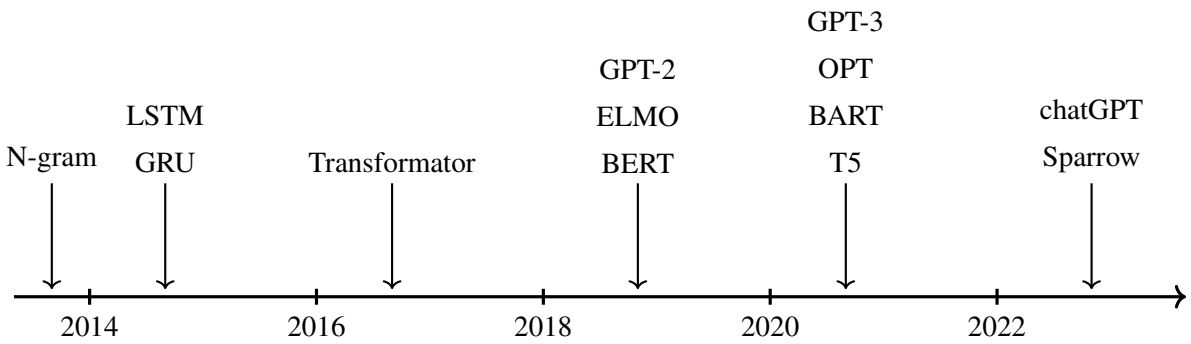
SADRŽAJ

1. Uvod

Kauzalno jezično modeliranje (engl. *causal language modeling*) teksta važan je problem obrade prirodnog jezika (engl. *natural language processing*), područja koje se bavi primjenom umjetne inteligencije na tekstne podatke. Kod jezičnog modeliranja cilj je naučiti vjerojatnosnu razdiobu rečenica i riječi prirodnoga jezika. Danas najbolje rezultate postižu duboki generativni autoregresivni modeli kod kojih predviđamo tekuću riječ pod pretpostavkom da su poznate prethodne riječi teksta.

Duboki modeli su modeli strojnog učenja bazirani na neuronskim mrežama koji sadrže više nelinearnih skrivenih slojeva. Duboki su modeli razvojem sklopovlja, prvenstveno grafičkih kartica, postali stanjem tehnike (engl. *state-of-the-art*) u mnogim potpodručjima umjetne inteligencije pa tako i u kauzalnom jezičnom modeliranju. Autoregresivni modeli su modeli koji pri generiranju koriste do sada generirane riječi, zajedno s početnim kontekstom, kao kontekst za generiranje sljedeće riječi. Na slici 1.1, uzetoj iz [2], prikazana je lenta vremena i modeli koji su u tom razdoblju bili stanje tehnike.



Slika 1.1: Duboki jezični modeli

N-gramski se model (engl. *N-gram model*) zasniva na pretpostavci da se svaka riječ u tekstu može predvidjeti na temelju n riječi koje joj prethode. Nedostatak kod takvog pristupa bila je nemogućnost praćenja duljeg konteksta. Kao odgovor na taj problem pojavile su se povratne neuronske mreže s mehanizmom dugoročne memorije (engl. *Long Short-Term Memory*, LSTM) i mehanizmom propusne povratne ćelije (engl. *Gated Recurrent Unit*, GRU), koji su omogućavali da model pamti podatke koji su "bitni", a zaboravlja one koji nisu. Nedostatak kod povratnih modela je, osim što unatoč mehanizmima za pamćenje ključnih po-

dataka imaju problema s praćenjem duljeg konteksta, mehanizam u kojem model mora čitati riječ po riječ, što jako ograničava prostor za paralelizaciju implementacije i iskorištavanja mogućnosti grafičkih kartica. Arhitektura transformatora (engl. *transformer*), zasnovana na mehanizmu samopažnje (engl. *self-attention*), riješila je oba gore navedena problema. Danas je arhitektura transformatora stanje tehnike u gotovo svim područjima obrade prirodnog jezika i u mnogim drugim područjima umjetne inteligencije.

U ovom radu odabran je predtrenirani generativni jezični model GPT-2, baziran na arhitekturi transformatora. Detaljno su ispitane njegove sposobnosti rješavanja zadataka odgovaranja na pitanja i jezičnog modeliranja. Rezultati ukazuju da model postiže dobre rezultate i bez ugađanja. Ugađanjem na nekim skupovima, rezultati su se značajno poboljšali.

U poglavlju 2 detaljno je opisana struktura svakog skupa koji je obrađen i što se njima testiralo. Poglavlje 3 opisuje arhitekturu transformatora i sve mehanizme u korištenom modelu. U Poglavlju 4 opisana je evaluacija, treniranje i rezultati nad svakim od skupova podataka.

2. Opis skupova podataka

2.1. Skup podataka *CBT-CN*

CBT-CN je podskup skupa podataka *CBT* (engl. *Children's book text*), koji sadrži tekstove iz dječjih knjiga. Skup podataka je osmišljen za mjerenje koliko dobro model može iskoristiti širi kontekst za predviđanje riječi. Svaki primjer se sastoji od skupa rečenica koje predstavljaju kontekst, rečenice u kojoj nedostaje jedna riječ i 10 ponuđenih riječi od kojih je jedna ona koja je izbačena iz ranije spomenute rečenice. Cilj je da model pogodi koja od 10 ponuđenih riječi je ona koja je izbačena. Skraćenica CN označava da je riječ koja nedostaje opća imenica (engl. *common noun*). Broj podataka po podskupu prikazan je u tablici 2.1.

Tablica 2.1: Podskupovi skupa *CBT*

Podskup	Treniranje	Validacija	Testiranje	Ukupno
CN	121k	2k	2.5k	124k
NE	109k	2k	2.5k	113k
P	339k	2k	2.5k	339k
V	106k	2k	2.5k	110k

Za učitavanje podataka korištena je HugginFace-ova biblioteka Datasets.[1] Kada se pomoću te biblioteke učitaju podaci u memoriju, svaki podatak je predstavljen Pythonovim rječnikom s ključevima `sentences`, `question`, `answer` i `options`. U tablici 2.2 prikazan je jedan primjer (kontekst je maljo dulji, pa je izostavljen).

U pitanju na mjestu označenom s pet znakova X izbačena je riječ *fairies*, a ponuđene riječi su navedene u stupcu Opcije. Cilj modela koji rješava ovaj problem predviđanja nedostajuće riječi je naučiti, uz pomoć konteksta, da se riječ *fairies* najbolje uklapa u danu rečenicu. Uspješnost modela mjeri se kao postotak primjera u kojima je model izabrao ispravnu riječ.

[1] `https://huggingface.co/`

Tablica 2.2: Primjer podatka iz skupa *CBT-CN*

Pitanje	Odgovor	Opcije
How often must I tell you that there are no XXXXX ?	fairies	People breakfast cat child fairies family pictures queen room time

2.2. Skup podataka *LAMBADA*

Skup podataka *LAMBADA* (engl. ***LA****nguage* ***M****odeling* ***B****roadened to* ***A****ccount for* ***D****iscourse* ***A****spects*) je skup pripovjednih odlomaka sakupljenih iz raznih romana. Cilj je da model na temelju danog teksta predvidi zadnju riječ u tekstu, kao kod skupa podataka CBT-CN s tom razlikom što se riječ koju ovdje predviđamo uvijek nalazi na kraju rečenice i nemamo ponuđene riječi, odnosno predviđamo bilo koju riječ iz vokabulara . Podatci u skupu za treniranje i validaciju birani su tako da je za uspješnu predikciju potrebno pratiti kontekst iz cijelog teksta, a ne recimo samo zadnju rečenicu. Podatci iz skupa za treniranje sastoje se od cijelih romana čiji odlomci nisu uključeni u skupove za validaciju i testiranje. Raspodjela podataka u podskupove za treniranje, validaciju i testiranje te prosječna duljina tekstova u svakom podskupu dana je u tablici 2.3.

Tablica 2.3: Podskupovi skupa *LAMBADA*

Metrika	Treniranje	Validacija	Testiranje	Ukupno
Broj podataka	2.66k	4.87k	5.15k	12.68k
Prosječna duljina svakog primjera u riječima	81k	74	73	17k

Po broju podataka i prosječnoj količini riječi po primjeru iz skupa, vidi se da je podskup za treniranje najveći u smislu ukupnog broja riječi, iako ima najmanje pojedinačnih primjera. Posljedica toga je da za ugađanje (engl. *fine-tuning*) modela treba imati jaču sklopovsku podršku pogodnu za treniranje velikih modela koji mogu uzeti u obzir tako širok kontekst. Primjer jednog primjera iz skupa za testiranje dan je u tablici 2.4.

U stupcu tekst dan je tekst kojem nedostaje zadnja riječ, a u stupcu odgovor nalazi se

Tablica 2.4: Jedan primjer za testiranje iz skupa *LAMBADA*

Tekst	Odgovor
"Yes, I thought I was going to lose the baby." "I was scared too," he stated, sincerity flooding his eyes. "You were ?""Yes, of course. Why do you even ask?" "This baby wasn't exactly planned for.""Do you honestly think that I would want you to have a _____?"	miscarriage

riječ koja nedostaje. Vidljivo je da zadnju riječ nije moguće predvidjeti samo na temelju zadnje rečenice, već je potrebno pratiti kontekst kroz cijeli tekst. Uspješnost modela mjeri se kao zbunjenost (engl. *perplexity*) modela na zadnjoj riječi; više o toj mjeri je rečeno u poglavlju 3.

