpun popis svih stavki skupa podataka se može pronaći u službenoj dokumentaciji (vidi [17]).

Sam skup podataka se fizički sastoji od dvije datoteke:

- `movie_conversations.txt` - datoteka koja sadrži listu identifikacijskih ključeva (ID) razgovora i povezane podatke (vidi sliku 3.4)

```
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L194', 'L195', 'L196', 'L197']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L198', 'L199']
u0 +++$+++ u2 +++$+++ m0 +++$+++ ['L200', 'L201', 'L202', 'L203']
```

Slika 3.4: Isječak tekstualne datoteke filmskih razgovora

- `movie_lines.txt` - datoteka koja sadrži ime govornika i tekst filmske replike asocirane s nekim identifikacijskim ključem (vidi sliku (3.5)

```
L1045 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ They do not!
L1044 +++$+++ u2 +++$+++ m0 +++$+++ CAMERON +++$+++ They do to!
L985 +++$+++ u0 +++$+++ m0 +++$+++ BIANCA +++$+++ I hope so.
```

Slika 3.5: Isječak tekstualne datoteke filmskih replika

Kako bi se skup podataka učitao, koristi se metoda biblioteke Tensorflow (vidi 3.2.3) `tf.keras.utils.get_file()` , koja učitava datoteku sa zadanog URL-a skupa podataka.

3.3.1.2 Pretprocesiranje teksta

Kako bi tekstualni ulaz modela mogli pravilno obraditi, potrebno ga je pretprocesirati.

Kako bi ograničili količinu primjera (eng. *sample*) koji ćemo nakon pretprocesiranja koristiti za treniranje uvodimo varijablu $max_{samples}$, koja predstavlja maksimalan broj rečenica koje ćemo pretprocesirati. Također, kao dodatno ograničenje uvodimo maksimalnu dopuštenu duljinu tokenizirane rečenice max_{length}.

Funkcija `preprocess_sentence(sentence)` osigurava pravilno pretprocesiranje rečenice prirodnog jezika. Ona uz pomoću operacije za zamjenu teksta koji se podudara s regularnim izrazom `re.sub()` izvršava iduće korake:

1. Stvara razmak između riječi i interpunkcije iza nje,
   npr. ”he is smart.” pretvara u ”he is smart .”.
2. Produljenje skraćenica (kontrakcija) iz engleskog jezika,
   npr. ”i’m” pretvara u ”i am”.
3. Mijenja sve znakove koji nisu u skupu (a-z, A-Z, ”.”, ”?”, ”!”, ”,”) sa znakom razmaka.

Kako bi pak pravilno učitali podatke za treniranje, potrebno ih je učitati kao parove filmskih replika, gdje jedna odgovara na drugu u razgovoru. Takve parove dijelimo na upit (eng. question) i odgovor (eng. answer).

Pri treniranju na prijenavedenim podacima (vidi 3.3.1.1) je potrebno izvesti i dodatno prethodno pretprocesiranje podataka: kao što je vidljivo iz slika 3.4 i 3.5, u dokumentima skupa podataka je, uz linijski prikaz podatka, niz znakova „++$++” korišten kao separator podataka u linijama dokumenta.

Kako bi dodatno pretprocesirali skup podataka, definiramo funkciju `load_conversations()`, koja sadrži iduće korake:

1. Konstruiraj riječnik sa ključem (ID) filmske replike na vrijednost teksta filmske replike.
   Iz svake linije teksta učitane datoteke `movie_lines.txt` izdvoji informacije na idući način:
   1.1. Makni ” \ *n*” znak za idući red iz teksta.
   1.2. Razdvoji svaku liniju teksta sa obzirom na separator ” „++$++”.
   1.3. Stvori rječnik iz podataka.
2. Izdvoji parove filmskih replika na popis pitanja (eng. questions) i odgovora (eng. answers).
   Iz svake linije teksta učitane datoteke `movie_conversations.txt` izdvoji informacije o razgovoru na idući način:
   2.1. Makni ” \ *n*” znak za idući red iz teksta.
   2.2. Razdvoji svaku liniju teksta sa obzirom na separator ” „++$++”.
   2.3. Iz linije teksta konstruiraj razgovor kao listu identifikacijskih ključeva filmskih replika.
   2.4. Za svaki ključ filmske replike u razgovoru:
      2.4.1. Dohvati tekst pitanja i odgovora iz rječnika po ključu filmske replike.
      2.4.2. Preprocesiraj tekst filmske replike koristeći funkciju `preprocess_sentence()`
      2.4.3. Kada je dohvaćen maksimalan definiran broj parova filmskih replika na kojima će se model trenirati, stani.
3. Vrati skupove pitanja i odgovora.

Nakon pretprocesiranja teksta, vrši se tokenizacija teksta. *Tokenizacija teksta* (eng. *tokenisation*) je proces razlamanja teksta u jedinice zvane *tokeni*. Tokeni su obično riječi, brojevi i/ili interpunkcijski znakovi. Za tokenizaciju se koristi klasa `tfds.deprecated.text.SubwordTextEncoder`, koja preslikava jedinicu ulaznog teksta u identifikacijski ključ jedinice teksta (token), gdje su jedinice ulaznog teksta riječi, brojevi i interpunkcijski znakovi.

U kontekstu ovog programa ulaz tokenizatora su tekstovi iz liste `questions` i `answers`, dok su izlazni podaci liste numeričkih vrijednosti (tokena). Izlazna lista vrijednosti se zatim sprema na poziciju ulaznog teksta odgovarajućeg teksta u listi `questions` ili `answers`.

Instanca klase `tfds.deprecated.text.SubwordTextEncoder` se konstruira s obzirom na podatke koji će se tokenizirati, kreirajući poseban identifikacijski ključ za svaku jedinicu ulaznog teksta iz originalnog korpusa podataka, što čini operaciju bijektivnom. Svaka instanca klase `tfds.deprecated.text.SubwordTextEncoder` koja je konstruirana s obzirom na isti korpus kao i instanca korištena za tokenizaciju zato ima mogućnost dekodiranja liste tokena u originalni ulazni tekst uz pomoć primjene metode `decode()`.

Proces tokeniziranja unosa se sastoji od idućih koraka:

1. Konstruiramo tokenizator kao instancu klase `tfds.deprecated.text.SubwordTextEncoder` iz korpusa podataka konkateniranih lista pitanja i odgovora. Rječnik se ”trenira” na korpusu podataka `questions + answers` , te se sve jedinice teksta pohranjuju u datoteku vokabulara čija je približna veličina `target_vocab_size`.

```
tokenizer = tfds.deprecated.text.SubwordTextEncoder.
   build_from_corpus(
questions + answers, target_vocab_size=2**13
)
```

   Tokeniziranjem ulaznog teksta uz pomoć metode `tokenizer.encode()` , riječi i znakovi punktuacije sada postaju numeričke vrijednosti (tokeni), gdje je dimenzija liste tokena (vektora) broj riječi i znakova punktuacije u rečenici.

```
: print(f"Sample sentence: {questions[20]}")
  print(f"Tokenized sentence: {tokenizer.encode(questions[20])}")

Sample sentence: i really , really , really wanna go , but i cannot . not unless my sister goes .
Tokenized sentence: [4, 303, 2, 303, 2, 150, 397, 180, 2, 41, 4, 550, 3, 11, 905, 32, 1455, 3256, 1]
```

Slika 3.6: Primjer tokenizacije rečenice

2. Tokeniziramo svaku rečenicu iz lista tekstova `questions` i `answers`. Ovaj proces provodi funkcija `tokenize_and_filter(questions, answers)`. Svakom primjeru (vektoru, to jest listi numeričkih vrijednosti) se dodaju početni i završni token, te se tokenizirani primjer interakcije odbacuje ukoliko je tokenizirana rečenica pitanja ili odgovora s početnim i završnim tokenom dulja od max_{length}.

```
questions, answers = tokenize_and_filter(questions, answers)
```

Podaci spremljeni u listama `questions` i `answers` su sada adekvatno pretprocesirani, te se kao liste vektora numeričkih vrijednosti predaju neuronskoj mreži transformer (vidi 3.3.1.3).

```
print(f"Vektor odgovora:{questions[20]}")

Vektor odgovora:[8245 1181 7203   24   55    3    4  339   10 1423   16  941  671 2130
  252   20   66   16   88 2577  724    2   58    4   25  330  194   54
    9   11 1135 2961 8021  324 1090 2966 2534  615    1 8246]
```

Slika 3.7: Podatkovni vektor nakon završne obrade

3.3.1.3 Arhitektura modela

U ovom odijelju ćemo opisati neuronsku mrežu tipa transformer (vidi [2]) u arhitekturi kakva je korištena za izradu chatbot programa (vidi [11]).

Opisat ćemo pojmove neuronske mreže, dekodera i enkodera, te promotriti samu arhitekturu transformera, mehanizme pažnje i samopažnje, značajke enkodera i dekodera te treniranje samog modela.

3.3.1.3.1 Osnove neuronskih mreža

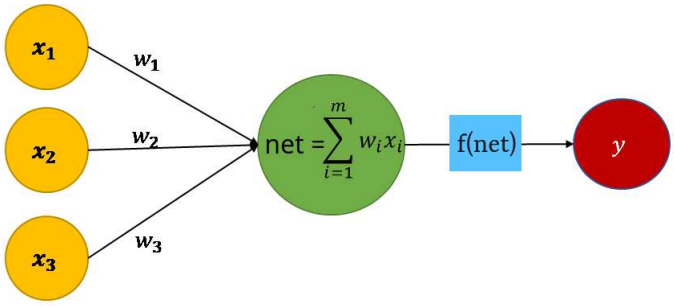
Neuronske mreže se kao pojam dijele na dvije kategorije: umjetne i biološke neuronske mreže. Primjer biološke neuronske mreže je živčani sustav čovjeka, čija je osnovna gradbena jedinica neuron.

Umjetne neuronske mreže (eng. *artificial neural networks*) (vidi [53]) su pak skup algoritama čiji je najčešći cilj prepoznavanje pozadinskih odnosa u podacima na kojima se algoritam provodi. Neuronske mreže (riječ „umjetne" se u literaturi često izostavlja) po svojoj strukturi, funkciji i obradi informacija su u mnogočemu analogne biološkim neuronskim mrežama. Algoritmi neuronskih mreža, nakon provedenog procesa „učenja" na ograničenom skupu primjera, mogu s velikom uspješnoću obaviti niz kompliciranih zadataka: prepoznati oblik na slici i opisati ga, generirati tekst, pretvoriti govor u tekst, itd.

Radi slične strukture, osnovna jedinica umjetne neuronske mreže iz područja biologije baštini naziv **neuron**. Svaki neuron posjeduje lokalnu memoriju u kojoj pamti podatke koje obrađuje. Te jedinice su povezane komunikacijskim kanalima (vezama). Podaci se ovim kanalima obično razmjenjuju na numerički način, gdje pritom neuron obrađuje samo svoj lokalni podatak i ulaze koje prima preko veza s okolnim neuronima. Umjetni neuroni imaju jednostavnu strukturu i slične funkcije kao i biološki neuroni. Tijelo neurona se u literaturi često naziva čvor (eng *node*) ili jedinica. Elementi neurona prikazanog na slici 3.8 su:

- $x_{1..n}$ – ulazni podaci (numeričke vrijednosti)
- $w_{1..n}$ – težinski koeficijenti (opisuju jakost veze)
- $f()$ – aktivacijska funkcija

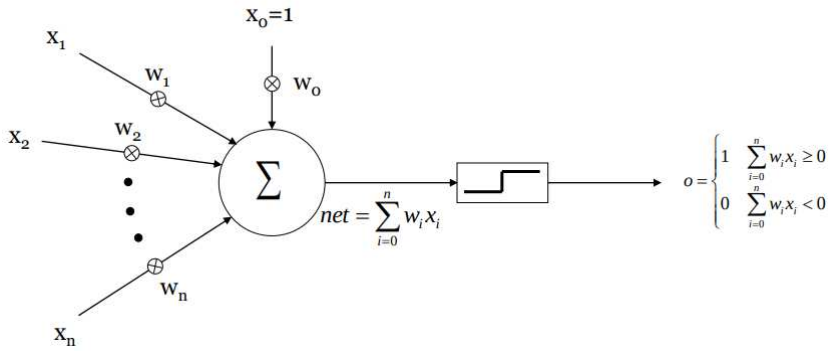
- y – izlazni podatak



Slika 3.8: Primjer neurona

Perceptron je osnovna jedinica neuronske mreže, odnosno umjetni neuron koji ima specifičnu funkciju praga (eng.*step function*) za aktivaciju, definiranu kao:

$$f\left(\sum_{i=1}^{n} w_i x_i\right) := \begin{cases} 1, \sum_{i=1}^{n} w_i x_i \geq 0 \\ 0, \text{inače} \end{cases}$$



Slika 3.9: Osnovna gradivna jedinica neuronske mreže - perceptron. Ukoliko se koristi individualno, može se smatrati najjednostavnijom neuronskom mrežom (vidi [53]).