2.3. Skup podataka *CoQA*

Skup podataka *CoQA* (engl. *Conversational Question Answering Challenge*) je skup odlomaka teksta i pitanja o danom odlomku. Podatci su skupljeni iz sedam različitih domena: dječjih priča, javno dostupne literature (s Projekta Gutenberg), srednjoškolskih ispita iz engleskog jezika, novinskih članaka (članaka CNN-a), članaka s Wikipedije, Reddita te znanstvenih članaka. Odgovor na svako od pitanja nalazi se u tekstu. Pitanja su koncipirana tako da se nadovezuju jedna na drugo, odnosno pitanja su postavljana kao da dvoje ljudi vode razgovor. Da bi model bio uspješan na ovom skupu podataka, treba s razumijevanjem pratiti tekst o kojem su pitanja te moći pratiti kontekst kroz ranije postavljena pitanja. U tablici 2.5 prikazana je raspodjela skupa u podskupove za treniranje i validaciju.

Tablica 2.5: Podskupovi skupa *CoQA*

	Treniranje	Validacija	Ukupno
Broj podataka	7199	500	7699

Ovaj skup je manji od prva dva po broju primjera. Također, bitno je napomenuti da svaki primjer sadrži oko 15 parova pitanje-odgovor, tako da u skupu ukupno ima oko 120000 parova pitanje-odgovor. U tablici 2.6 je neveden jedan skraćeni primjer s tekstom i pitanjima. Radi jednostavnosti nisu navedna sva pitanja za taj primjer nego samo njih 5. Na neka je pitanja nemoguće odgovoriti ako ne pratimo kontekst iz prošlih pitanja (recimo pitanje "and?"). Bitno je napomenuti da je u skupu za svaki odgovor dana i pozicija u tekstu gdje se odgovor nalazi. Uspješnost modela može se mjeriti preko preciznosti točnih odgovora ili preko zbunjenosti modela, više o tome u poglavlju 4.

Tablica 2.6: Primjer iz skupa *CoQA*

Tekst	Pitanje	Odgovor
"The Vatican Apostolic Library, more commonly called the Vatican Library or simply the Vat, is the library of the Holy See, located in Vatican City. Formally established in 1475, although it is much older, it is one of the oldest libraries in the world and contains one of the most significant collections of historical texts. It has 75,000 codices from throughout history, as well as 1.1 million printed books, which include some 8,500 incunabula. The Vatican Library is a research library for history, law, philosophy, science and theology. The Vatican Library is open to anyone who can document their qualifications and research needs. Photocopies for private study of pages from books published between 1801 and 1990 can be requested in person or by mail."	"When was the Vat formally opened?" "what is the library for?" "for what subjects?" "and?" "how many printed books does it contain?"	"It was formally established in 1475" "research" "history, and law" "philosophy, science and theology" "1.1 million"

3. Metode

3.1. Transformator

Arhitektura transformatora je arhitektura koder-dekoder(engl. *encoder-decoder arhitecture*), gdje se koder i dekoder zasnivaju na mehanizmu pažnje. U nastavku je objašnjena motivacija i mehanizam arhitekture koder-dekoder te ostali mehanizmi na kojima se temelji po uzoru na radove [8], [1] i [4].

3.1.1. Arhitektura koder-dekoder

Motivaciju za arhitekturu koder-dekoder je najbolje objasniti preko zadatka strojnog prevođenja teksta s jednog jezika na drugi. Ulaz u model je vektorska reprezentacija teksta na prvom jeziku a izlaz iz modela je vektorska reprezentacija teksta na drugom jeziku. Primjetimo da za uspješno prevođenje moramo u svakom trenutku znati cijeli izvorni tekst, odnosno da pristup u kojem prevodimo riječ po riječ ne bi davao dobre rezultate (npr. u jednom jeziku gramatika može propisati da glagol mora biti isključivo na početku rečenice, dok u drugom jeziku glagol ide u sredinu ili na kraj rečenice). Također, duljina izvornog i prevedenog teksta ne moraju biti jednake. Ideja na kojoj se i danas temelji stanje tehnike je da model pomoću kodera pretvori ulaznu vektorsku reprezentaciju izvornog teksta u neku apstraktnu reprezentaciju, a zatim taj apstraktni oblik dekoder pretvori u vektorsku reprezentaciju teksta na drugom jeziku. Izlaz iz kodera može biti fiksne duljine (kao kod povratnih modela s mehanizmom *GRU* i *LSTM*), a može biti i proizvoljne duljine (kao kod modela koji koriste mehanizam pažnje). Modeli koji generiraju tekst proizvoljne duljine baziraju se na dekoderskom dijelu arhitekture transformatora.

3.1.2. Pozicijsko ugrađivanje

Pozicijsko ugrađivanje (engl. *Positional embedding*) je mehanizam kojim transformator osigurava kodiranje pozicija riječi u tekstu na svom ulazu. Pretpostavimo da već imamo vektorsku reprezentaciju teksta, koja je rastavljena na k simbola (engl. *tokens*) te je svaki simbol predstavljen vektorom dimenzije d.

Pozicijsko kodiranje (engl. *Positional encoding*) simbola na i-tom mjestu određujemo tako da poizcijski vektor (engl. *positional vector*) koji njime dobijemo bude jedinstven za indeks i i da nam govori u kojem dijelu teksta se simbol nalazi. Jedan način za određivanje pozicijskog vektora je da ga definiramo kao vrijednost neke injektivne funkcije $f : \mathbb{R} \rightarrow U \subset \mathbb{R}^d$ kojoj kao argument predajemo indeks i. Također, funkcija f mora biti ograničena, odnosno njene vrijednosti uvijek moraju biti razumno velike kako bismo izbjegli prelijevanje (engl. *overflow*). Jedan primjer takve funkcije je

$$f(i)_{2j} = \sin \frac{i}{n^{\frac{2j}{d}}}$$

$$f(i)_{2j+1} = \cos \frac{i}{n^{\frac{2j}{d}}}$$

gdje je n proizvoljan broj, i indeks simbola, $2j$ označava parne pozicije u vrijednosti funkcije, a $2j + 1$ neparne.

Pozicijsko ugrađivanje svakog simbola svodi se na zbroj njegove vektorske reprezentacije i pozicijskog vektora dobivenog pozicijskim kodirajem njegove pozicije u tekstu. Izlaz je iste dimenzije kao i vektorska reprezentacija, odnosno sastoji se od k d-dimenzionalnih vektora. Cilj pozicijskog ugrađivanja je modelu dati u isto vrijeme informaciju o kojem se simbolu radi i gdje se u tekstu nalazi. To znatno povećava mogućnost paralelizacije učenja i evaluacije, jer za razliku od povratnih modela ne moramo obrađivati jedan po jedan simbol.

3.1.3. Mehanizam pažnje

Mehanizam pažnje (engl. *attention mechanism*) ključan je mehanizam za rad transformatora. Da bismo opisali princip njegovog rada, uzet ćemo ponovno za primjer problem prevođenja s jednog jezika na drugi. Kao što je rečeno ranije, takvi modeli imaju arhitekturu koder-dekoder. Problem kod povratnih modela koji koriste mehanizme *LSTM* ili *GRU* je što imaju fiksnu duljinu skrivenoga stanja, odnosno kontekst cijeloga teksta reprezentiraju vektorom fiksne duljine. Očito je da vektor fiksne duljina može sadržavati konačno mnogo informacija te se pokazalo da takvi modeli rade znatno lošije na duljim tekstovima. Ideja mehanizma pažnje jest kontekstni vektor računati kao težinsku sumu vektorskih reprezentacija svih simbola izvornog teksta. Na taj način model ima mogućnost obratiti različitu razinu pozornosti na različite dijelove ulaza s obzirom na svaki simbol ulaza.

Mehanizam pažnje u povratnoj neuronskoj mreži

U radu [1] je korišten model s arhitekturom koder-dekoder na zadatku strojnog prevođenja teksta. Koder je izveden kao dvosmjerna povratna neuronska mreža te ga nećemo dublje opisivati s obzirom da je predmet ovog rada istražiti kako model zasnovan na dekoderu generira

tekst. Bitno je da u koderu za svaki simbol x_i dobivamo skriveno stanje h_i. Dekoder radi na način da za svaki simbol na izlazu računa distribuciju vjerojatnosti prema sljedećoj formuli:

$$p(y_i|y_1, y_2, ..., y_{i-1}, \mathbf{X}) = g(y_{i-1}, s_i, c_i)$$

gdje je y_i i-ti simbol kojeg je dekoder generirao, X niz vektorskih reprezentacija izvornog teksta, c_i kontekstni vektor izračunat preko mehanizma pažnje, a s_i skriveno stanje i-tog simbola u dekoderu izračunato kao $s_i = f(s_{i-1}, y_{i-1}, c_i)$. Razlika u odnosu na modele s mehanizmom *LSTM* ili *GRU* je što se različiti kontekstni vektor računa za svaki generirani simbol na izlazu iz svakog sloja bloka dekoder. Upravo je mehanizam pažnje ključan da kontekstni vektor za generiranje svakog simbola "usmjeri pažnju" na one dijelove konteksta koji su za taj simbol "najbitniji". Računanje kontekstnog vektora c_i možemo podijeliti u tri koraka:

1. računanje sličnosti $e_{i,j}$ između prošlog skrivenog stanjas_{i-1} sa svakim stanjem kodera h_j pomoću modela poravnanja (engl. *alignment model*),
2. pretvaranje vrijednosti vektora e_i u kategoričku distribuciju α_i pomoću funkcije $softmax$,
3. računanje kontekstnog vektora kao težinske sumu skrivenih stanja kodera h_j i odgovarajućih težina $\alpha_{i,j}$.