Arhitekturu neuronske mreže predstavlja shema neurona i njihovih veza. Po arhitekturi, neuronske mreže se razlikuju prema broju neuronskih slojeva. Obično svaki sloj prima ulaze iz prethodnog sloja, a svoje izlaze šalje narednom sloju. Prvi sloj se obično naziva ulazni, posljednji je izlazni, a ostali slojevi se obično nazivaju skrivenim slojevima.

Jedna od najčešćih arhitektura neuronskih mreža je mreža sa tri sloja. Prvi sloj (ulazni) je jedini sloj koji prima signale iz okruženja. Prvi sloj prenosi signale sljedećem sloju (skriveni sloj) koji obrađuje ove podatke i izdvaja značajke i informacijske strukture iz primljenih ulaznih podataka. Podaci koji se smatraju važnima se upućuju posljednjem, izlaznom sloju mreže. Na izlazima neurona trećeg sloja se tako dobivaju konačni rezultati obrade.

Većina neuronskih mreža proces učenja obavlja po nekom pravilu, te se kao rezultat težinski koeficijenti veza između neurona prilagođavaju na osnovi ulaznih podataka. Drugim riječima, neuralne mreže „uče" preko primjera, i posjeduju sposobnost generalizacije poslije izlaganja dovoljnom broju primjera - proces zvan **treniranje**. Najčešće korišten algoritam za učenje je *backpropagation*, uspoređuje izraz neuronske mreže sa željenim iznosom izlaza te zatim, koristeći **račun greške** za svaki čvor u mreži, neuronska mreža podešava težine veza prema vrijednostima greške dodijeljene svakom čvoru.

Nakon modifikacije parametara, neuronska mreža dobiva novi ulaz, te proces treniranja staje tek kada je neuronska mreža natrenirana na dovoljnom broju podataka ili sa dovoljnom točnosti. Složenije neuronske mreže mogu imati i više skrivenih slojeva, povratne veze i elemente koji su dizajnirani da omoguće što efikasnije odvajanje važnih osobina ili shema sa ulaznog sloja.

3.3.1.3.2 Neuronska mreža transformer

Transformer (vidi [2]) je vrsta neuronske mreže sa osnovnom strukturom koja se sastoji od enkoder-dekoder modela (vidi 3.3.1.3.4), s dodanim završnim linearnim slojem.

Neuronske mreže uče putem brojeva, tako da za treniranje transformer mreže za generiranje tekst, svaka riječ se mora preslikati u vektor s kontinuiranim vrijednostima koje predstavljaju tu riječ (vidi 3.3.1.2).

Hiperparametre transformer modela predstavljaju zadani podaci čijim mijenjanjem se mijenja broj slojeva i dimenzije ulaza i izlaza komponenti neuronske mreže, gdje pritom odnosi komponenti ostaju isti. Mijenjanjem hiperparametara se mijenja broj čvorova i veza unutar same neuronske mreže, što obično ima utjecaja i na rezultat. Hiperparametri neuronske mreže transformer su:

- d_{model} - Dimenzija modela, ujedno dimenzija svih ulaza i izlaza komponenti modela: enkodera, dekodera i linearnog sloja. Također, to je i dimenzija ulaza i izlaza svih podslojeva enkodera i dekodera. Mora biti pozitivan cijeli broj budući da predstavlja dimenzionalnost izlaznog vektora.

- num_{layers} - Broj slojeva enkoder-sloj i dekoder-sloj unutar enkodera, to jest dekodera (vidi 3.3.1.3.4). Mora biti pozitivan cijeli broj.

- num_{heads} - Broj glava u svakom mehanizmu pažnje s više glava (vidi 3.3.1.3.8).

- *units* - Definira veličinu izlaza linearnih gustih (eng. *dense*) slojeva unutar enkoder-sloja i dekoder-sloja (vidi 3.3.1.3.7). Mora biti pozitivan cijeli broj budući da predstavlja dimenzionalnost izlaznog vektora.

- *dropout* - Postotak nasumično odabranih neurona koji se zanemaruju tijekom treninga (vidi 3.3.1.3.7). To znači da se njihov doprinos aktivaciji neurona u daljnjim slojevima privremeno uklanja pri prolazu naprijed, a bilo kakva ažuriranja težine se ne primjenjuju na neuron pri prolazu unatrag.

Transformer je tzv. *auto-regresivni* model: dakle predviđa jedan po jedan dio odgovora, i koristi svoj dosadašnji izlaz da odluči što dalje. Tijekom obuke transformer za chatbota koristi tzv. *forsiranje učitelja* (eng. *teacher-forcing*): prosljeđivanje pravog izlaza na sljedeći vremenski korak bez obzira na to što model predviđa u trenutnom vremenskom koraku. Dok transformer predviđa svaku riječ, mehanizam samopažnje (vidi 3.3.1.3.8) mu omogućuje da „pogleda" prethodne riječi u nizu unosa kako bi bolje predvidio sljedeću riječ.

Ulazni sloj transformera

Ulazni sloj modela transformer prima ulazne podatke modela, te je je podijeljen na dijelove pod nazivom `inputs` , koji prima listu pretprocesiranih tokeniziranih upita `questions` , i `dec_inputs` , koji prima listu pretprocesiranih tokeniziranih odgovora `answers`.

Slojevi maski

Kako bi implementirali mehanizme pažnje i samopažnje, podaci dijela ulaznog sloja transformera `inputs` se šalju sloju maske za dopunjavanje koja maskira tokenizirani unos sa nulama. Podaci iz ulaznog sloja se u implementaciji neuronske mreže predaju slojevima maski za nadopunu `enc_padding_mask` i `dec_padding_mask` čiji izlazni podaci su ujedno dio ulaznih podataka enkodera i dekodera (vidi sliku 3.10).

Kako bi dekoder za predviđanje određene riječi odgovora koristio samo prethodne poznate rezultate za riječi odgovora koje dolaze prije nje, pretprocesirani podaci odgovora iz ulaznog sloja `dec_inputs` se šalju kao ulazni podaci sloju maske gledanja unaprijed `look_ahead_mask` (vidi 3.3.1.3.3) Ulazni podaci sloja dekodera su tako i podaci iz ulaznog sloja `dec_inputs` i izlaz međusloja `look_ahead_mask` (vidi sliku 3.10).

Detaljniji opis slojeva maski nadopune i maske gledanja unaprijed je vidljiv u 3.3.1.3.3.

Enkoder i dekoder

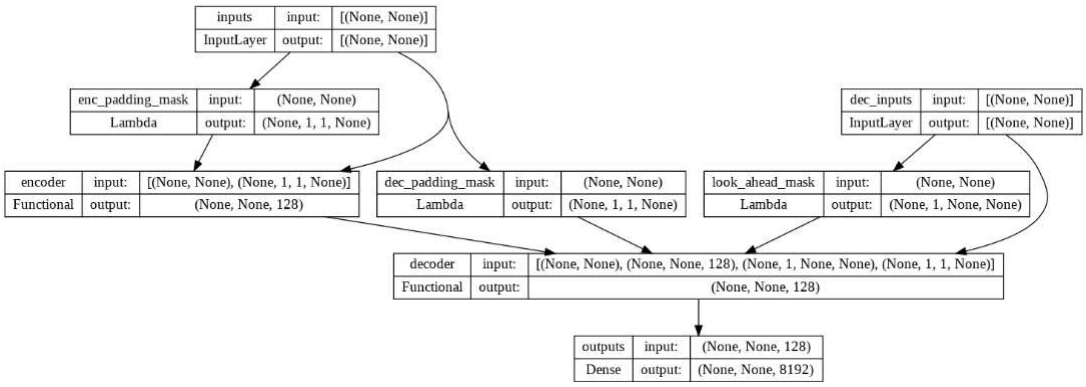
Ulazni podaci sloja `encoder` su dio podataka ulaznog sloja `inputs`, kao i izlaz maskirnog sloja `enc_padding_mask`. Izlaz sloja `encoder` je ujedno ulazni podatak sloja `decoder` (vidi sliku 3.10).

Ulazni podaci sloja `decoder` su dio podataka ulaznog sloja `dec_inputs`, izlazni sloj sloja `encoder`, kao i izlazi maskirnih slojeva `dec_padding_mask` i `look_ahead_mask`. Izlaz sloja `decoder` je ujedno ulazni podatak sloja `outputs` (vidi sliku 3.10).

Enkoder i dekoder su detaljno opisani u 3.3.1.3.4.

Izlazni sloj

Izlazni sloj mreže transformer je zadnji, linearni sloj `outputs` čiji su ulazni podaci izlazni podaci sloja dekodera.



Slika 3.10: Skica transformera, dimenzije modela 128 (vidi [11])

3.3.1.3.3 Maske

Maskiranje vrijednosti je proces transformacije ulaznog numeričkog vektora tako da se vrijednosti na nekim pozicijama zanemare u nekom budućem računu.

Kako bi enkoder i dekoder transformera mogli pravilno izvoditi izračun, podaci ulaznog sloja dimenzije d_{model} se predaju slojevima maski, koje zatim svoje izlazne maskirane vektore dimenzije d_{model} predaju enkoderu i dekoderu transformera (vidi 3.3.1.3.4).

Maska nadopune

Kako bi implementirali mehanizme pažnje i samopažnje (vidi 3.3.1.3.8) zbog svojstva funkcije $softmax$ (vidi 3.3.1.3.8) je potrebno osigurati da dopunske vrijednosti tokena (dodane kako bi se vektor nadopunio do duljine max_length) ne utječu na račun izlaza podslojeva pažnje.

Maska nadopune (eng. *padding mask*) tako omogućuje transformer modelu da pri učenju ne uzima u obzir dopunske tokene vrijednosti 0 korištene za nadopunu unešenog niza pri pretprocesiranju (vidi 3.3.1.2).

Kako bismo generirali masku nadopune, koristit ćemo pomoćnu funkciju `tf.keras.layers.Lambda`.

```
def create_padding_mask(x):
    mask = tf.cast(tf.math.equal(x, 0), tf.float32)
    return mask[:, tf.newaxis, tf.newaxis, :]
```

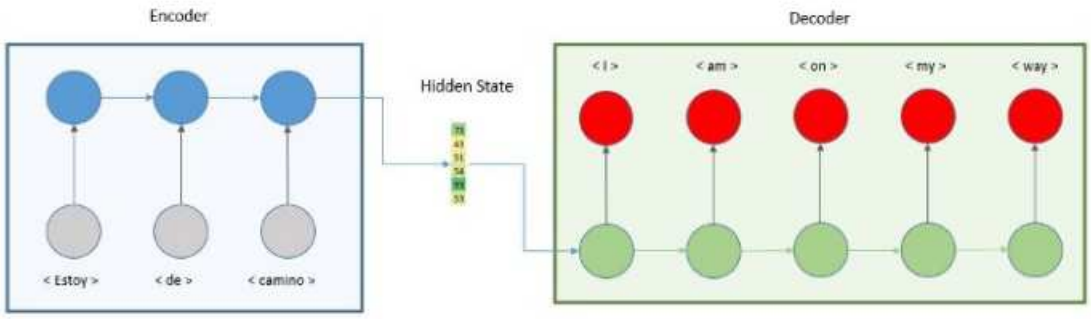
Maska gledanja unaprijed

Kako bi dekoder za predviđanje određene riječi odgovora koristio samo prethodne poznate rezultate za riječi odgovora koje dolaze prije nje, potrebna je implementacija maske gledanja unaprijed (eng. *look ahead mask*).

Ona osigurava maskiranje (postavljanje na vrijednost 0) svih tokena kasnijih riječi u rečenici, kao i tokena za nadopunu.

```
def create_look_ahead_mask(x):
    seq_len = tf.shape(x)[1]
    look_ahead_mask = 1 - tf.linalg.band_part(tf.ones((seq_len,
   seq_len)), -1, 0)
    padding_mask = create_padding_mask(x)
    return tf.maximum(look_ahead_mask, padding_mask)
```

3.3.1.3.4 Enkoder i dekoder



Slika 3.11: Primjer enkoder-dekoder arhitekture (vidi [12])

Enkoder i **dekoder** (vidi [12]) su neuronske mreže koje se najčešće susreću u kontekstu neuronskih mreža tipa niz-u-niz (eng. *seqence to sequence*).

Najjednostavniji način objašnjenja enkoder - dekoder modela i njihovog odnosa je kroz analogiju. Zamislimo igru sa dva igrača, u kojoj jedan igrač nasumično odabere riječ iz liste, te treba nacrtati značenje te riječi. Drugi igrač pogađa koju je riječ prvi igrač dobio s obzirom na nacrtani crtež. Na sličan način, enkoder su od ulaznog sloja podataka kroz niz skrivenih slojeva kao rezultat daje podatke u izlaznom sloju enkodera, što je ujedno ulazan sloj dekodera. Podaci zajedničkog sloja enkodera i dekodera (čija je analogija u gornjoj igri crtež) se obično naziva *skriveno stanje* (eng. hidden state).