U prvom koraku modelom poravnanja $e_{i,j} = a(s_{i-1}, h_j)$ računamo koliko ulazni simboli oko pozicije j utječu na izlazne simbole oko pozicije i (računamo sa s_{i-1} jer na temelju njega računamo distribuciju za y_i, sa s_i računamo distribuciju za y_{i+1}). U radu je za model a korištena unaprijedna potpuno povezana neuronska mreža koja se trenira zajedno s modelom. Kasnije ćemo vidjeti da model poravnavanja može biti definiran i kao običan skalarni produkt.

U drugom koraku vrijednosti $e_{i,j}$ vektora e_i pomoću funkcije $softmax$ pretvaramo u odgovarajuće kategoričke vjerojatnosti $\alpha_{i,j}$.

$$\alpha_{i,j} = \frac{\exp e_{i,j}}{\sum_{k=1}^{|X|} \exp e_{i,k}}$$

Vrijednosti $\alpha_{i,j}$ možemo shvatiti kao *vjerojatnost da se simbol y_i preveo iz simbola x_j*. Vrijednost $\alpha_{i,j}$ također možemo interpretirati kao *važnost simbola x_j* prilikom određivanja distribucije simbola y_i.

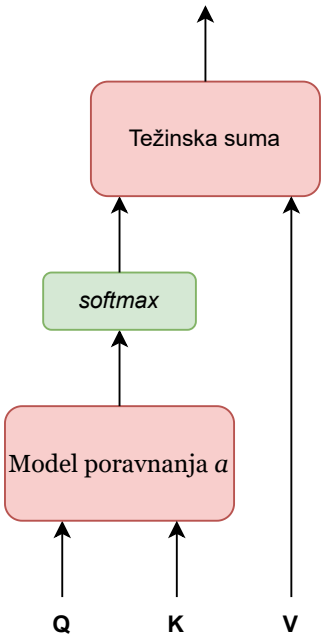
U trećem koraku računamo kontekstni vektor c_i po formuli za težinsku sumu

$$c_i = \sum_{j=1}^{|\mathbf{X}|} \alpha_{i,j} h_j.$$

Težine su vrijednosti $\alpha_{i,j}$, a vrijednosti odgovarajući vektori skrivenih stanja h_j. Težinsku sumu možemo definirati kao očekivanu vrijednost kontekstnog vektora s obzirom na distribuciju vjerojatnosti skrivenih stanja. Recimo da smo u drugom slučaju hipotetski dobili da su sve vrijednosti $\alpha_{i,j}$ približno jednake 0, osim jedne koja je približno jednaka 1. Tada bi naš kontekstni vektor bio gotovo pa jednak upravo skrivenom stanju za čiju smo vjerojatnost dobili da je približno 1. Na taj način kontekstni vektor više uvažava utjecaj skrivenih stanja onih simbola koji su bitni za određivanje distribucije sljedećeg simbola, a reducira utjecaj onih koja nisu bitna.

Mehanizam samopažnje u transformatoru

Samopažnja je mehanizam koji za svaki simbol u ulaznom tekstu računa pažnju prema svim simbolima u ulaznom tekstu. Na taj način dobivamo mjeru korelacije između svaka dva simbola na ulazu. Mehanizam samopažnje predstavljen u radu [8] generalizirao je mehanizam pažnje koji je ranije objašnjen. Shema mehanizma samopažnje prikazana je na slici 3.1.



Slika 3.1: Shema generaliziranog mehanizma pažnje

U tom mehanizmu pažnje svaki simbol ima tri vektorske reprezentacije: vektorsku reprezentaciju za upit, za ključ i za vrijednost. Kada računamo pažnju za neki simbol, uzimamo njegovu reprezentaciju za upit i provlačimo ju kroz model poravnanja s reprezentacijama za ključ svih ostalih simbola u tekstu. Nakon što na rezultate primjenimo funkciju $softmax$ i tako dobijemo odgovarajuće težine za svaki od ostalih simbola, pažnju računamo kao težinsku sumu odgovarajućih reprezentacija vrijednosti i težina. Primijetimo da je mehanizam kakav je ranije objašnjen mehanizam samopažnje u kojem su upiti posljednje skriveno stanje

dekodera, a ključevi i vrijednosti skrivena stanja kodera (odgovarajući ključevi i vrijednosti su jednaki). U transformatoru se vrijednosti upita, ključa i vrijednosti dobiju kao prikazi vektora dobivenih pozicijskim kodiranjem u različitim vektorskim prostorima. Odgovarajuće matrice prijelaza M_Q, M_K i M_V se uče tijekom treniranja. Q, K i V jednaki su

$$Q_i = P_i \times M_Q$$

$$K_i = P_i \times M_K$$

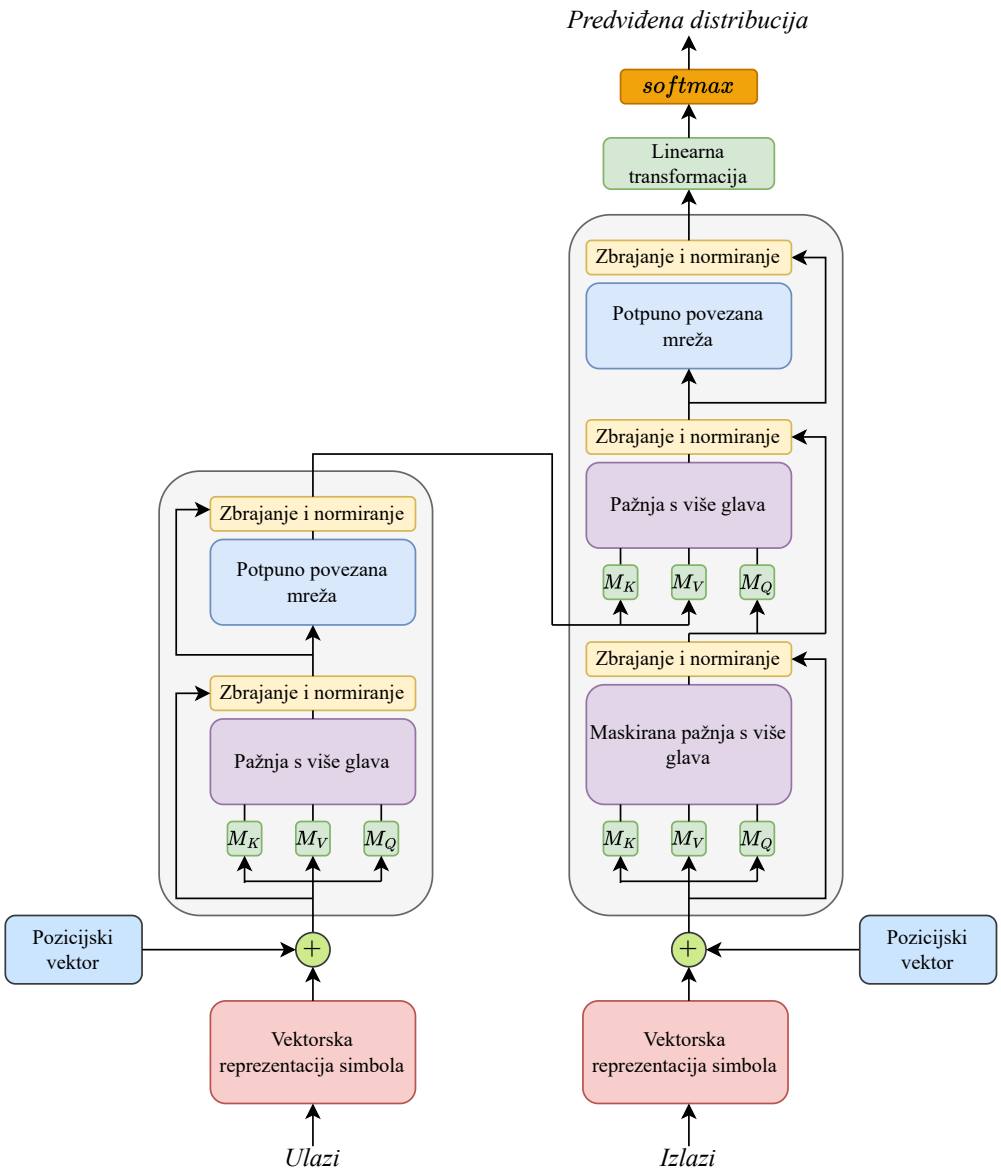
$$V_i = P_i \times M_V$$

gdje P_i označava vektor dobiven pozicijskim ugrađivanjem i-tog simbola. Za model poravnanja se u radu [8] koristi skalirani skalarni produkt.

Ranije opisani generalizirani mehanizam pažnje se u transformatoru koristi u slojevima pažnje s više glava (engl. *multi-head attention layer*). Pažnja s više glava se računa tako da treniramo više trojki matrica M_Q, M_K i M_V, sa svakom od njih računamo vektore Q, K i V te na svaku od tako dobivenih trojki primijenimo generalizirani mehanizam pažnje. Tako dobiveni vektori pažnji se spoje u jedan vektor (dimenzije *broj glava* $\times$ *dimenzija_vektora_vrijednosti*), koji se onda linearnom transformacijom prebacuje u prostor dimenzije pozicijski kodiranih vektora.