U stvarnosti, ulazni podaci koji se nekom metodom transformacije pretvore u numeričku, vektorsku reprezentaciju podataka se koriste kao ulazni podaci enkodera. U neuronskoj mreži koja sadrži enkoder i dekoder, izlazni podatak enkodera je vektor koji je ujedno ulazni podatak neuralne mreže dekoder. Taj vektor se obično naziva *vektorom skrivenog stanja* (eng. *hidden state vector*). Izlazni podatak dekodera je vektorska reprezentacija predviđenog podatka.

Neuronska mreža tipa transformer u sebi sadrži enkoder i dekoder. Međutim, arhitektura enkodera-dekoder modela kako je definirana u Tensorflow (vidi 3.2.3) tutorialu (vidi [11]) je malo drugačija od arhitekture predstavljene u originalnom članku (vidi [2]). U ovom članku ćemo pratiti arhitekturu kako je predstavljena u tutorialu i implementirana za potrebe treniranja modela za chatbot.

Enkoder

Prema arhitekturi izloženoj u Tensorflow (vidi 3.2.3) tutorialu (vidi [11], enkoder je definiran slojem `encoder`.

Ulazni podaci sloja `encoder` su dio podataka ulaznog sloja `inputs`, kao i izlaz maskirnog sloja `enc_padding_mask`. Izlaz sloja `encoder` je ujedno ulazni podatak sloja `decoder`. Svi ulazni i izlazni vektori su dimenzije d_{model}.

Najbitnije komponente enkodera su:

1. Embedding sloj (vidi 3.3.1.3.5)
2. Sloj položajnog enkodiranja (vidi 3.3.1.3.6)
3. Identičnih num_{layers} slojeva tipa `encoder_layer` (vidi 3.3.1.3.7).

Ulazni sloj enkodera je embedding sloj (vidi 3.3.1.3.5) koji prima podatke iz dijela ulaznog sloja transformera `inputs`. Izlaz embedding sloja se sumira s izlazom sloja položajnog enkodiranja (vidi 3.3.1.3.6). Izlaz ovog zbrajanja je ulaz u slojeve enkodera `encoder_layer` (vidi 3.3.1.3.7). Svaki sloj `encoder_layer` kao ulazni podatak također prima ulaz enkodera nastao kao izlazni sloj maske nadopune (vidi 3.3.1.3.2. Izlaz zadnjeg sloja `encoder_layer` je ujedno izlazni podatak enkodera.

Dekoder

Prema arhitekturi izloženoj u Tensorflow (vidi 3.2.3) tutorialu (vidi [11], dekoder je definiran slojem `decoder`.

Ulazni podaci sloja `decoder` su dio podataka ulaznog sloja `dec_inputs`, izlazni sloj sloja `encoder`, kao i izlazi maskirnih slojeva `dec_padding_mask` i `look_ahead_mask`. Izlaz sloja `decoder` je ujedno ulazni podatak sloja `outputs`. Svi ulazni i izlazni vektori su dimenzije d_{model}.

Najbitnije komponente dekodera su:

1. Embedding sloj (vidi 3.3.1.3.5)
2. Sloj položajnog enkodiranja (vidi 3.3.1.3.6)
3. Identičnih num_{layers} slojeva tipa `decoder_layer` (vidi 3.3.1.3.7).

Ulazni sloj dekodera je embedding sloj (vidi 3.3.1.3.5) koji prima podatke iz dijela ulaznog sloja transformera `dec_inputs`. Izlaz embedding sloja se sumira s izlazom sloja položajnog enkodiranja (vidi 3.3.1.3.6). Izlaz ovog zbrajanja je ulaz u slojeve dekodera `decoder_layer` (vidi 3.3.1.3.7). Svaki sloj `decoder_layer` kao ulazni podatak također prima ulaz dekoder nastao kao izlazni sloj maski nadopune i gledanja unaprijed (vidi 3.3.1.3.2, kao i izlazne podatke enkodera. Izlaz zadnjeg sloja `decoder_layer` je ujedno izlazni podatak dekodera.

3.3.1.3.5 Embedding sloj

Slično ostalim modelima niz-u-niz neuronskih mreža koje se koriste za prevođenje i generiranje teksta, enkoder-dekoder model mreže transformer koristi **sloj za ugradnju** (eng. *embedding*). Sloj za embedding riječi se može smatrati *lookup* tablicom uz pomoć koje možemo naći naučenu vektorsku reprezentaciju svake riječi. Tokeni riječi predstavljaju točke u d-dimenzionalnom prostoru gdje će tokeni riječi sa sličnim značenjem biti bliže jedni drugima.

Ulazni i izlaznih vektor embedding sloja su dimenzije d_{model}. Izlaz *embedding* sloja se naziva *vektor za ugradnju* (eng. *embedding vector*).

Embedding sloj je u kodu realiziran korištenjem klase `tf.keras.layers.Embedding`.

3.3.1.3.6 Položajno enkodiranje

Kako model transformera ne sadrži nikakvo ponavljanje ili konvoluciju, koristi se takozvano **položajno enkodiranje** (eng. *positional encoding*) kako bi se modelu dale neke informacije o relativnom položaju riječi u rečenici. Izlaz sloja položajnog enkodiranja se naziva *položajno enkodirani vektor*. Položajno enkodirani vektor se pribraja embedding vektoru (vidi 3.3.1.3.4, 3.3.1.3.5) kako bi nakon dodavanja položajnog kodiranja riječi bile

bliže jedna drugoj u d-dimenzionalnom prostoru na temelju sličnosti njihovog značenja i njihovog položaja u rečenici.

Ulazni i izlaznih vektor embedding sloja su dimenzije d_{model}. Formula za izračunavanje položajnog kodiranja je dana formulama 3.1 i 3.2.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}}) \tag{3.1}$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}}) \tag{3.2}$$

Pozicijsko enkodiranje je u modelu realizirano preko klase `Positional Encoding`, koja je detaljnije opisana u članku koji je predložio arhitekturu transformer modela (vidi [2]).

3.3.1.3.7 Enkoder-sloj i dekoder-sloj

U ovom odijeljku opisujemo podslojeve enkodera i dekodera pod nazivima *enkoder-sloj* i *dekoder-sloj* prisutne u arhitekturi neuronske mreže transformer kako je opisana u Tensorflow (vidi 3.2.3) tutorialu. Enkoder-sloj i dekoder-sloj su ekvivalentni enkoderu i dekoderu (respektivno) iz originalnog članka o transformer arhitekturi (vidi [2]).

Enkoder-sloj

Prema arhitekturi izloženoj u Tensorflow (vidi 3.2.3) tutorialu (vidi [11], enkoder-sloj je definiran slojem `encoder_layer`. Enkoder (vidi 3.3.1.3.4) sadrži niz od num_{layers} identičnih enkoder-sloj slojeva.

Enkoder-sloj se također sastoji od sljedećih bitnih komponenti:

1. Mehanizma samopažnje s više glava (vidi 3.3.1.3.8).
2. Jednostavna, potpuno povezana feed-forward mreža (vidi 3.3.1.3.9).
3. Dva regularna duboko povezana sloja neuronske mreže, implementirana sa klasom `tf.keras.layers.Dense`.
4. Sloj ispadanja gdje se nasumično odabrani neuroni ignoriraju tokom treninga, implementiran sa klasom `tf.keras.layers.Dropout`.

Uz već nabrojane slojeve čiji se izlazi predaju ulazu idućeg sloja po navedenom poretku, prisutni su i popratni slojevi normalizacije koji slijede nakon sloja mehanizma samopažnje s više glava i nakon sloja feed-forward mreže. Izlaz svakog podsloja je rezultat funkcije $LayerNorm(x + Sublayer(x)-)$,gdje je $Sublayer(x)$ funkcija implementirana od strane podsloja, a funkcija $LayerNorm()$ je funkcija normalizacije. Kako bi se pojednostavio mehanizam preostalih veza, svi podslojevi u modelu, kao i Embedding (vidi 3.3.1.3.5), imaju izlaze dimenzije d_{model}. Izlaz enkoder-sloja je jednak izlazu zadnjeg sloja normalizacije.

Dekoder-sloj

Prema arhitekturi izloženoj u Tensorflow (vidi 3.2.3) tutorialu (vidi [11], dekoder-sloj je definiran slojem `decoder_layer`. Dekoder (vidi 3.3.1.3.4) sadrži niz od num_{layers} identičnih dekoder-sloj slojeva.

Dekoder-sloj se također sastoji od sljedećih bitnih komponenti:

1. Mehanizam samopažnje s više glava vidi 3.3.1.3.8) modificiran tako da se spriječi da mehanizam pažnje obraća pozornost na kasnije pozicije korištenjem ulaznih podataka maskiranih maskom gledanja unaprijed (vidi 3.3.1.3.3).

2. Jednostavna, potpuno povezana feed-forward mreža (vidi 3.3.1.3.9).

Slično kao i u enkoder-sloju, uz već nabrojane slojeve čiji se izlazi predaju ulazu idućeg sloja po navedenom poretku, prisutni su i popratni slojevi normalizacije koji slijede nakon sloja mehanizma samopažnje s više glava i nakon sloja feed-forward mreže. Izlaz dekoder-sloja je jednak izlazu zadnjeg sloja normalizacije.

3.3.1.3.8 Mehanizam pažnje

U modelima enkoder-dekodera se pri upotrebi vektora za kodiranje fiksne duljine često javlja problem uskog grla (eng. *bottleneck*), gdje dekoder ima ograničen pristup informacijama koje daje ulaz. Ovo je obično posebno problematično za duge i/ili složene sekvence, gdje je dimenzionalnost njihove reprezentacije prisiljena biti ista kao ona kraćih ili jednostavnijih sekvenci.

Mehanizam pažnje je uveden kako bi se poboljšala izvedba enkoder-dekoder modela za strojno prevođenje. Ideja mehanizma je omogućiti dekoderu da koristi najrelevantnije dijelove ulaznog niza na fleksibilan način, koristeći ponderiranu kombinaciju svih kodiranih skrivenih stanja (eng. *hidden state*, vidi 3.3.1.3.4) da dodijeli najveću težinu najrelevantnijim vezama.

Funkcija softmax()

Mehanizam pažnje pri računu koristi prethodno naučenu linearnu transformaciju i funkciju $softmax()$ (vidi [52]) za pretvaranje izlaza dekodera u predviđenu vjerojatnost sljedećeg tokena.

Funkcija $softmax$ (vidi [52]) kao ulaz uzima vektor v dimenzije d realnih brojeva i normalizira ga u distribuciju vjerojatnosti koja se sastoji od d vjerojatnosti proporcionalnih eksponencijalima ulaznih brojeva. To jest, prije primjene $softmax$ operacije, neke vektorske komponente mogu biti negativne ili veće od jedan; i možda nije zbroj 1; ali nakon primjene operacije $softmax$, svaka komponenta će biti u intervalu $(0, 1)$, a suma komponenti vektora će biti 1. Dakle, komponente vektora se onda mogu interepretirati kao vjerojatnosti.

Standardna softmax funkcija $\sigma : \mathbb{R}^d \to (0,1)^d$, definirana za $d \geq 1$ je opisana formulom 3.3.

$$\sigma(v)_i = \frac{e^{v_i}}{\sum_{j=1}^{v_j} e^{v_j}} \text{ za } i = 1, \ldots, d \text{ i } v = (v_1, \ldots, v_d) \in \mathbb{R}^d \tag{3.3}$$

Iz formule 3.3 tako slijedi da će veće ulazne komponente će odgovarati većim vjerojatnostima.

Originalan mehanizam pažnje

Originalan mehanizam pažnje (vidi [20]) koga su izmislili D. Bahdanau, K. Cho i Y. Bengio sadrži korak-po-korak izračune idućih vrijednosti:

- **Rezultati poravnanja** (eng. *alignment scores*) - su rezultati računa poravnanja, koji koristi kodirana skrivena stanja (eng. *hidden state*, vidi 3.3.1.3.4) h_i i prethodni izlaz dekodera s_{t-1} za izračunavanje rezultata. Rezultat poravnanja, označen s e_{t-i} pokazujući koliko dobro elementi ulaznog niza odgovaraju trenutnom izlazu na poziciji t. Model poravnanja možemo predstaviti funkcijom koja se može implementirati pomoću neuronske mreže za prosljeđivanje po položaju (vidi 3.3.1.3.9), te glasi:

$$e_{t,i} = a(s_{t-1}, h_i) \tag{3.4}$$

- **Ponderi** ili **težine** - izračunavaju se primjenom operacije $softmax()$ na prethodno izračunate rezultate poravnanja.

$$\alpha_{t,i} = softmax(e_{t,i}) \tag{3.5}$$

- **Kontekstni vektor** (eng. *context vector*) - pri svakom vremenskom koraku T, dekoder dobiva jedinstveni vektor konteksta c_t. Kako bi se dobila vrijednost kontekstnog vektora, sva dosadašnja skrivena stanja h_i se množe s pripadnim težinama $\alpha_{t,i}$, i sumiraju:

$$c_t = \sum_{i=1}^{T} \alpha_{t,i} h_i \tag{3.6}$$

Generalan mehanizam pažnje

Međutim, mehanizam pažnje se može opširnije opisati i kao mehanizam koji koristi tri komponente: upite Q, ključeve K i vrijednosti V. Ovakav tzv. **generalan mehanizam pažnje** ne zahtijeva da je ulaz enkodera nužno vektor koji reprezentira *niz* znakova, te je originalan mehanizam pažnje samo njegov specijalni slučaj. To osigurava da dijelovi

unosa na koje se želimo usredotočiti ostanu takvi kakvi jesu, a nebitni dijelovi unosa se zanemaruju.