3.1.4. Arhitektura transformatora

Na slici 3.2 je prikazana arhitektura transformatora koji je treniran za strojno prevođenje teksta iz rada [8]. Model na temelju ulaznog niza i do sada generiranog izlaznog niza određuje distribuciju za sljedeći simbol na temelju predefiniranog vokabulara dostupnih simbola. Na lijevoj strani slike nalazi se koder, a na desnoj dekoder. Model iz rada se sastoji od 6 identičnih kodera i 6 identičnih dekodera. Izlaz svakog osim zadnjeg kodera je ulaz u sljedeći koder u nizu. Isto vrijedi i za dekodere. Izlaz iz zadnjeg kodera zovemo posljednjim skrivenim stanjem (engl. *last hidden state*). Na dnu kodera i dekodera se nalaze mehanizmi za pozicijsko kodiranje. Mehanizam zbraja vektorsku reprezentaciju simbola s njegovim pozicijskim vektorom te pozicijski kodirani vektor ulazi u prvi koder odnosno dekoder. Na početku svakog kodera nalazi se sloj pažnje s više glava. M_K, M_V i M_Q predstavljaju skupove matrica za različite reprezentacije vektora K, V i Q (kod pažnje s više glava računamo više trojki K, V, Q i svaku provlačimo kroz mehanizam pažnje). Na izlazu iz sloja pažnje dobiveni vektor zbrojimo s ulazom u koder te rezultat normiramo. Tako dobiveni rezultat ulazi u unaprijednu potpuno povezanu neuronsku mrežu, te njen izlaz zbrajamo s vektorom koji smo stavili na ulaz mreže. Rezultat normiramo i šaljemo na izlaz kodera (izlaz je skriveno stanje). U dekoderu za pozicijski kodirani vektor ponovno računamo odgovarajuće reprezentacije K, V i Q te ih šaljemo na ulaz sloja maskirane pažnje. Sloj maskirane pažnje je isti kao i sloj pažnje



Slika 3.2: Shema arhitekture transformatora (prilagođeno iz [8])

u koderu s time da težine simbola koji se nalaze desno od simbola koji trenutno obrađujemo prije ulaza u $softmax$ postavljamo na $-\infty$. Time postižemo da simboli generirani nakon trenutnog simbola imaju težinu 0 prije računanja težinske sume, odnosno neće nikako utjecati na formiranje kontekstnog vektora tog simbola (ovo nam je naravno bitno samo kada je poznat cijeli izlazni niz, recimo kod treniranja). Na izlazu iz sloja maskirane pažnje, kao i kod kodera, zbrojimo ulaz u dekoder s izlazom iz sloja pažnje i normiramo rezultat. Sljedeći sloj pažnje u dekoderu isti je kao i sloj pažnje u koderu (nije maskiran). U ovom sloju računamo pažnju simbola na izlazu naspram simbola na ulazu. Kontekstne vektore ulaza, odnosno posljednje skriveno stanje, reprezentiramo kao ključeve i vrijednosti, a izlaz iz prošlog sloja pažnje u dekoderu reprezentiramo kao upit. Tako dobiveni izlaz iz sloja zbrojimo s izlazom iz prošlog sloja pažnje i normiramo. Na kraju, kao i kod kodera, vektor iz posljednjeg sloja

pažnje prolazi kroz potpuno povezanu mrežu, izlaz se zbraja s vektorom iz zadnjeg sloja pažnje i normira. Tako dobiveni vektor izlazi iz dekodera. Kod posljednjeg u nizu dekodera, izlaz se linearno transformira, prolazi kroz funkciju $softmax$, te se dobiva distribucija za sljedeći simbol.

Glavna prednost arhitekture transformatora naspram povratnih modela jest mogućnost paralelizacije. Kada ulazni niz prolazi kroz koder, nema nikakvih međuovisnosti, ako imamo dovoljno računske snage, možemo izračunati posljednje skriveno stanje za sve simbole istovremeno. Također, mehanizam pažnje u transformatoru se pokazao učinkovitijim naspram onog predloženog u radu [1]. Postoje razne varijante transformatora za obradu prirodnog jezika, ugrubo ih možemo podijeliti na one koji sadrže samo kodere, samo dekodere te i kodere i dekodere. Također ih možemo razlikovati po tome na kakvim su zadatcima trenirani.

3.2. Generativni jezični model

Generativni jezični modeli s arhitekturom transformatora sastoje se najčešće samo od dekodera. U nastavku su detaljnije objašnjeni po uzoru na rad [4]. Obično su trenirani na zadatcima kauzalnog jezičnog modeliranja (engl. *causal language modeling*). U tom zadatku, model treba predvidjeti svaku riječ u rečenici na temelju svih prethodno generiranih riječi. Model na izlazu može davati vjerojatnosnu distribuciju za sljedeći simbol, ili može davati neskalirane vrijednosti koje zovemo logiti te koje onda možemo primjenom funkcije $softmax$ interpretirati kao vjerojatnosti. Vjerojatnost koju dobijemo na izlazu jednaka je

$$P(x_i|x_1, x_2, ..., x_{i-1})$$

gdje je x_i simbol na i-tom mjestu, a prije njega su poznati svi simboli na pozicijama $1, 2, \ldots, i-1$. Simboli mogu biti zadani, a mogu biti i generirani od strane modela. Uspješnost modela možemo mjeriti kao točnost ispravno generiranih riječi ili pomoću pogreške unakrsne entropije (engl. *cross-entropy error*).

3.2.1. Gubitak i pogreška unakrsne entropije

U ovom pododjeljku formalizirat ćemo kriterij učenja na temelju onovnih pojmova iz teorije informacije po uzoru na udžbenik iz dubokog učenja [3].

Količina informacije, entropija i *KL*-divergencija

Količina informacije (engl. *self-information*) slučajnog događaja je mjera koja nam govori koliko taj događaj nosi informacije s obzirom na vjerojatnost tog događaja. Na primjer, gotovo sigurni događaj ne donosi nam puno informacije jer smo i očekivali da će se dogoditi.

S druge strane događaj koji je malo vjerojatan nosi veliku količinu informacije, jer za njega *nismo baš očekivali* da će se dogoditi. Sukladno tome, količina informacije događaja x definira se kao $I(x) = -\log P(x)$. Shannonova entropija (engl. *Shannon entropy*) je mjera koja definira količinu neizvjesnosti u nekoj vjerojatnosnoj distribuciji kao očekivanje količine informacije slučajnog događaja iz te distribucije. Shannonova entropija za kategoričku vjerojatnosnu distribuciju P definira se kao

$$H(P) = \mathbb{E}_{x\sim P}[I(x)] = -\mathbb{E}_{x\sim P}[\log P(x)] = -\sum_{x\sim P} P(x)\log P(x)$$

gdje $\mathbb{E}_{x\sim P}$ označava očekivanje koje se računa po varijabli x iz distribucije P. Unakrsna entropija (engl. *cross entropy*) definira se kao očekivanje količine informacije prema nekoj distribuciji Q za slučajan događaj koji dolazi iz distribucije P. Drugim riječima, unakrsna entropija govori nam koliku će neizvjesnost imati distribucija Q, ako događaji koje ona opisuje dolaze iz distribucije P. Unakrsna entropija distribucija P i Q definira se kao

$$H(P, Q) = -\mathbb{E}_{x\sim P}[\log(Q(x))] = -\sum_{x\sim P} P(x)\log Q(x).$$

Kullback-Leiblerova divergencija (engl. *Kullback-Leibler divergence*) (ili kraće *KL*-divergencija) je mjera koja nam govori koliko se distribucije P i Q koje opisuju isti skup događaja razlikuju. Definira se kao očekivanje log-omjera vjerojatnosti kao što slijedi:

$$D_{KL}(P||Q) = \mathbb{E}_{x\sim P}[\log \frac{P(x)}{Q(x)}] = \mathbb{E}_{x\sim P}[\log P(x) - \log Q(x)].$$

KL-divergenciju možemo izraziti na temelju veličina koje smo objasnili do sada:

$$D_{KL}(P||Q) = E_{x\sim P}\log P(x) - E_{x\sim P}\log Q(x) = H(P, Q) - H(P).$$

Prikazani izraz opisuje vezu između *KL*-divergencije distribucija P i Q, unakrsne entropije distribucija P i Q te entropije distribucije P. Važna posljedica te veze je da minimiziranjem unakrsne entropije između distribucija P i Q po distribuciji Q minimiziramo *KL*-divergenciju između distribucija P i Q (ako optimiramo samo distribuciju Q, entropija distribucije P se ne mijenja). Drugim riječima, ako distribuciju P želimo aproksimirati distribucijom Q, ima smisla minimizirati $D_{KL}(P||Q)$, a tome ekvivalentan problem je minimizirati unakrsnu entropiju $H(P, Q)$. Upravo to je motivacija da definiramo funkciju gubitka kao unakrsnu entropiju distribucije koju želimo da model aproksimira i distribucije modela.