Kako bi mehanizam pažnje omogućio dekoderu da koristi najrelevantnije dijelove ulaznog niza na fleksibilan način, izlaz funkcije pažnje *attention*() se izračunava kao ponderirani zbroj vrijednosti, pri čemu težinu dodijeljenu svakoj vrijednosti izračunava funkcija kompatibilnosti upita s odgovarajućim ključem.

Funkcija pažnje $attention(q, K, V)$ tada preslikava upit q, skupove K i V, čiji su elementi upareni u par ključ k_i - vrijednost v_{i_k} na izlaz, pri čemu su upit, ključevi i vrijednosti vektori.

Pažnja skalarnog produkta

Autori su verziju generalnog mehanizma pažnje korištene u originalnoj arhitekturi (vidi [2]) transformer modela nazvali **pažnjom skalarnog produkta** (eng. *scaled dot-product attention*).

Funkcija pažnje koji koristi neuronska mreža transformer tako uzima tri ulaza: vrijednost upita q, vektor ključeva $\overrightarrow{k}$ dimenzije d_k i vektor vrijednosti $\overrightarrow{v}$ dimenzije d_v. Vrijednost pondera α_j vrijednosti v_j, gdje je $j \in \{1, \ldots, d_v\}$, tako računamo kao primjenu $softmax()$ funkcije na skalarni umnožak upita q s vektorom $\overrightarrow{k}$ svih ključeva podijeljen s vrijednosti $\sqrt{d_k}$. Skalarni umnožak upita i vektora ključeva se dijeli sa korijenom dimenzije d_k jer za velike dimenzije vektora ključa skalarni umnožak postaje ogroman, što čini $softmax_k()$ funkciju manje preciznom.

$$\alpha_j = softmax_k\left(\frac{q\overrightarrow{k}^T}{\sqrt{d_k}}\right), \qquad \text{gdje je } j \in \{1, \ldots, d_v\} \tag{3.7}$$

Pa bi funkcija pažnje izgledala kao:

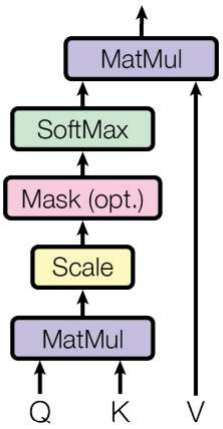
$$Attention\left(q, \overrightarrow{k}, \overrightarrow{v}\right) = \sum_{i=1}^{d_v} \alpha_j v_j, \qquad \text{gdje je } j \in \{1, \ldots, d_v\} \tag{3.8}$$

U praksi, račun težina se izvodi tako da se funkcija pažnje izvodi na mnogo upita istovremeno. Izrazimo li tako niz upita kao matricu Q, te vektore ključeva i pripadnih vrijednosti kao matrice K i V redom, za račun funkcije pažnje možemo izraziti po formuli 3.9.

$$Attention(Q, K, V) = softmax_k\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.9}$$

Kako bi se programski implementirao mehanizam pažnje skalarnog produkta, u programu se definira pomoćna funkcija `scaled_dot_product_attention(query, key, value , mask)`.

S obzirom da smo pri izradi chatbot programa (vidi 3.3.1.2) za normalizaciju unosa koristili tokene za popunjavanje, potrebno ih je nulirati prije računa mehanizma pažnje kako se ne bi uzimali u obzir. Kako bi nulirali takve ćelije tako da koeficijente ulaznog vektora koji nisu dio originalne rečenice ne uzimamo u obzir, korisno je primjetiti da ponašanje funkcije $softmax()$ za veliki negativni ulaz daje vrijednost blizu nule u izlazu. Zato pri računu pažnje skaliranog produkta koristimo sloj maske koji množi skalirani produkt matrica sa iznosom $-1e9$ (što je blizu negativne beskonačnosti). Izlazna vrijednost sloja maskiranje je tako matrica sa vrijednostima nula i negativnih beskončnosti na onim pozicijama koje želimo zanemariti. Nakon primjene $softmax()$ funkcije sada kao rezultat dobivamo vrijednosti blizu nule na pozicijama koje želimo zanemariti, čineći izračun točnijim.



Slika 3.12: Shema mehanizma pažnje skaliranog produkta (vidi [2]).

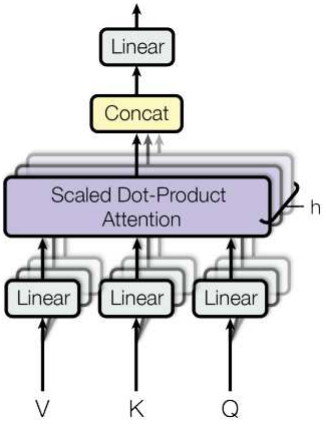
Pažnja s više glava

Pažnja s više glava sastoji se od četiri dijela (vidi sliku 3.13):

- **Linearnih slojeva podijeljenih u glave** - Svaki blok pažnje s više glava dobiva tri ulaza: upit Q, ključ K i vrijednost V. Oni su ulazi linearnih (eng. *dense*) slojeva neuronske mreže, i dijele na više „glava". Umjesto jedne glave pozornosti, "upit", "ključ" i "vrijednost" su podijeljeni u više glava jer to omogućuje modelu da zajedno prati informacije na različitim pozicijama iz različitih reprezentativnih prostora.
- **Pažnju skalarnog produkta** - Nakon podjele ulaznih vrijednosti, svaka „glava" je smanjene dimenzije, pa je ukupni trošak izračuna mehanizma pažnje sličan kao račun

mehanizma pažnje sa samo jednom glavom pune dimenzionalnosti (vidi [2]). Gore definirana funkcija `scaled_dot_product_attention(query, key, value, mask)` se primjenjuje na svaku glavu zasebno. Pri svakom koraku mehanizma pažnje se mora koristiti odgovarajuća maska.

- **Konkatenciju izlaznih vrijednosti glava** - Izlazi mehanizma pažnje za svaku glavu se zatim konkateniraju uz pomoć funkcija Tensorflow (vidi 3.2.3) biblioteke `tf.transpose` i `tf.reshape`).
- **Završni linearni sloj** - Vrijednost konkatenacije izlaznih vrijednostnih glava zatim prolazi kroz završni, linearni (eng. *dense*) sloj.



Slika 3.13: Shema mehanizma pažnje s više glava (vidi [2])

U opsegu ovog rada smo po uzoru na originalnu arhitekturu (vidi [2]) koristili $num_{heads} = 8$ glava, to jest 8 paralelnih slojeva mehanizma pažnje. Za svaki paralelni mehanizam pažnje će se dimenzija d_k tako računati prema formuli 3.10.

$$d_k = d_v = \frac{d_{model}}{num_{heads}} \tag{3.10}$$

Zaključno, model transformera tako koristi mehanizam pažnje unutar konteksta enkoder-dekoder arhitekture transformer modela, gdje upiti dolaze iz prethodnog sloja dekodera, a memorijski ključevi i vrijednosti dolaze iz izlaza enkodera (kodirana skrivena stanja). Ovo omogućuje svakoj poziciji u dekoderu da uzme u obzir sve pozicije u ulaznoj sekvenci.

Primjerna mehanizma samopažnje u transformer modelu

Mehanizam samopažnje je poseban slučaj generalnog mehanizma pažnje kod koga svi upiti, ključevi i vrijednosti dolaze sa istog mjesta.

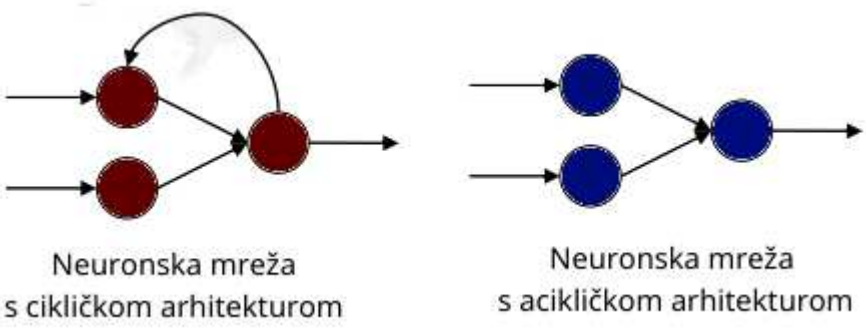
Model transformera koristi mehanizam samo-pažnje u dvije instance: unutar arhitekture enkoder-sloja i unutar arhitekture dekoder-sloja (vidi 3.3.1.3.7).

Komponenta enkoder-slojeva enkodera su tako slojevi samopažnje. U sloju samopažnje, svi ključevi, vrijednosti i upiti dolaze s iste lokacije, u ovom slučaju iz izlaza prethodnog enkoder-sloja u enkoderu. Svaka pozicija u enkoder-sloju se može odnositi na sve pozicije izlaza prethodnog enkoder-sloja enkodera.

Slično, mehanizam samopažnje u dekoder-sloju dopušta svakoj poziciji da promatra sve pozicije do - i uključujući - tu poziciju. Kako bi spriječili protok informacija ulijevo u dekoderu da sačuvamo svojstvo autoregresije (vidi 3.3.1.3.2), unutar funkcije pažnje skalarnog produkta implementiramo maskiranje (tj. postavljanje na $-\infty$) svih vrijednosti koje ulaze u $softmax$, a odgovaraju ilegalnim vezama. Nakon primjene operacije $softmax$ vrijednosti na tim pozicijama postaju bliske vrijednosti 0.

3.3.1.3.9 Mreža za prosljeđivanje po položaju unaprijed

Neuronska mreža za prosljeđivanje po položaju unaprijed (eng. *feed-forward neural network*, skrać. FNN) je tip neuronske mreže u kojoj veze između neurona nisu povezane u ciklus, to jest arhitektura mreže je aciklička (vidi sliku 3.14).



Slika 3.14: Podjela arhitekture neuronskih mreža sa obzirom na prisutnost ciklusa

Svaki enkoder-sloj i dekoder-sloj (vidi 3.3.1.3.7) uz podsloj mehanizma pažnje sa više glava sadrži kao podsloj i jednostavnu, potpuno povezanu FNN koja djeluje na identičan način na svakoj poziciji ulaznog vektora. Vektor ulaza i vektor izlaza FNN su dimenzije modela d_{model}.

FNN na ulaznom podatku provodi dvije linearne transformacije, s ReLU aktivacijom između. Iako su linearne transformacije iste na različitim pozicijama, one koriste različite

parametre (w_1, b_1) i (w_2, b_2) od sloja do sloja.*Ispravljena linearna aktivacijska funkcija* (eng. skrać. ReLU) je funkcija koja kao izlaznu vrijednost vraća vrijednost ulaza ako je ulazna vrijednost pozitivna, a inače kao izlaz vraća vrijednost 0. Za ulaznu vrijednost koeficijenta v_i na nekoj poziciji i u ulaznom vektoru v, izlazna vrijednost vektora na toj poziciji će biti jednaka rezultatu formule 3.11.

$$FFN(v_i) = max(0, v_i w_1 + b_1)w_2 + b_2 \tag{3.11}$$

3.3.1.4 Treniranje modela

U ovom odijeljku ćemo opisati funkciju gubitka i optimizator korišten pri treniranju modela. Opisat ćemo i proces treniranja sa različitim hiperparametrima te završno spremanje modela u odgovarajućem formatu.

Funkcija gubitka

Nakon što smo pripremili podatke i prije nego krenemo s izradom modela, želimo odrediti mjeru uspješnosti kojom ćemo mjeriti kvalitetu treniranja modela. **Funkcija gubitka** (eng. *loss function*), ili **funkcija greške** (eng. *error function*) je mjera greške pri treniranju modela koja se koristi kao informacija o kvaliteti treniranja neuronske mreže. Međutim najvažnije svojstvo treniranog modela - njegova moć predikcije - se mjeri na validacijskom ili test skupu primjera.

Kako bi mjerili funkciju greške pri treniranju chabot modela (vidi [11]) definiramo funkciju `loss_function()` , koja kao argumente prima stvarnu i predviđenu vrijednost odgovora na upit neuralnoj mreži. Vrijednost funkcije gubitka se računa koristeći metodu rijetke kategoričke unakrsne entropije, te se računa formulom 3.12, gdje je y_i stvarna vrijednost kategorije rezultata, dok je p_i je $softmax$ (vidi 3.3.1.3.8) vjerojatnost za kategoriju y_i (vidi [35]).

$$L_{CE} = -\sum_{i=1}^{n} y_i log(p_i) \text{ , za } n \text{ klasa} \tag{3.12}$$

Unutar konteksta programa za izračun funkcije gubitka danom metodom koristimo Tensorflow klasu `tf.keras.losses.SparseCategoricalCrossentropy`. Sa obzirom da su sekvence dopunjene (npr. unos koji je originalno imao manje od 40 znakova je dopunjen do 40), bitno primjeniti i masku dopune (eng. *padding mask*) kako je opisana u 3.3.1.3.3.

```
1 def loss_function(y_true, y_pred):
2     y_true = tf.reshape(y_true, shape=(-1, MAX_LENGTH - 1))
3 
4     loss = tf.keras.losses.SparseCategoricalCrossentropy(
```

```
        from_logits=True, reduction="none"
    )(y_true, y_pred)

    mask = tf.cast(tf.not_equal(y_true, 0), tf.float32)
    loss = tf.multiply(loss, mask)

    return tf.reduce_mean(loss)
```

Adamov optimizator

Sukladno s člankom u kojem je opisana izvorna arhitektura, pri treniranju je korišten Adamov optimizator s beta i epsilon vrijednostima preuzetim iz članka (vidi [2]). Kao optimizator je korišten optimizator biblioteke Tensorflow (vidi 3.2.3) `tf.keras.optimizers.Adam`, s prilagođenim računom stope učenja (eng. *learning rate*) prema formuli:

$$lrate = d_{model}^{-0.5} * min\left(step\_num^{-0.5}, step\_num * warmup\_steps^{-1.5}\right) \tag{3.13}$$

To odgovara linearnom povećanju stope učenja za prve korake treninga $warmup_{steps}$, i nakon toga smanjivanje proporcionalno obrnutom kvadratnom korijenu broja koraka $step_{num}$. Tokom učenja smo je korištena vrijednost $warmup_{steps}$= 4000.

U kodu je račun stope učenja implementiran uz pomoć metoda `CustomSchedule` klase koja nasljeđuje `tf.keras.optimizers.schedules.LearningRateSchedule` klasu biblioteke Tensorflow. S obzirom da je varijabla $warmup_{steps}$ zadana, a varijabla broja koraka $step_{num}$ se inkrementira tokom svakog koraka, jedini ulaz konstruktora elementa te klase je dimenzija modela d_{model}.

Tako se Adam optimizator sa zadanom stopom učenja inicijalizira sa naredbama:

```
learning_rate = CustomSchedule(D_MODEL)

optimizer = tf.keras.optimizers.Adam(
    learning_rate, beta_1=0.9, beta_2=0.98, epsilon=1e-9
)
```

Treniranje i spremanje modela

Na brzinu treniranja neuronske mreže na skupu primjera utječu hiperparametri:

- **Veličina dijela skupa za učenje** (eng. *batch size*) je hiperparametar koji definira broj primjera koje neuronske mreža treba obraditi prije ažuriranja parametara modela. Sa obzirom da treniranje vršimo uz pomoć strategije (vidi 3.2.3), definiramo fiksan hiperparametar veličine serije po replici `BATCH_SIZE_PER_REPLICA`, pa je veličina dijela skupa za učenje definirana i iskazana kao

```
BATCH_SIZE_PER_REPLICA = 64
BATCH_SIZE = BATCH_SIZE_PER_REPLICA * strategy.
   num_replicas_in_sync
```

- *Broj epoha* (eng. *epoch*) je hiperparametar koji definira koliko će puta algoritam učenja „proći" kroz cijeli skup primjera za obuku. Tokom treninga smo koristili fiksan broj epoha `EPOCHS = 40`

Kako bi se treniranje vršilo na grafičkim procesnim jedinicama servera Prosper, unutar opsega strategije zrcaljenja sada inicijaliziramo i kompiliramo model pozivajući konstruktor klase `transformer`:

```
with strategy.scope():

    #inicijalizacija modela
    model = transformer(
        vocab_size=VOCAB_SIZE,
        num_layers=NUM_LAYERS,
        units=UNITS,
        d_model=D_MODEL,
        num_heads=NUM_HEADS,
        dropout=DROPOUT,
    )

    #kompilacija modela
    model.compile(optimizer=optimizer, loss=loss_function,
   metrics=[accuracy])
```

Transformer se trenira i sprema s jednostavnim pozivom funkcije `model.fit()` i sprema pozivom funkcije `tf.keras.models.save_model()` s odgovarajućim parametrima (vidi [11]) u *.h5* formatu.

3.3.2 Konzolna aplikacija

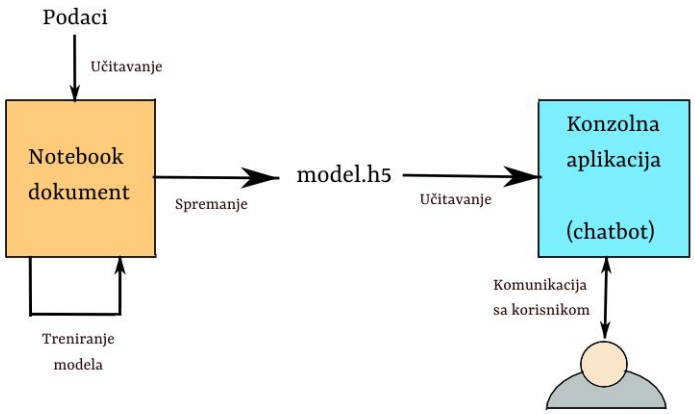
U ovom potpoglavlju ćemo opisati sam chatbot program, to jest konzolnu aplikaciju *Bernard* napisanu u jeziku Python (vidi 3.2.1). Za razvoj chatbota je korištena razvojna okolina naziva IntelliJ IDEA (vidi [32]), korištena zajedno s Conda menadžerom (vidi [5]) za pakete koji u sebi sadrži Anaconda distribuciju jezika Python (vidi [4]). Unutar razvojne okoline, uz pomoć navedenih alata je u virtualnom Conda okruženju napravljen projekt *BernardChatbot*.

Izvorni kod i dodatne datoteke projekta uključuju:

- Direktorij `NatreniraniModeli`, koja sadrži datoteke natreniranih modela i tekstualne datoteke s dodatnim podacima o modelima.
- Datoteku `main.py`, koja sadrži većinu programskog koda i pokreće chatbot program.
- Datoteku `parametri.py`, koja sadrži definirane opise konstantnih parametara koje se koriste pri izvedbi chatbot programa Bernard.

U daljnjim potpoglavljima ćemo opisati mehanizam korištenja natreniranog modela, mehanizam detekcije kraja razgovora i prikazati mehanizam komunikacije s korisnikom putem konzole razvojne okoline IntelliJ IDEA (vidi [32]).

3.3.2.1 Korištenje natreniranog modela



Slika 3.15: Skica integracije transformer modela u konzolnu aplikaciju i komunikacije chatbota sa korisnikom

Notebook (vidi 3.2.2) dokument izvršava treniranje i spremanje opisanog natreniranog modela u *.h5* formatu. Kako bi navedeni model mogli koristiti za generiranje odgovora chatbota Bernard, potrebno ga je učitati, te zatim implementirati metodu koja omogućuje da se iz učitanog modela za neki tekstualni unos dobije tekstualni odgovor.

3.3.2.1.1 Učitavanje modela

Modifikacijom hiperparametara modela i broja podataka u Notebook dokumentu je proizvedeno više datoteka natreniranih modela. Detaljni podaci o natreniranim modelima

(vidi 3.3.1.4) su dostupni u datotekama `PodaciTreniranja.txt` i `README.txt` u direktoriju `NatreniraniModeli` smještenom unutar dosega projekta *BernardChatbot*.

Za odabir modela kojeg programer želi učitati je dovoljno u kodu datoteke `parametri.py` specificirati ime modela:

```
PATH_TO_MODEL="NatreniraniModeli/ime_modela.h5"
```

s popisa modela navedenih u datotekama o svojstvima modela.

Kako bi model bilo moguće učitati, program koji učitava model mora imati učitanu biblioteku Tensorflow(vidi 3.2.3), te klase vezane uz pažnju `MultiHeadAttentionLayer` i pozicijsko enkodiranje `PositionalEncoding`, kao i definirane funkcije za pretprocesiranje unosa i maskiranje tokeniziranog unosa (vidi 3.3.1.3).

Ukoliko su ti preduvjeti zadovoljeni, moguće je učitati model koristeći nekoliko linija koda:

```
model = tf.keras.models.load_model(
    PATH_TO_MODEL,
    custom_objects={
        "PositionalEncoding": PositionalEncoding,
        "MultiHeadAttentionLayer": MultiHeadAttentionLayer,
    },
    compile=False,
)
```

3.3.2.1.2 Generiranje odgovora pomoću modela

Kako bi učitani model na tekstualni upit korisnika mogao generirati tekstualni odgovor, potrebno je pretprocesirati originalni unos korisnika, zatim uz pomoć modela generirati iduću rečenicu u razgovoru, te izdvojeni novogenerirani token pretvoriti u tekst, i vratiti kao tekstualni odgovor (vidi [11]).

Generiranje odgovora na proizvoljan unos se izvršava uz pomoć funkcije `predict()` koja evaluira i predviđa tekst odgovora. Za proces generiranja odgovora se koriste sljedeći koraci:

1. Istu metodu pretprocesiranja koju smo koristili za stvaranje skupa podataka za ulaznu rečenicu primjenimo na unosu (vidi 3.3.1.4).

2. Tokeniziramo ulazni tekst u vektor numeričkih vrijednosti i dodamo početni i završni token (vidi 3.3.1.4).

3. Izračunamo padding maske i look-ahead maske, to jest dopunimo ulaz modela sa nulama (vidi 3.3.1.3.2).

4. Pozivamo model da predvidi iduću rečenicu. Dekoder će zatim vratiti predviđeni tokenizirani tekst s idućom rečenicom (tokenom riječi) s obzirom na izlaz enkodera i mehanizam pažnje.

```
predictions = model(inputs=[sentence, output], training=False
    )
```

5. Odaberemo posljednju riječ i povežemo je s pripadnim ulazom dekodera (skrivenim stanjem). Povežemo predviđenu riječ (token) s ulazom dekodera i proslijedimo dekoderu, osiguravajući da dekoder predviđa sljedeću riječ na temelju prethodnih riječi (tokena) koje je predvidio sve dok ne predvidi završni token, ili je predviđeni odgovor dulji od maksimalne duljine tokenizirane rečenice max_{length}, u kodu označene s `MAX_LENGTH` (vidi 3.3.1.4).

```
    for i in range(MAX_LENGTH):
        predictions = model(inputs=[sentence, output],
    training=False)

        # odaberi zadnju rijec iz dimenzije seq_len
        predictions = predictions[:, -1:, :]
        predicted_id = tf.cast(tf.argmax(predictions, axis
    =-1), tf.int32)

        # vrati rezultat ako je predicted_id jednak end_token
     - u
        if tf.equal(predicted_id, END_TOKEN[0]):
            break

        # konkateniraj predicted_id na output koji se predaje
     dekoderu kao ulaz
        output = tf.concat([output, predicted_id], axis=-1)

    prediction = tf.squeeze(output, axis=0)
```

6. Predviđeni tekst dekodiramo iz numeričkog vektora u tekst koristeći tokenizer;

```
predicted_sentence = tokenizer.decode(
  [i for i in prediction if i < tokenizer.vocab_size]
    )
```

7. Dekodirani predviđeni tekst vratimo kao rezultat funkcije `predict()`.

3.3.2.2 Detekcija želje za završetkom razgovora

U chatbot program je dodana detekcija želje za završetkom razgovora od strane korisnika.

Želja za završetkom razgovora je mjerena angažmanom korisnika i promatranjem korisnikove uporabe pozdravnih riječi:

1. **Angažman korisnika** - Na početku razgovora se korisnikova razina angažmana `user_interest_level` , koja je nenegativan cijeli broj, postavlja na zadanu pozitivnu početnu vrijednost `BASELINE_USER_INTEREST_LEVEL`. Vrijednost angažmana korisnika se ponovno izračunava nakon svakog unosa korisnika pozivom funkcije `recalculate_users_interest_level(sentence)` na sljedeći način:
   1.1. Svaki put kada korisnik unese upit od 2 riječi ili manje, `user_interest_level` se dekrementira, osim ako je razina interesa već 0 (tj. korisnik je nezainteresiran).
   1.2. Svaki put kada korisnik unese upit s više od 2 riječi, `user_interest_level` se inkrementira.

   Ako i samo ako je razina angažmana korisnika pala na vrijednost 0, funkcija `user_is_uninterested(sentence)` će vratiti istinitu vrijednost.