Gubitak i pogreška unakrsne entropije

Gubitak unakrsne entropije za k klasa s mekim oznakama (engl. *soft labels*) definiramo kao

$$L(x, y, model) = -\sum_{i=1}^{k} y_i \log P_{model}(c_i|x)$$

gdje x predstavlja podatak za koji računamo gubitak, y ispravnu distribuciju podatka x, y_i vjerojatnost i-te klase za simbol x, c_i i-tu klasu, a P_{model} distribuciju koju je predvidio model. Kod klasifikacijskog problema s tvrdim oznakama (engl. *hard labels*) vjerojatnost svih klasa osim ispravne je jednaka 0, pa gubitak prelazi u oblik

$$L(x, y, model) = -\log P_{model}(y|x)$$

gdje y predstavlja klasu podatka x. Na temelju tako izračunatih gubitaka, računa se funkcija pogreške, koja je jednaka očekivanju funkcije gubitka te ju zovemo cijena (ili pogreška) unakrsne entropije (engl. *cross entropy cost*). Cijena, odnosno pogreška unakrsne entropije definirana je kao

$$C(X, Y, model) = -\mathbb{E}_{x,y\sim\hat{P}_{XY}}[\log P_{model}(y|x)] = -\frac{1}{n}\sum_{i=1}^{n}\log P_{model}(Y_i|X_i)$$

gdje $\mathbb{E}_{x,y\sim\hat{P}_{XY}}$ označava očekivanje koje se računa prema združenoj distribuciji podataka X i oznaka Y (koja odgovara uniformnoj distribuciji preko skupa za učenje), X_i i-ti podatak u skupu podataka X te Y_i klasu i-tog podatka u skupu X.

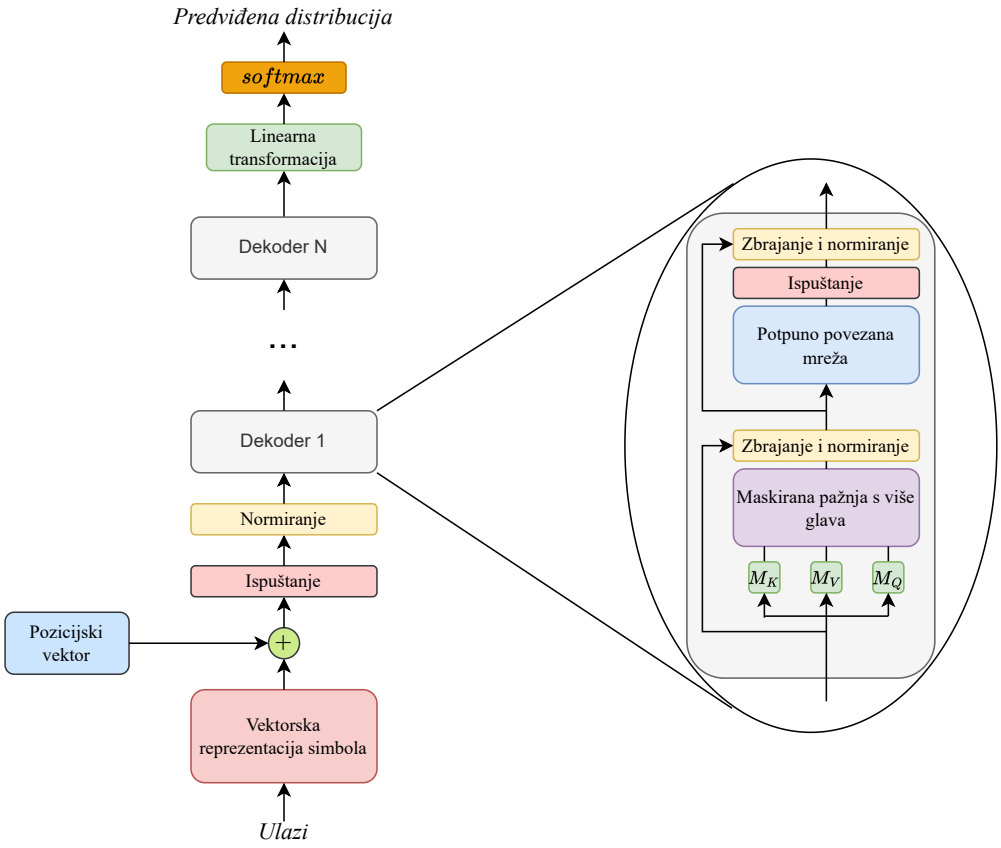
3.2.2. Treniranje generativnih modela

Prilikom treniranja model čita simbol po simbol, za svaki simbol računa gubitak te na temelju izračunatih gubitaka računa cijenu za taj primjer. Na temelju izračunate cijene optimiraju se parametri modela. Na taj način model učimo da distribuciju sljedećeg simbola za svaki podniz u tom primjeru prilagođava *one hot* distribuciji koja ima vrijednost 1 na simbolu koji dolazi nakon tog podniza. Drugim riječima, učimo model da predviđa simbole iz teksta na temelju simbola koji se nalaze prije njih u tekstu.

3.3. GPT-2

GPT-2 (engl. *generative pre-trained transformer*) je generativni jezični model razvijen 2019. od strane tvrtke OpenAI te su njegove sposobnosti prezentirane u radu [7]. Model ima gotovo identičnu arhitekturu kao i njegov prethodnik GPT-1. Također, model ima puno više parametara (najveća verzija 1.5 milijardi naspram 117 milijuna u GPT-1) i treniran je na puno više podataka (40GB tekstnih podataka naspram 4.5GB kod GPT-1). Na slici 3.3 prikazana je arhitektura modela GPT. Postoje 4 inačice modela GPT-2 koje se razlikuju po broju dekodera, dimenziji vektora s kojima radi i broju parametara (verzije imaju 12, 24, 36 i 48 dekodera).

Model je vrlo sličan dekoderu transformatora iz rada [8]. Na ulaz modela se daje tekst na temelju kojeg model treba predvidjeti vjerojatnosnu distribuciju sljedećeg simbola. Kada se model koristi za generiranje teksta, simbole koje model predvidi možemo dodati na kraj



Slika 3.3: Shema arhitekture modela GPT

teksta, ponovo ih vratiti na ulaz modela i tako generirati simbol po simbol. Razlika između dokedera GPT-a i prvog jezičnog transformatora iz rada [8] jest što GPT-ov dekoder ima samo jedan (maskirani) sloj pažnje. Drugi sloj pažnje generativnom modelu nije potreban zato što generativni model ne razlikuje simbole s ulaza i izlaza, odnosno simbole koje generira možemo smatrati novim ulaznim simbolima. Još jedna razlika jest sloj za ispuštanje (engl. *dropout layer*), koji nasumično postavlja elemente vektora koji u njega ulaze na 0. Svrha ispuštanja neurona jest regularizacija modela kako bi se spriječila prenaučenost. Eksperimenti u ovom radu su zbog nedostatka računalnih resursa rađeni s najmanjom od četiri verzije GPT-2 (12 dekodera, 124 milijuna parametara).

4. Modeliranje, eksperimenti i evaluacija

4.1. Treniranje i evaluacija modela GPT-2

Za potrebe ovog rada korištena je implementacija modela GPT-2 iz *HuggingFace*-ove biblioteke *Transformers*. Konkretno, korištena je izvedba *GPT2LMHeadModel* (engl. *GPT2 language modeling head model*) kod koje je linearni sloj na izlazu povezan s kodiranjem simbola tako da daje vjerojatnosnu distribuciju sljedećeg simbola. Izvedbe modela u biblioteci *HuggingFace* sadrže mnogo metoda koje su izvan okvira ovog rada. Svi eksperimenti su provedeni u oblaku (engl. *cloud*) na stranici *Kaggle* te je korištena grafička kartica Nvidia Tesla V100 sa 16GB radne memorije. Eksperimenti prvo ispituju performance predtreniranog modela na javno dostupnim skupovima podataka za jezično modeliranje i odgovaranje na pitanja bez ugađanja, a zatim ugađaju model na podatcima za treniranje te ispituju performancu ugađanog modela.

4.1.1. Treniranje

Kada se model ugađa podatci se grupiraju u mini-grupe (engl. *mini-batches*). Veličina grupe je jedan od hiperparametara učenja. Njome se definira koliko će podataka istovremeno biti obrađeno. Zbog ograničenih računalnih resursa, veličina grupe je uvijek bila postavljena na 4 (za veću veličinu grupe potrebna je veća količina radne memorije u grafičkoj kartici). Ostali hiperparametri koji su mijenjani su broj epoha, početni korak učenja, broj koraka za zagrijavanje, epsilon parametar optimizatora AdamW[1] i frekvencija kontrolnog ispisa (svakih koliko grupa se ispisuje prosječni gubitak). Za optimizator je u svim eksperimentima korištena *HuggingFace*-ova implementacija optimizatora AdamW. Također, korištena je *HuggingFace*-ova implementacija linearnog planera s koracima za zagrijavanje (engl. *linear scheduler with warmup steps*). Prilikom računanja gubitka su sve varijable i numerički izračunati gradijenti zapisivani kao 16-bitni brojevi s pomičnim zarezom kako bi mogli imati veću veličinu grupa (bez te pretvorbe model je znao zauzeti previše radne memorije na grafičkoj kartici kada bi veličina grupe bila 4). Grupe su stvarane, uzimane nasumično i redoslije-

[1] Loshchilov i Hutter [6].

dom u skupu pomoću klasa DataLoader, RandomSampler i SequentialSampler iz biblioteke PyTorch. Pseudokod za treniranje u svim eksperimentima je prikazan u algoritmu 1.

Algorithm 1 Pseudokod za treniranje

for $i = 0$ do $broj\_epoha$ **do**
  for all $grupe\_u\_skupu\_za\_ucenje$ **do**
    $model.ponisti\_gradijent()$
    $optimizator.ponisti\_gradijent()$
    $gubitak \leftarrow model(grupa)$
    $ukupni\_gubitak \leftarrow ukupni\_gubitak + gubitak$
    $gubitak.propagacija\_unatrag()$
    $optimizator.korak()$
    $planer.korak()$
    if $broj\_grupe \mod kontolni\_ispis = 0$ **then**
      $ispisi\_prosjecni\_gubitak()$
    end if
  end for
  $evaluiraj\_na\_skupu\_za\_validaciju()$
end for

Primjeri za treniranje su bili predstavljeni posebnom klasom u kojoj se pamtili vektorska reprezentacija teksta, oznake za taj podatak i maska pažnje (engl. *attention mask*), te se na temelju tih vrijednosti može samo pozvati model s odgovarajućim parametrima te čitavo vrijeme imati isti kod za treniranje.

Model se na svakom skupu podataka prvo evaluirao bez učenja (engl. *zero-shot evaluation*), zatim je ugađan na skupu podataka za treniranje te je na kraju evaluiran ugađani model na skupu za testiranje.

4.1.2. Evaluacija

Najvažnija metoda koja je korištena za evaluiranje je metoda `__call__()`, koja nadjačava metodu `forward()`. Metoda `forward()` *provlači* zadani tekst reprezentiran kao niz identifikatora simbola kroz cijeli model i vraća rječnik s različitim informacijama o izlazu iz modela. Najbitniji izlazi modela su pod ključem `logits`, koji predstavlja vrijednost svakog neurona prije ulaza u sloj $softmax$-a te vrijednost pod ključem `loss` koji predstavlja gubitak modela izračunat kao gubitak unaksrsne entropije. Da bi model mogao odrediti gubitak, potrebno mu je dati točne oznake (engl. *labels*) svakog simbola u tekstu za koji treba računati gubitak. Model će na temelju svake oznake "odrezati" dio teksta koji mu je potreban za određivanje distribucije vjerojatnosti za tu oznaku (odnosno uzeti će sve simbole koji se

nalaze lijevo od simbola za koji računa vjerojatnost) te na temelju vjerojatnosti izračunate za svaku od oznaka izračunati gubitak. Oznake na simbolima za koji nisu uključeni u računanje gubitka se dogovorno postavljaju na -100. Na primjer, oznake za niz identifikatora simbola `[101, 146, 1686, 1107, 13203, 106, 102]` za čiji se peti i šesti simbol računa gubitak se označavaju vektorom `[-100, -100, -100, -100, 13203, 106, -100]`.

4.2. Skup podataka *CBT-CN*

4.2.1. Evaluacija

Skup podataka *CBT-CN* smo evaluirali tako da bismo oznake riječi koju treba umetnuti do kraja postavili na vrijednosti odgovarajućih identifikatora simbola, a ostale oznake na -100. Na primjer, ako tekst dobiven spajanjem konteksta i pitanja završava rečenicom "Danas je jako lijep XXXXX pa ću se prošetati." te riječ koju treba umetnuti je "dan", pod pretpostavkom da svaka riječ odgovara jednom simbolu, vektor oznaka bi izgledao ovako: `[-100, ..., -100, dan, pa, ću, se, prošetati]` (zbog jasnoće je kod svake riječi zamijenjen s riječi koju predstavlja). Model s tako zadanim oznakama računa prosječni gubitak za svaku od riječi koju model mora generirati počevši od riječi koju biramo. Dakle, za ranije navedeni primjer, model bi prvo računao gubitak za riječ "dan", pa za riječ "pa", itd. te na kraju vratio prosjek tako izračunatih gubitaka. Time se uzima i lijevi i desni kontekst u obzir pri odabiru riječ s najmanjim prosječnim gubitkom. Primjere dulje od 1024 simbola smo preskakali jer je to ograničenje za broj simbola modela GPT-2. Izračunati prosječni gubitak jednak je

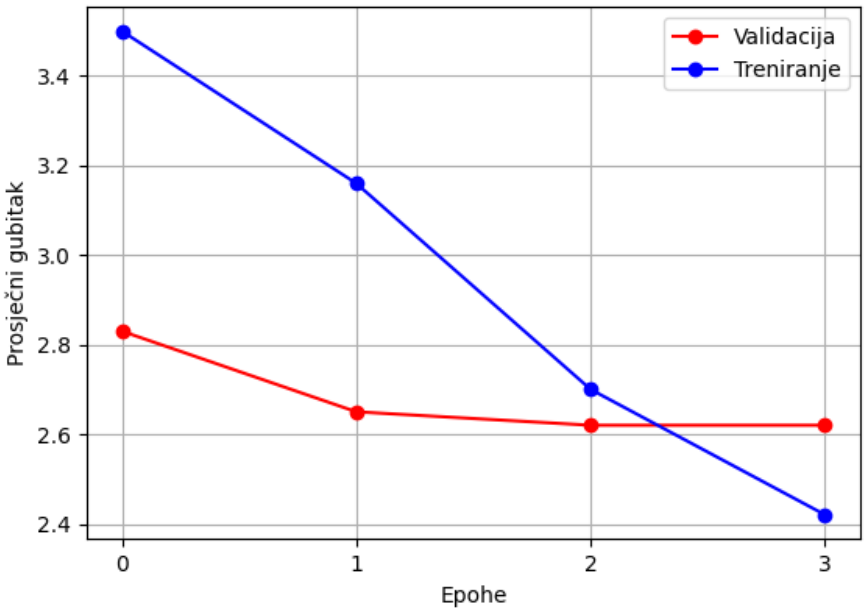
$$L_{ukupni} = \frac{1}{d}\sum_{i=k}^{n} L_i = \frac{1}{d}\sum_{i=k}^{n} -\log P(X_i|X_{<i})$$

gdje je k indeks prvog simbola koji trebamo odabrati, L_i gubitak i-tog simbola, n broj simbola na ulazu, $d = n - k$ broj simbola za koje računamo gubitak, X_i simbol na i-tom mjestu te $X_{<i}$ oznaka za skup svih simbola koji se nalaze prije simbola na indeksu i. Odluka o tome koju je riječ model odabrao se donosi tako da se odabere riječ među ponuđenim riječima za koju je gubitak najmanji. Uspješnost modela se mjeri preciznošću odabira riječi koja nedostaje.

4.2.2. Treniranje

Za treniranje je zbog ograničenosti računalnih resursa iz skupa za treniranje slučajno odabrano oko 30000 podataka ne duljih od 1024 simbola. Model se trenirao tako da bi na mjesto predviđeno za umetanje riječi bila umetnuta riječ koja nedostaje te se model učio da

predviđa tekst od te riječi pa do kraja rečenice, jer nije uvijek bio slučaj da je riječ koju treba predvidjeti na kraju ulaznog niza simbola. Model se trenirao kroz tri epohe sa sljedećim hiperparametrima [`epochs=3, learning_rate=5e-5, warmup_steps=1e2, epsilon=1e-8, batch_size=4`]. Na slici 4.1 prikazano je kako se prosječni gubitak na skupu za treniranje i validaciju ponašao kroz trening.



Slika 4.1: Treniranje modela na podskupu od 30000 primjera

Prije treniranja, prosječni je gubitak na skupu za treniranje iznosio oko 3.5, a na skupu za validaciju oko 2.8. Nakon prve epohe, prosječni je gubitak značajno pao i na skupu za treniranje i na skupu za validaciju. Nakon druge i treće epohe, prosječni je gubitak na skupu za treniranje nastavio padati, a na skupu za validaciju je ostao približno jednak. Treniranje je provedeno u tri epohe jer se prosječni gubitak nije značajnije mijenjao nakon druge i treće epohe, a postojala je mogućnost da se model prenauči na skupu za treniranje.

4.2.3. Rezultati

Rezultati se navode kao točnost modela na skupu na testiranje. Model je bez ugađanja na skupu za testiranje ispravno predvidio riječ koja nedostaje u 83.6% slučajeva. Nakon ugađanja, model je točno predvidio riječ koja nedostaje u 84.3% slučajeva. Točnost se nakon ugađanja povećala za oko 0.7%. Vjerojatno bi rezultati bili bolji da se model trenirao na više podataka, no zbog ograničenih resursa to nije bilo moguće.

4.3. Skup podataka *LAMBADA*

4.3.1. Evaluacija

Skup podataka *LAMBADA* je možda i najjednostavniji za evaluaciju. Potrebno je samo izračunati gubitke za simbole koji predstavljaju posljednju riječ u ulaznom tekstu. Dakle sve oznake je potrebno postaviti na -100 osim onih koje predstavljaju zadnju riječ (može ih biti i više od jedne). U eksperimentima su za primjere dulje od 1024 simbola uzimana posljednja 1024 simbola. Uspješnost modela se mjerila mjerom zbunjenosti koja se računa kao

$$perplexity(L) = \exp\left(\overline{L}\right)$$

$$\overline{L} = \frac{1}{N}\sum_{i=1}^{N} L_i = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{d_i}\sum_{j=k_i}^{n_i} -\log P(X_{i,j}|X_{i,<j})$$

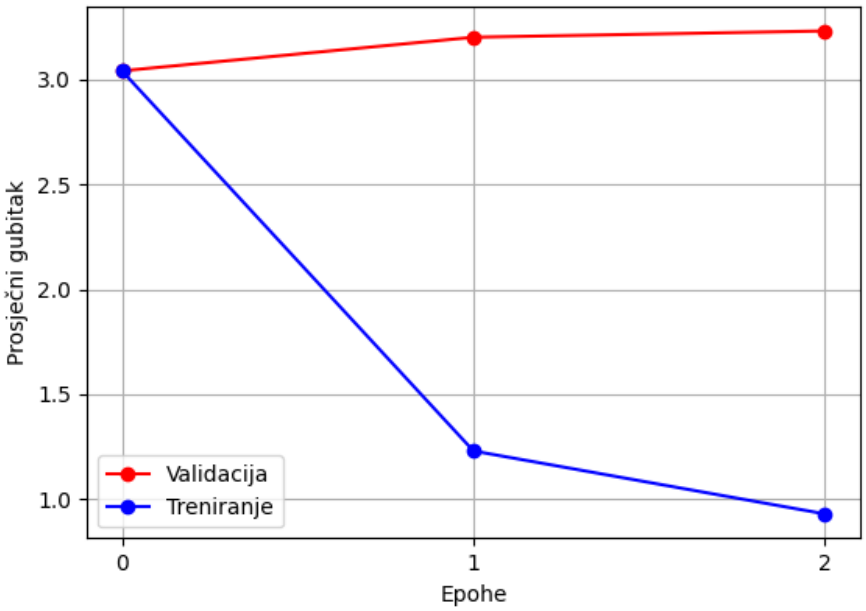
gdje je L lista izračunatih gubitaka, N broj primjera, L_i i-ti element liste L, n_i duljina i-tog primjera, k_i indeks prve oznake različite od -100 u i-tom primjeru, $d_i = n_i - k_i$ broj oznaka za koje treba računati gubitak u i-tom primjeru, $X_{i,j}$ j-ti simbol u i-tom podatku te $X_{i,<j}$ oznaka za sve simbole u i-tom primjeru koji se nalaze prije simbola na j-tom mjestu.