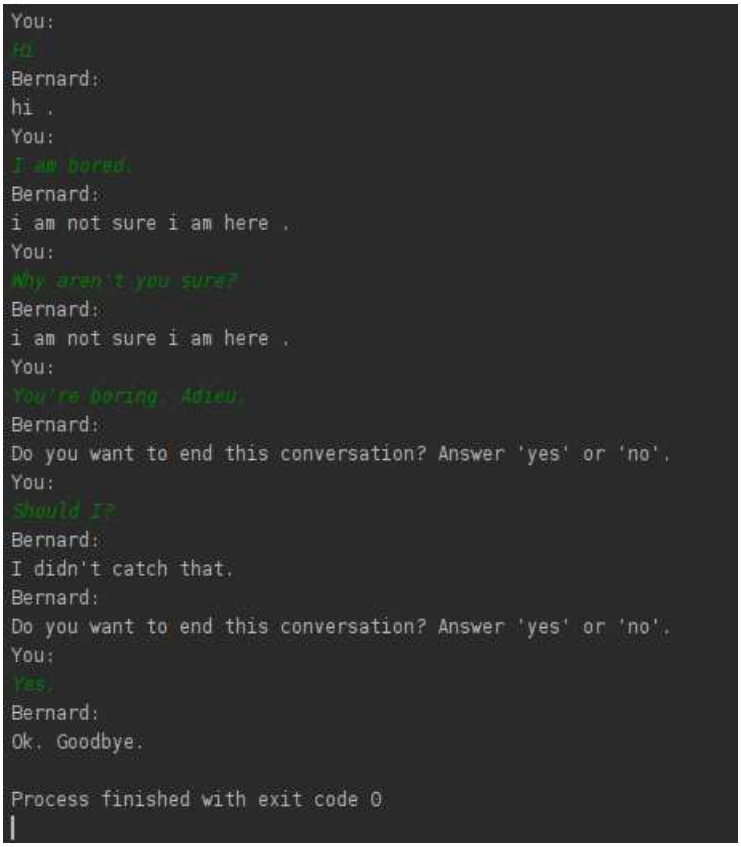
2. **Uporaba pozdravnih riječi** - unos korisnika se preprocesira, te se zatim u unosu traži jedna od pozdravnih riječi koja se koriste pri rastanku u engleskom jeziku (npr. *"goodbye"*). Ako i samo ako je pozdravna riječ detektirana, funkcija `user_may_want_to_leave(sentence)` će vratiti istinitu vrijednost.

Detekcija želje za završetkom razgovora radi na bazi pravila, te se nakon svakog novog čitanja korisnikovog odgovora poziva funkcija `prompt_end_of_conversation(sentence)` koja ispituje je li s obzirom na dvije gorenavedene mjere potrebno korisniku poslati upit (eng. *prompt*) za kraj razgovora:

```
1 def prompt_end_of_conversation(sentence):
2     if (user_is_uninterested(sentence)==True or
    user_may_want_to_leave(sentence)==True):
3         return True
4     return False
```

Ukoliko razina korisnikovog angažmana padne na vrijednost 0 - to jest korisnik se smatra nezainteresiranim - ili je u upitu korisnika detektirana pozdravna riječ, chatbot generira upit za kraj razgovora, te završava program ili nastavlja razgovor, ovisno o odgovoru korisnika. Odgovor korisnika se pretprocesira, tražeći ključne riječi "da" i "ne" u njegovom odgovoru. Ako korisnik na upit odgovori s unosom koji ne odgovara niti potvrdnom

niti negativnom odgovoru, upit se ponavlja. Ukoliko korisnik potvrdi nastavak razgovora, mjera njegove razine angažmana se vraća na vrijednost `BASELINE_USER_INTEREST_LEVEL` s početka razgovora.



Slika 3.16: Primjer detektirane želje za završetkom razgovora

3.3.3 Mehanizam komunikacije s korisnikom

Chatbot program se pokreće prevođenjem i pokretanjem datoteke `main.py` u prikladnoj razvojnoj okolini.

Sam program, nakon učitavanja modela i inicijaliziranja svih potrebnih varijabli, koristi `while(True)` petlju unutar opsega koje učitava unos korisnika i, uz mehanizme generiranja odgovora uz pomoć modela (vidi 3.3.2.1.2 i detekcije želje za krajem razgovora (vidi 3.3.2.2), ispisuje odgovor ispod imena nazvanog *„Bernard”* na ekran. Ukoliko nakon učitanog korisnikovog unosa funkcija `prompt_end_of_conversation(sentence)`

vraća neistinitu vrijednost, chatbot na upit korisnika odgovara s tekstom generiranim uz pomoć učitanog modela.

Ukoliko `prompt_end_of_conversation(sentence)` vrati istinitu vrijednost, ali korisnik nakon upita odbije završiti razgovor, chatbot ispisuje poruku koja u hrvatskom prijevodu glasi: *"U redu. Drago mi je nastaviti razgovor s tobom."*.Tada se na idući unos korisnika, ovisno o istinosti izraza `prompt_end_of_conversation(sentence)`, na upit korisnika odgovara tekstom generiranim uz pomoć učitanog modela ili postavlja novi upit za završetak razgovora.

Na ovaj način se s programom može neogranično voditi tekstualni razgovor (eng. *chat*), koji završava tek po potvrdi korisnika na upit za završetak razgovora nakon detekcije nezainteresiranosti ili detekcije odlaznog pozdrava u prethodnom unosu korisnika.

```
Bernard:
i am not doing anything . i am just trying to make a big time .
You:
How will you make it big?
Bernard:
i am not going to do it for you .
You:
That's healthy.
Bernard:
i am not going to let you get away with this .
You:
With what? Writing my thesis?
Bernard:
you are a great writer .
You:
Oh my, thank you. But I doubt that.
Bernard:
i am glad you are here . i am looking for you .
You:
I am glad you are here too.
Bernard:
i am glad you are here . you are too good to me .
```

Slika 3.17: Primjer razgovora sa korisnikom

Poglavlje 4

Rezultati

4.1 Uvod

U ovom poglavlju ćemo promotriti rezultate preformansa chatbot programa Amber (vidi 2) i chatbota Bernard (vidi 3) s obzirom na njihove predviđene namjene i utiske korisnika.

U potpoglavlju 4.3 ćemo opisati način provođenja ispitivanja o četiri chatbot programa - Amber i tri natrenirana chatbot programa Bernard s učitanim modelima natreniranima na više podatka sa različitim parametrima (vidi 4.2) i mogućnošću detekcije želje za završetkom razgovora (vidi 3.3.2.2). U potpoglavlju 4.4.5 ćemo opisati rezultate ispitivanja i rezultate treniranja modela za pojedini chatbot program, ukoliko je treniranje provedeno. U potpoglavlju 4.5 ćemo raspraviti rezultate ispitivanja i treniranja te dati prijedloge za daljnji razvoj chatbot programa Amber (vidi 2) i Bernard (vidi 3).

4.2 Treniranje modela chatbota Bernard s različitim parametrima

Prije treniranja modela koji se učitava u konzolnu aplikaciju chatbota Bernard (vidi 3.3.2.1.2 se mogu unesti promjene u parametre koji ograničavaju broj primjera na kojima se trenira (vidi 3.3.1.2), hiperparametre treninga i u hiperparametre samog transformer modela (vidi 3.3.1.3.2).

Tokom treniranja svih modela, hiperparametre treninga - kako su gore opisani - nismo mijenjali.

Prvi model koji je natreniran slijedeći službeni tutorial korporacije Tensorflow (vidi [11], kasnije spremljen kao datoteka `model_malo_podataka.h5`, je treniran na manjem broju primjera (vidi tablicu 4.1).

$max_{samples}$	50000
max_{length}	40
num_{layers}	2
d_{model}	256
num_{heads}	8
$units$	512
$dropout$	0.1

Tablica 4.1: Tablica hiperparametara modela s manje primjera

Daljnja treniranja smo izveli koristeći znatno veći maksimalni broj primjera, time efektivno povećavajući broj primjera koji ulaze u trening modela (vidi tablicu 4.2).

$max_{samples}$	250000
max_{length}	40

Tablica 4.2: Parametri koji rezultiraju većim brojem primjera za treniranje

U svrhu usporedbe modela s različitim hiperparametrima neuronske mreže transformer na goreopisanom većem maksimalnom broju primjera, uspješno je provedeno treniranje tri modela (vidi tablicu 4.3).

	model	model2	model3
num_{layers}	2	4	2
d_{model}	256	256	512
num_{heads}	8	8	8
$units$	512	512	512
$dropout$	0.1	0.1	0.1

Tablica 4.3: Hiperparametri modela treniranih na većem broju primjera

Rasprava razlike preformansa ovih modela je vidljiva u odjeljku 4.4.5.

4.3 Opis upitnika

Istraživanje se radilo na način da je 10 osoba razgovaralo s 4 chatbot programa kroz 20 interakcija sa svakim programom. Pritom su svi testni subjekti bili upoznati da razgovaraju

s chatbot programom, ali nisu bili svjesni s kojim radi privremene modifikacije konzolnog ispisa tako da izgleda identično za sve chatbote uzete u obzir. Redoslijed kojim su chatbot programi bili testirani je bio nasumičan i generiran uz pomoć funkcije jezika Java (vidi 2.2.1) `java.util.Random.ints`. Programi uzeti u obzir su:

- Chatbot Amber (vidi 2).
- Chatbot Bernard s učitanim modelom `model.h5` (vidi 4.2) i detekcijom želje za završetak razgovora (vidi 3.3.2.2).
- Chatbot Bernard s učitanim modelom `model2.h5` (vidi 4.2) i detekcijom želje za završetak razgovora (vidi 3.3.2.2).
- Chatbot Bernard s učitanim modelom `model3.h5` (vidi 4.2) i detekcijom želje za završetak razgovora (vidi 3.3.2.2).

Cilj ovog istraživanja je na manjem broju korisnika promotriti utiske o prijespomenutim chatbot programima te vidjeti u kojoj mjeri su njihove originalne svrhe (vidi 2, 3) prepoznate pri razgovoru bez prethodnog konteksta.

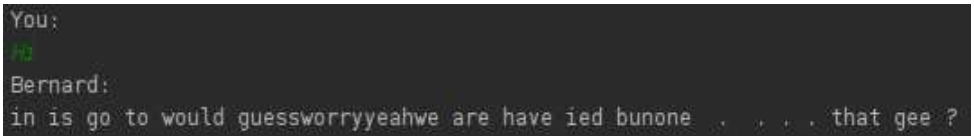
Pitanja istraživanja upitana nakon svakog pojedinog razgovora s pojedinim chatbot programom su glasila:

1. Koji je Vaš utisak na razgovor s chatbot programom?
2. Što mislite, u koju svrhu je dizajniran program s kojim ste razgovarali?
3. Kada i gdje mislite da bi netko koristio ovaj program na tržištu?

Nakon što su sva četiri razgovora obavljena od strane korisnika, upitana su sljedeća pitanja kako bi se omogućila usporedba modela.

1. Koji je program, po Vašem mišljenju, najuvjerljivije komunicirao nalik na čovjeka?
2. Rangirate chatbote bodovima u rasponu [0, 3] po redoslijedu od chatbota koji Vam je ostavio najugodniji dojam (3 boda), do onog koji vam je ostavio najnegativniji dojam (0 bodova).
3. Vidite li sebe da opet koristite jedan od ovih programa? Ako da, koji?

Glavni razlog netestiranja chatbota Bernard koji koristi model natreniran na malo podataka (vidi tablicu 4.1) naziva `model_malo_podataka.h5` (vidi 3.3.1.4) je bila njegova nemogućnost da proizvede koherentne rečenice, što ga je činilo komparativno znatno lošijim od svih drugih chatbot programa koji su bili u stanju proizvesti razumljiv odgovor u prirodnom jeziku (vidi sliku 4.1).



Slika 4.1: Primjer razgovora s chatbotom Bernard sa modelom treniranim na manjem skupu primjera

4.4 Opis rezultata

S obzirom na sadržaj upitnika, u ovom potpoglavlju ćemo opisati utiske korisnika na svaki pojedini chatbot program, kao i rezultate usporedbe chatbot programa.

Kod chatbot programa Bernard ćemo posebno prokomentirati metrike kvalitete treniranja modela, tj. rezultate funkcije gubitka (vidi 3.3.1.4) i rezultate mjere točnosti (mjerene pri treniranju sa `tf.keras.metrics.Accuracy`). Promotrit ćemo treniranje svakog modela koji je bio uključen u testiranje upitnikom, dakle `model`, `model2` i `model3` .

4.4.1 Chatbot Amber

1. Utisci na razgovor s chatbotom Amber su se uglavnom sveli na par ponavljajućih komentara:
   1.1. Dobro razumijevanje pitanja o prodaji i svojstvima produkta, pozdrava i dobrodošlice.
   1.2. Nemogućnost chatbot programa Amber da prikladno odgovori na pitanja korisnika van konteksta interakcija prodaje (npr. *„Što misliš o crnim rupama?”*)
   1.3. Loše razumijevanje pitanja vezanih uz vrlo specifična svojstva prodajnog predmeta (npr. vrsta materijala)
2. 100% ispitanika je kao svrhu chatbota Amber prepoznao da je dizajnirana kao asistent pri prodaji proizvoda.
3. 80% ispitanika je kao uporabnu svrhu programa predložilo ispomoć pri korištenju webshopa, dok je 20% smatralo da program ne odgovara prikladno na dovoljan broj scenarija da bi bio primjenjiv u industriji.