4.3.2. Treniranje

Prosječna duljina teksta na skupu za treniranje je 81000 riječi dok model iz eksperimenata može u isto vrijeme raditi s 1024 simbola te su iz tog razloga podatci za treniranje morali biti uzorkovani. Model se trenira tako da ispravno predviđa zadnju riječ u svakom uzorkovanom primjeru.

Naivno uzorkovanje

Naivni pristup koji se pokazao kao nedovoljno dobar je iz svakog primjera uzeti zadnja 1024 simbola te učiti model da na temelju njih predviđa zadnju riječ u tekstu. Nedostatak tog pristupa je što se njime dobiva samo 2660 primjera za treniranje, dok skupovi za testiranje i validaciju zajedno sadrže gotovo 10000 primjera. Također, ovakav pristup dovodi do promjene u distribuciji podataka između podataka za treniranje i testiranje (distribucija posljednjih riječi u romanu ne mora biti jednaka distribuciji riječi u slučajno odabranom tekstu). Model je treniran na tako dobivenom skupu kroz dvije epohe sa sljedećim hiperparametrima: `{epochs=2, learning_rate=1e-5, warmup_steps=1e2, epsilon=1e-8, batch_size=4}`. Treniranje je provedeno u dvije epohe s ciljem da se vidi hoće li gubitak na skupu za validaciju padati (da je gubitak padao, proveli bismo ponovno treniranje s istim hiperparametrima u više epoha).



Slika 4.2: Treniranje na naivno generiranim podatcima

Na slici 4.2 prikazano je kako se prosječni gubitak na skupu za treniranje i validaciju ponašao kroz trening.

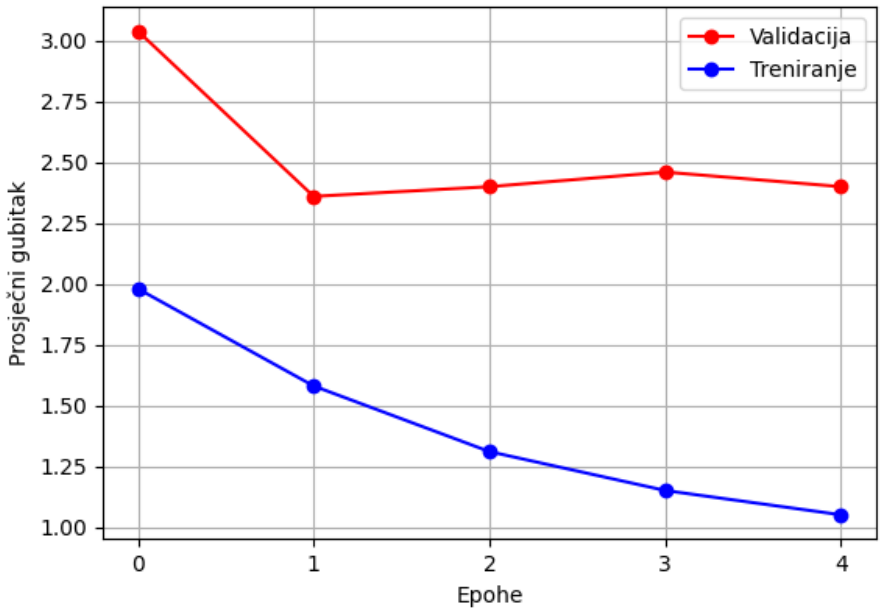
Prije treniranja je prosječni gubitak bio približno jednak na skupu za treniranje i na skupu za validaciju. Nakon prve epohe prosječni se gubitak na skupu za validaciju blago povećao, a na skupu za treniranje značajno smanjio. Slično je bilo i nakon druge epohe; prosječni se gubitak na skupu za validaciju blago povećao, a na skupu za treniranje opet značajno smanjio (ovaj put malo manje nego nakon prve epohe, što je i logično jer se korak optimizatora smanjivao nakon svake iteracije). To govori da se model prenaučio (engl. *overfitting*) na podatcima za treniranje te nije poboljšao (dapače, pogoršao je) performancu na skupu za validaciju. Budući da se performanca na skupu za validaciju nije poboljšala, nije imalo smisla model trenirati u više epoha.

Nasumično uzorkovanje

Iz prvog eksperimenta se može zaključiti da za treniranje treba više podataka te da podatci trebaju biti uzeti iz različitih dijelova romana kako ne bismo imali problem s distribucijom riječi. Drugi eksperiment je proveden tako da je svaki tekst razdvojen u rečenice pomoću funkcije `sent_tokenize()` iz biblioteke `nltk`.[2] Iz tako dobivenih lista rečenica je iz svakog primjera stvoreno po 13 primjera za testiranje. Primjeri su stvarani tako da se slučajno izabere jedna rečenicu nakon nje uzme određen broj drugih rečenica. Bu-

[2] https://www.nltk.org/api/nltk.tokenize.html

dući da se primjeri za validaciju i testiranje sastoje od 5 do 10 rečenica, uzimano je po 7 rečenica nakon prve slučajno odabrane rečenice. Riječ koja se predviđa je posljednja riječ u posljednjoj odabranoj rečenici. Bitno je napomenuti da su primjeri kojima se zadnja riječ ne sastoji isključivo od slova abecede preskakani. Na taj je način dobiveno oko 34500 primjera za treniranje. Treniranje je provedeno na isti način kao i u prvom eksperimentu sa sljedećim hiperparametrima: `{epochs=4, learning_rate=1e-5, warmup_steps=1e2, epsilon=1e-8, batch_size=4}`. Na slici 4.3 prikazano je kako se mijenjao prosječni gubitak na skupovima za treniranje i validaciju prilikom treniranja na nasumično uzorkovanim podatcima. Prosječni je gubitak na skupu za validaciju prije treniranja iznosio oko 3.0, a na skupu za treniranje oko 2.0. Nakon prve epohe prosječni se gubitak i na skupu za testiranje i na skupu za validaciju značajno smanjio. Nakon druge i treće epohe prosječni je gubitak na skupu za validaciju blago rastao, dok se na skupu za treniranje i dalje značajno smanjivao. Nakon četvrte epohe, odnosno na kraju treniranja, prosječni se gubitak i na skupu za validaciju i na skupu za treniranje blago smanjio u donosu na treću epohu.



Slika 4.3: Treniranje na nasumično uzorkovanim podatcima

4.3.3. Rezultati

Uspješnost modela smo mjerili zbunjenošću i gubitkom modela. Rezultati na skupu za testiranje dani su u tablici 4.1.

Rezultati modela iz prvog eksperimenta su se pogoršali nakon treniranja, što smo mogli i

Tablica 4.1: Rezultati na skupu za testiranje skupa podataka *LAMBADA*

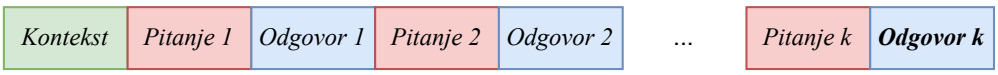
Metrika	*Zero-shot*	Prvi eksperiment	Drugi eksperiment
Prosječni gubitak	3.36	4.29	2.74
Zbunjenost	28.76	73.19	15.61

očekivati s obzirom da je prosječni gubitak na skupu validaciju tokom treniranja bio sve veći nakon svake epohe. Model iz drugog eksperimenta je postigao znatno bolje rezultate nakon treniranja. Performanca bi se vjerojatno još poboljšala kada bi se malo promijenili hiperparametri modela (prosječni je gubitak nakon druge i nakon treće epohe lagano rastao na skupu za validaciju) ili povećao skup podataka za treniranje.