4.4.2 Chatbot Bernard s učitanim modelom `model.h5` i detekcijom želje za završetak razgovora

1. Treniranje modela spremljenog pod nazivom `model.h5` je pri treniranju zadnje epohe dalo rezultate vidljive u tablici 1..

funkcija gubitka	1.0184
mjera točnosti	0.1227

Tablica 4.4: Metrike kvalitete neuronske mreže

2. Utisci na razgovor s chatbotom Bernard s učitanim modelom `model.h5` i detekcijom želje za završetak razgovora su se sveli na par komentara:
   2.1. 40% ispitanika je percipiralo negativnu emocionalnu reakciju chatbot sugovornika, opisanu pridjevima *ustrašen*, *paranoičan* i *nesiguran*.
   2.2. 30% ispitanika je izrazilo svoju zabrinutost oko samosvjesnosti programa, zbog čestih izjava oblika *„Ja nisam siguran da sam ja ovdje."* (eng. *I am not sure I am here."*)
   2.3. 30% korisnika je primjetilo učestalost par istih odgovora, pogotovo kada bi korisnik postavio upitno pitanje kao unos.
   2.4. 70% ispitanika je primjetilo da su svi odgovori chabota jednostavne izjavne rečenice, te da chatbot ne postavlja nikakva dodatna pitanja kako bi aktivno sudjelovao u razgovoru.
   2.5. 40% ispitanika je izjavilo kako je iskustvo razgovora bilo dosadno.
3. 60% ispitanika je kao svrhu chatbota Bernard s učitanim modelom `model.h5` je izjavilo kako je svrha programa zabava, dok je ostatak ispitanika izjavio da ne može pojmiti svrhu programa.
4. 20% ispitanika je kao uporabnu svrhu programa predložilo izradu aplikacije za zabavu, dok je ostatak ispitanika iskazao mišljenje da je program nedovoljno razvijen za ikakvu svrhu.

4.4.3 Chatbot Bernard s učitanim modelom `model2.h5` i detekcijom želje za završetak razgovora

1. Treniranje modela spremljenog pod nazivom `model2.h5` je pri treniranju zadnje epohe dalo rezultate vidljive u tablici 1..

funkcija gubitka	1.0158
mjera točnosti	0.1238

Tablica 4.5: Metrike kvalitete neuronske mreže

2. Utisci na razgovor s chatbotom Bernard s učitanim modelom `model2.h5` i detekcijom želje za završetak razgovora su se sveli na par komentara:
   2.1. 40% ispitanika je percipiralo vrlo negativnu emocionalnu reakciju chatbot sugovornika, opisanu pridjevima *traumatiziran* i *socijalno anksiozan*.
   2.2. 90% korisnika je primjetilo ograničenost chatbota s odgovaranjem *„Ne znam, samo sam malo nervozan, to je sve."* (eng. *"I do not know, i am just a little nervous, that is all"*) na skoro svaki unos korisnika.
   2.3. 80% je izrazilo da je iskustvo razgovora bilo dosadno.
3. 40% ispitanika je izjavilo kako je svrha programa zabava, dok je ostatak ispitanika izjavio da ne može pojmiti svrhu programa.
4. 100% ispitanika je iskazalo mišljenje da je program nedovoljno razvijen za ikakvu svrhu.

4.4.4 Chatbot Bernard s učitanim modelom `model3.h5` i detekcijom želje za završetak razgovora

1. Treniranje modela spremljenog pod nazivom `model3.h5` je pri treniranju zadnje epohe dalo rezultate vidljive u tablici 1..

funkcija gubitka	0.8433
mjera točnosti	0.1427

Tablica 4.6: Metrike kvalitete neuronske mreže

2. Utisci na razgovor s chatbotom Bernard s učitanim modelom `model3.h5` i detekcijom želje za završetak razgovora su se sveli na par komentara:
   2.1. 50% ispitanika je izrazilo kako odgovori programa imaju koherentnu osobnost kroz cijeli razgovor, dok je ostatak tvrdio kako je razgovor besmisleno šarolik, i nema smisla.
   2.2. 80% ispitanika je primjetilo širi dijapazon tema o kojima mogu razgovarati.

2.3. 20% ispitanika je primjetilo humoristične odogovore od strane chatbot programa.

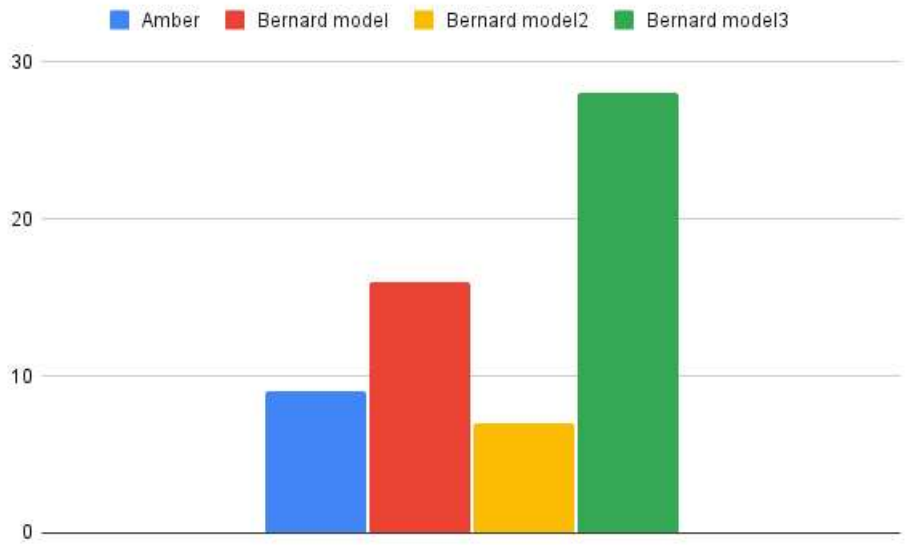
2.4. 100% ispitanika je bilo zabavljeno razgovorom.

3. 70% ispitanika je procijenilo kako je svrha programa zabava i socijalni kontakt. Ostatak je predložio pričanje izmišljenih priča kroz dijalog kao svrhu.

4. 70% ispitanika je iskazalo mišljenje da bi program mogao biti korišten za zabavu i razgovor s kronično osamljenim osobama kroz neku aplikaciju. 20% ispitanika je predložilo korištenje programa kao ispomoć pri pisanju dijaloga za knjige ili scenarija za filmove. Preostali ispitanik je izrazio mišljenje da bi program mogao biti korišten kao telefonska sekretarica.

4.4.5 Usporedba chatbot programa

Na upit koji je program ostavio otisak najljudskijeg sugovornika, 90% ispitanika je izjavilo da je razgovor chatbota Bernard s učitanim modelom `model3.h5` najviše sličio tekstualnom razgovoru sa drugim ljudskim bićem . Preostali ispitanik je tvrdi da mu nijedan razgovor nije bio dovoljno nalik na čovjeka.



Slika 4.2: Suma bodova dodijeljih rangiranim chatbot programima od strane korisnika sa obzirom na ugodnost iskustva razgovora.

Po dojmu, to jest ugodnosti iskustva korisnika, chatbot programi su rangirani od vrijednosti 0 (najlošiji dojam) do vrijednosti 3 (najbolji dojam). Usporedimo li sume vrijednosti ocjena korisnika (vidi sliku 4.2) po dojmu, vidljivo je da je chatbot Bernard s učitanim modelom `model3.h5` izglasan kao chatbot koji je većini pružio najugodnije iskustvo razgovora. On je zbog svog svojih raznovrsnijih odgovora ostavio bolji dojam no chatbot Bernard s učitanim modelom `model.h5` . S druge stane, chatbot Bernard s učitanim modelom `model2.h5` je zbog stalnog ponavljanja istog odgovora ostavio najlošiji dojam - lošiji čak i od vrlo ograničenog FAQ chatbota (vidi 1.2.2) Amber.

10% sugovornika je izrazilo da ne bi ponovilo interakciju ni s jednim chatbot programom. 60% sugovornika je izrazilo da bi rado ponovno razgovarali sa chatbotom Bernard s učitanim modelom `model3.h5` . Preostalih 20% sugovornika je izrazilo da bi rado ponovno razgovarali sa chatbotom Bernard s učitanim modelom `model.h5`.

4.5 Zaključak

Iz rezultata (vidi 4.4.5) je vidljivo da je chatbot Amber (vidi 2) od strane većine korisnika prepoznat kao chatbot napravljen sa svrhom asistencije pri kupnji proizvoda, to jest njegova namjena je bila očita van konteksta platforme za kupovinu na internetu (eng. *webshop*). S obzirom da svrha chatbota Amber nije bila zabava korisnika nego pomoć pri dovršetku zadatka, njegova popularnost i zabavljenost korisnika pri korištenju nije od važnosti. Iz odgovora korisnika je uočeno da bi za daljnji napredak modela ovog *FAQ* chatbota (vidi 1.2.2) bilo poželjno implementirati sljedeća poboljšanja:

- Dodavanje više kategorija upita kako bi chatbot program mogao korisniku dati još više vrsta informacija o proizvodu.
- Dodavanje kategorije ili modifikacija modela kategorizatora kako bi se omogućio bolji odgovor chatbot programa na korisnikove upite koji izlaze van kategorija upita koji bi mogli biti postavljeni unutar konteksta dovršavanja zadatka prodaje, to jest mehanizam povrata teme razgovora na proces prodaje.

Ispitanici su jednoglasno izrazili da su različiti modeli chatbota Bernard (vidi 3) po svrsi različiti od chatbota Amber, te ovisno o hiperparametrima treniranja (vidi tablicu 4.7) model su postignuti vidljivo različiti rezultati u dojmu korisnika na interakciju s programom.

S obzirom na metrike funkcije gubitka i parametre točnosti, `model3.h5` je pri treniranju imao najbolju (tj. najmanju) vrijednost funkcije gubitka, dok su su modeli `model.h5` i `model2.h5` imali lošije vrijednosti funkcije gubitka.

Model `model.h5` je treniran s originalnim hiperarametrima treniranja iz Tensorflow (vidi 3.2.3) tutoriala izrade transformer modela (vidi [11]), ali na puno većem broju poda-

	model.h5	model2.h5	model3.h5
$max_{samples}$	250000	250000	250000
max_{length}	40	40	40
num_{layers}	2	4	2
d_{model}	256	256	512
num_{heads}	8	8	8
$units$	512	512	512
$dropout$	0.1	0.1	0.1

Tablica 4.7: Parametri različitih treniranja transformer modela učitanih u chatbot Bernard

taka. Njegova izvedba kao zabavnog chatbota namijenjenog za socijalnu interakciju (vidi 1.2.4) je donekle zadovoljavajuća s obzirom da je 60% ispitanika točno opisalo njegovu namjenu. Međutim, većina korisnika je također izrazila da je program izrazito pasivan u razgovoru zbog prekomjernog korištenja jednostavnih rečenica, te je 80% ispitanika izrazilo da ne želi ponoviti razgovor s tim, ili ijednim, chatbot programom.

Model `model2.h5` je treniran s originalnim hiperarametrima treniranja iz Tensorflow (vidi 3.2.3) tutoriala izrade transformer modela (vidi [11]), ali na puno većem broju podataka i dvostrukim brojem podslojeva num_{layers} (vidi 3.3.1.3.2). Njegov preformans kao zabavni chatbot namijenjen za socijalnu interakciju (vidi 1.2.4) je bio nezadovoljavajuć s obzirom da je 40% ispitanika točno opisalo njegovu namjenu, a 80% korisnika je opisalo interakciju kao dosadnu. Program je tokom razgovora stalno ponavljao iste odgovore, rezultirajući time da je 100% ispitanika izrazilo da ne želi ponoviti razgovor sa tim, ili ijednim, chatbot programom. Ovo upućuje na potencijalnu *pretreniranost* (eng. *overfitting*) neuronske mreže. Pretrenirana neuronska mreža ne „prepoznaje" temeljni trend u podacima, već uči podatke „napamet", što kao rezultat pogoršava njenu moć davanja zadovoljavajućeg odgovora.

Model `model3.h5` je treniran s originalnim hiperarametrima treniranja iz Tensorflow (vidi 3.2.3) tutoriala izrade transformer modela (vidi [11]), ali na puno većem broju podataka i dvostruke dimenzije modela d_{model} (vidi 3.3.1.3.2). Njegov preformans kao zabavnog chatbota namijenjenog za socijalnu interakciju (vidi 1.2.4) je bio vrlo zadovoljavajuć s obzirom da je 70% ispitanika točno opisalo njegovu namjenu, 100% korisnika je bilo zabavljeno razgovorom te je 60% korisnika izrazilo želju da ponovno razgovara sa programom. Program je tokom razgovora mogao dati prikladan odgovor pri razgovoru o širem dijapazonu tema, no njegova sposobnost praćenja teme razgovora kroz više interakcija nije čvrsto negirana ni potvrđena.

Većina korisnika - unatoč tome što su upoznati s činjenicom da razgovaraju s programom - jest detektirala emocije i različite karakteristike osobnosti kroz odgovore Bernard

chabot programa unatoč njihovoj jednostavnosti i manjku ikakve implementacije imitacije emocija, što daje naslutiti da izbor hiperparametara utječe na „karakter" rezultantnog programa čak i ako je treniranje obavljeno nad istim skupom primjera.

U svrhu daljnjeg unaprijeđenja chatbota Bernard, s obzirom na dosadašnje rezultate, bilo bi poželjno provesti treniranje transformer modela s hiperparametrima za treniranje (vidi tablicu 4.7) modela `model3.h5` (obzirom da je taj chatbot program najpozitivnije doživljen od strane ispitanika), ali na znatno većem broju primjera za treniranje.

Zaključno, iz rezultata ispitivanja je moguće zaključiti da su chatbot programi Amber i Bernard implementirani dovoljno dobro da je njihova namjena jasna prosječnom neupućenom korisniku, ali nijedan nije u stanju uspješno imitirati ljudskog sugovornika.

Kodovi svih chatbotova prezentiranih u radu se mogu pronaći na `https://github.com/IvaTutis/Amber` i `https://github.com/IvaTutis/Bernard`.

Bibliografija

[1] *Oxford Advanced Learner's Dictionary chatbot definition*. https://www.oxfordlearnersdictionaries.com/definition/english/chatbot#:~:text=chatbot-,noun,person, (pristupljeno: 01.6.2022).

[2] A. Vaswani, N. Shazeer, N. Parmar J. Uszkoreit L. Jones A.N. Gomez Ł.Kaiser I. Polosukhin: *Attention is all you need*. Advances in neural information processing systems, 30 (2017):5998–6008.

[3] A.M.Turing: *Computing machinery and intelligence*. 2009.

[4] Anaconda, Inc.: *Anaconda distribution*. https://www.anaconda.com/products/distribution, (pristupljeno: 20.10.2022).

[5] Anaconda, Inc.: *Conda documentation*. https://docs.conda.io/en/latest/, (pristupljeno: 20.10.2022).

[6] Apache: *Apache Maven documentation*. https://maven.apache.org/, (pristupljeno: 20.10.2022).

[7] Apache: *Apache OpenNLP pretrained models*. https://opennlp.sourceforge.net/models-1.5/, (pristupljeno: 20.10.2022).

[8] ApacheOpenNLP: *Apache OpenNLP Developer Documentation*. https://opennlp.apache.org/docs/2.0.0/manual/opennlp.html, (pristupljeno: 20.10.2022).

[9] B.M.Li: *About Bryan M. Li*. https://bryanli.io/, (pristupljeno: 20.10.2022).

[10] B.M.Li: *Transformer Chatbot made with TensorFlow 2.0*. https://github.com/bryanlimy/tf2-transformer-chatbot, (pristupljeno: 20.10.2022).

[11] B.M.Li: *A Transformer Chatbot Tutorial with TensorFlow 2.0*. https://blog.tensorflow.org/2019/05/

transformer-chatbot-tutorial-with-tensorflow-2.html, (pristupljeno: 20.10.2022).

[12] B.M.Nechu: *What is an encoder decoder model?* https://towardsdatascience.com/what-is-an-encoder-decoder-model-86b3d57c5e1a, (pristupljeno 23.10.2022).

[13] C. Danescu-Niculescu-Mizil, L. Lee: *Chameleons in imagined conversations: A new approach to understanding coordination of linguistic style in dialogs*. arXiv preprint arXiv, 1106.3077 (2011):1–12.

[14] Canonical: *Enterprise Open Source and Linux*. https://ubuntu.com/, (pristupljeno: 20.10.2022).

[15] Canonical: *Generate SSH Keys on Windows 10*. https://ubuntu.com/tutorials/ssh-keygen-on-windows#1-overview, (pristupljeno: 20.10.2022).

[16] C.Danescu-Niculescu-Mizil: *Cornell Movie Dialogs Corpus*. https://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html, (pristupljeno: 20.10.2022).

[17] C.Danescu-Niculescu-Mizil: *Cornell Movie-Dialogs Corpus - convokit 2.5.3 documentation*. https://convokit.cornell.edu/documentation/movie.html, (pristupljeno: 30.10.2022).

[18] Chatdesk: *Taco Bell Slack Chatbot*. https://www.chatbotguide.org/taco-bell-bot, (pristupljeno: 09.10.2022).

[19] C.Rogers: *Client Centered Therapy*. Hachette UK, London, 2012.

[20] D. Bahdanau, K. Cho, Y. Bengio: *Neural machine translation by jointly learning to align and translate*. arXiv preprint arXiv, 1409.0473 (2014):1–15.

[21] E.A.J. Croes, M.L. Antheunis: *Can we be friends with Mitsuku? A longitudinal study on the process of relationship formation between humans and a social chatbot*. Journal of Social and Personal Relationships, 38 (2021):279–300.

[22] Eclipse: *Eclipse documentation*. https://www.eclipse.org/ide/, (pristupljeno: 20.10.2022).

[23] E.Mollick: *Establishing Moore's law*. IEEE Annals of the History of Computing, 28 (2006):62–75.

[24] F. Kaysi, M. Yavuz, E. Aydemi: *Investigation of university students' smartphone usage levels and effects*. International Journal of Technology in Education and Science (IJTES), 5 (2021):411–426.

[25] G. Paul, C. Stegbauer: *Is the digital divide between young and elderly people increasing?* First Monday, 10 (2005):60–69.

[26] G.Chrupała: *Towards a machine-learning architecture for lexical functional grammar parsing*. Dublin City University, Dublin, 2008.

[27] Gradle, Inc.: *Gradle documentation*. `https://docs.gradle.org/current/userguide/userguide.html`, (pristupljeno: 20.10.2022).

[28] H.Y. Shum, X. He, D. Li: *From Eliza to XiaoIce: challenges and opportunities with social chatbots*. Frontiers of Information Technology & Electronic Engineering, 19 (2018):10–26.

[29] J. Moskalewicz, R. Room, B. Thom: *Comparative monitoring of alcohol epidemiology across the EU*. Baseline assessment and suggestions for future action. Synthesis report. Warsaw: RARHA, 1 (2016):6–363.

[30] J.A. Ask, M. Facemire, A. Hogan H.B. Conversations: *The state of chatbots*. Forrester. com report, 20 (2016):1–16.

[31] J.C.Reynar: *Topic segmentation: Algorithms and applications*. University of Pennsylvania, Philadelphia, 1998.

[32] JetBrains: *IntelliJ IDEA documentation*. `https://www.jetbrains.com/help/idea/getting-started.html`, (pristupljeno: 20.10.2022).

[33] Jupyter: *The Jupyter Notebook documentation*. `https://jupyter-notebook.readthedocs.io/`, (pristupljeno: 20.10.2022).

[34] KDE: *Konsole: the KDE Terminal Emulator*. `https://konsole.kde.org/`, (pristupljeno: 20.10.2022).

[35] K.E.Koech: *Cross-entropy loss function*. `https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e`, (pristupljeno 25.10.2022).

[36] K.K. Fitzpatrick, A. Darcy, M. Vierhile: *Delivering cognitive behavior therapy to young adults with symptoms of depression and anxiety using a fully automated conversational agent (Woebot): a randomized controlled trial*. JMIR mental health, 4 (2017):77–85.

[37] L.M. Heinrich, E. Gullone: *The clinical significance of loneliness: A literature review*. Clinical psychology review, 26 (2006):695–718.

[38] Luka, Inc.: *The AI companion who cares*. https://replika.ai/, (pristupljeno: 13.10.2022).

[39] M.B.Hoy: *Alexa, Siri, Cortana, and more: an introduction to voice assistants*. Medical reference services quarterly, 37 (2018):81–88.

[40] NVIDIA: *CUDA toolkit*. https://developer.nvidia.com/cuda-toolkit, (pristupljeno: 20.10.2022).

[41] NVIDIA: *NVIDIA cuDNN documentation*. https://docs.nvidia.com/deeplearning/cudnn/developer-guide/index.html, (pristupljeno: 21.10.2022).

[42] Oracle: *Java documentation*. https://docs.oracle.com/en/java/, (pristupljeno: 20.10.2022).

[43] Oracle: *NetBeans documentation*. https://docs.oracle.com/cd/E40938_01/doc.74/e40142/gs_nbeans.htm, (pristupljeno: 20.10.2022).

[44] Python: *What is Python? Executive Summary*. https://www.python.org/doc/essays/blurb/, (pristupljeno: 20.10.2022).

[45] Ravik: *Create your own chat bot in Java using Apache OpenNLP*. https://tinyurl.com/ymnhsraw, (pristupljeno: 20.10.2022).

[46] R.Willy: *Pretrained lemmatizer model*. https://raw.githubusercontent.com/richardwilly98/elasticsearch-opennlp-auto-tagging/master/src/main/resources/models/en-lemmatizer.dict, (pristupljeno: 20.10.2022).

[47] S. Boiano, A. Borda, G. Gaia S. Rossi P. Cuomo: *Chatbots and new audience opportunities for museums and heritage organisations*. Electronic visualisation and the arts, 1 (2018):164–171.

[48] Spyder: *Welcome to Spyder's Documentation*. https://docs.spyder-ide.org/current/index.html, (pristupljeno: 20.10.2022).

[49] Tensorflow: *Distributed training with TensorFlow*. https://www.tensorflow.org/guide/distributed_training, (pristupljeno: 21.10.2022).

[50] Tensorflow: *Introduction to TensorFlow*. https://www.tensorflow.org/learn, (pristupljeno: 20.10.2022).

[51] Tensorflow: *Neural machine translation with a Transformer and Keras*. https://www.tensorflow.org/text/tutorials/transformer, (pristupljeno: 20.10.2022).

[52] T.Wood: *What is the softmax function?* https://deepai.org/machine-learning-glossary-and-terms/softmax-layer, (pristupljeno 23.10.2022).

[53] T.Šmuc: *web stranice kolegija Strojno Učenje*. https://web.math.pmf.unizg.hr/nastava/su/materijali/, (pristupljeno: 23.10.2022).

[54] V.Hart: *Hexaflexagons*. https://www.youtube.com/watch?v=VIVIegSt81k, (pristupljeno: 20.10.2022).

[55] Weizenbaum, J.: *ELIZA—a computer program for the study of natural language communication between man and machine*. Communications of the ACM, 9 (1966):36–45.

Sažetak

U ovom radu smo opisali koncept chatbota, to jest sustava za automatizirani generalni tekstualni razgovor sa ljudskim korisnikom. Usredotočili smo se na dizajn dva chatbota u jezicima Java i Python sa pratećim tehnologijama i bibliotekama koje implementiraju algoritme za procesiranje prirodnog jezika (NLP) algoritme i algoritme dubokog učenja. Nakon implementacije chatbota, programe smo testirali i prokomentirali njihove preformanse.

Summary

In this thesis, we described the concept of a chatbot, a software designed for automated conversation with a live human agent. We focused on designing two chatbots using Java and Python programming languages with accompanying technologies and libraries that implement natural language processing (NLP) algorithms and deep learning algorithms. After implementing the chatbota, we tested them and commented on their preformance.

Životopis

Rodena sam 14. rujna 1996. godine u Zagrebu. U istom gradu sam 2011. godine završila osnovnu školu Antuna Gustava Matoša, te upisala XV. gimnaziju. Tokom osnovnog i srednjoškolskog obrazovanja sam više puta sudjelovala u državnim natjecanjima iz predmeta Matematike, Logike i Fizike.

2015. godine, po završetku srednje škole, upisujem preddiplomski studij Matematika na Prirodoslovno-matematičkom fakultetu u Zagrebu. Dani studij sam završila 2019. godine stekavši titulu *univ. bacc. math.*, te sam iste godine upisala diplomski studij Računarstvo i matematika na istom fakultetu.

Tokom studija sam niz godina sudjelovala kao student-volonter u udruzi *Mladi Nadareni Matematičari "Marin Getaldić"*. Tamo sam u svrhu popularizacije matematike i pripreme učenika srednjih i osnovnih škola za nacionalna i internacionalna natjecanja održavala edukativne redionice i služila kao mentor studentima.

Tokom diplomskog studija, godine 2021. počinjem raditi u *ECCOS inženjering d.o.o.* na poziciji Junior Software Engineer.

Godine 2022. sam sudjelovala u projektu fakulteta financiranom iz Europskog socijalnog fonda, *ProSper PMF*. Nastavno na projekt, sudjelovala sam u šestomjesečnim edukativnim radionicama o projektima financiranima iz EU fondova u sklopu Akademije regionalnoga razvoja i fondova Europske unije.

Godine 2022. sam pohađala i predmet *Stručna praksa* na Prirodoslovno-matematičkom fakultetu u Zagrebu s temom *Konstrukcija programa za opći razgovor koristeći metode dubokog učenja*, čiji je prirodni nastavak ovaj diplomski rad.