4.4. Skup podataka *CoQA*

4.4.1. Evaluacija

Skupovi podataka u kojima model odgovara na pitanja se mogu evaluirati na više načina. Možda najintuitivniji način je postotkom točnih odgovora na pitanja. Drugi način je računati gubitak i zbunjenost, kao što je napravljeno na skupu podataka *LAMBADA*. S obzirom da ovaj rad proučava modelom za jezično modeliranje, najsmislenije ga je evaluirati preko gubitka i zbunjenosti. Primjeri su evaluirani tako da bi se od svakog primjera stvorlo N primjera, gdje je N broj parova pitanje-odgovor za taj primjer. Za svaki od parova bi se izračunao kontekst, koji bi se sastojao od početnog konteksta te parova pitanja i odgovora koja su dolazila prije pitanja za koje se stvara kontekst. Na tako dobiveni kontekst se dodalo naše pitanje i odgovor. Također, obraćena je pozornost na to da u svakom pitanju i odgovoru početno slovo uvijek bude veliko te da svako pitanje završava znakom upitnik. Na kraju su postavljene oznake za taj primjer tako da model računa gubitak isključivo za odgovor na zadnje postavljeno pitanje (odnosno ono za koje je i računan kontekst). Na slici 4.4 prikazano je kako bi izgledao primjer za k-ti par pitanje-odgovor u nekom neobrađenom primjeru. Podebljani tekst označava dio teksta za koji se računa gubitak.



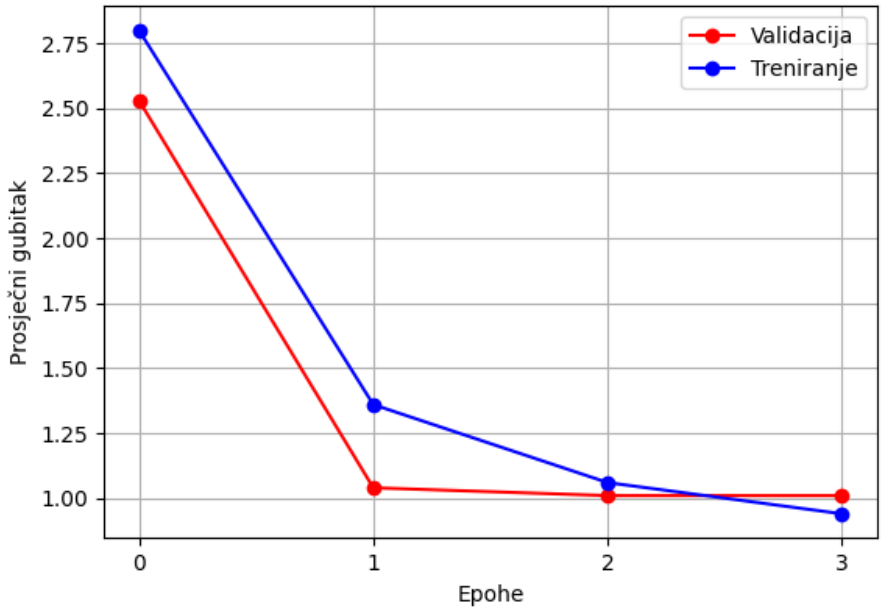
Slika 4.4: Stvaranje podataka za skup podataka *CoQA*

Budući da ovaj skup podataka ima samo podskupove za treniranje i validaciju, skup za validaciju je podijeljen na pola tako da se dobije i skup za testiranje. Prilikom kreiranja podataka preskočeni su svi primjeri koji bi bili dulji od 1024 simbola nakon konkatenacije s obzirom

na ograničenje modela GPT-2.

4.4.2. Treniranje

Budući da skup za treniranje sadrži više od 7000 primjera te svaki primjer u prosjeku 15 pitanja, zbog ograničenih resursa primjeri su morali biti uzorkovani. Uzorkovanje je provedeno tako da je slučajno izabrana četvrtina primjera iz skupa za treniranje te je od svakog od njih stvoren po jedan primjer za svaki par pitanje-odgovor postupkom koji je opisan u pododjeljku 4.4.1. Primjeri dulji od 1024 simbola su preskakani. Na taj je način dobiveno oko 26700 podataka. Model je treniran sa sljedećim hiperparametrima: `{epochs=3, learning_rate=1e-5, warmup_steps=1e2, epsilon=1e-8, batch_size=4}`. Isprobano je treniranje s nasumičnim odabirom grupa i s biranjem grupa u redoslijedu u kojem se nalaze u skupu podataka. Na taj je način dobiveno da se primjeri iz istog teksta nalaze u istim grupama. Međutim, dobiveni su gotovo identični rezultati i s nasumičnim odabirom grupa i s odabirom koji odgovara redoslijedu. Zato su navedeni samo rezultati dobiveni nasumičnim odabirom grupa. Na slici 4.5 prikazano je kako se prosječni gubitak na skupu za treniranje i validaciju ponašao kroz trening.



Slika 4.5: Treniranje modela na trening podskupu skupa CoQA od 26000 primjera

Prosječni je gubitak prije treniranja iznosio oko 2.5 na skupu za validaciju, odnosno 2.75 na skupu za treniranje. Nakon prve epohe, prosječni se gubitak značajno smanjio i na skupu za validaciju i na skupu za treniranje. Nakon druge i treće epohe, prosječni se gubitak na skupu za treniranje blago smanjivao, dok je na skupu za treniranje ostao približno isti. Isto tako,

zanimljivo je da su nakon zadnje epohe prosječni gubitak na skupu za treniranje i validaciju gotovo pa jednaki.

4.4.3. Rezultati

U Tablici 4.2 prikazani su rezultati na skupu za testiranje. Rezultati su znatno bolji nakon ugađanja modela. Prenaučenost nije problem jer pad gubitka na skupu za validaciju prati pad gubitka na skupu za treniranje. Rezultat bi sigurno bio i bolji da je treniranje provedeno na cijelom podskupu za treniranje.

Tablica 4.2: Rezultati na skupu za testiranje skupa podataka *CoQA*

Metrika	*Zero-shot*	Ugađani model
Prosječni Gubitak	2.76	1.02
Zbunjenost	15.83	2.78

5. Zaključak

Predtreniranje i manje verzije generativnog modela kao što je GPT-2 je vrlo skupo i dugotrajno te bismo predtrenirani model htjeli moći primijeniti na što više različitih tipova problema bez ikakvog dodatnog treniranja. Iz tog razloga bitno je proučiti koliko dobro takav model rješava različite zadatke bez ikakvog treniranja (*zero-shot* scenarij) te koliko se performance mogu poboljšati treniranjem koje traje nekoliko sati i može se izvesti s ograničenim resursima.

Cilj ovog rada bio je usporediti performance modela GPT-2 na složenijim zadacima jezičnog modeliranja prije i nakon ugađanja. Model je evaluiran prije i nakon ugađanja na tri različita skupa podataka. Svaki od skupova ispituje performance modela na različitim zadatcima. Na skupu podataka *CBT* nije došlo do značajnijeg poboljšanja performance, dok su na skupovima *LAMBADA* i *CoQA* dobiveni dosta bolji rezultat nakon ugađanja. Svi eksperimenti su izvršeni u oblaku tako da ih bilo tko može ponoviti, bez potrebe za većim računalnim resursima. Također, pokazalo se da model na mnogim problemima i bez ugađanja daje vrlo dobre rezultate. Iz toga najbolje vidimo koliko su predtrenirani transformatori moćni.

Rezultati bi sigurno bili bolji da je treniranje bilo provedeno na cijelim podskupovima za treniranje, no zbog ograničenih resursa moralo se na svakom skupu uzeti samo dio primjera (ciljano je da na skupu za treniranje uvijek bude oko 30000 primjera). Bilo bi zanimljivo trenirati model na skupu *CoQA* na način koji je predložen u radu [5]. U tom su radu dijelovima teksta *kontekst*, *pitanje* i *odgovor* dodavani prefiksi *"Context: "*, *"Question: "* i *"Gold answer: "* te je model da na taj način prepoznaje dijelove teksta. Bilo bi zanimljivo vidjeti koliko bi se tada performanca modela poboljšala. Također, na tom istom skupu bilo bi zanimljivo probati naći vezu između skrivenih stanja i dijela teksta u kojem se odgovor nalazi. U tom eksperimentu probali bismo istražiti može li se vidjeti u skrivenim stanjima da je prilikom generiranja odgovora na pitanje povećana pažnja na dio teksta u kojem se nalazi odgovor.

LITERATURA

[1] Dzmitry Bahdanau, Kyunghyun Cho, i Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] Yihan Cao, Siyu Li, Yixin Liu, Zhiling Yan, Yutong Dai, Philip S. Yu, i Lichao Sun. A comprehensive survey of AI-Generated Content (AIGC): A History of n Generative AI from GAN to ChatGPT, 2023.

[3] Ian J. Goodfellow, Yoshua Bengio, i Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. http://www.deeplearningbook.org.

[4] Katikapalli Subramanyam Kalyan, Ajit Rajasekharan, i Sivanesan Sangeetha. AMMUS: A Survey of Transformer-based Pretrained Models in Natural Language Processing, 2021.

[5] Daniel Khashabi, Sewon Min, Tushar Khot, Ashish Sabharwal, Oyvind Tafjord, Peter Clark, i Hannaneh Hajishirzi. UnifiedQA: Crossing Format Boundaries With a Single QA System, 2020.

[6] Ilya Loshchilov i Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[7] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, i Illia Polosukhin. Attention Is All You Need, 2017.

Primjena generativnog jezičnog modela na zadatke odgovaranja na pitanja i jezičnog modeliranja

Sažetak

Rad proučava mogućnosti generativnog jezičnog modela GPT-2. Detaljno je objašnjena ideja iza generativnih jezičnih modela i principi na kojima se temelje. Objašnjena je arhitektura modela i mehanizmi koje konkretan model koristi. Performance modela mjerene su na tri različita skupa podataka bez ugađanja i s ugađanjem na podskupovima za treniranje. Skupovi podataka testiraju sposobnost modela za rješavanje jezičnih zadataka odgovaranja na pitanja i jezičnog modeliranja. Za svaki skup je objašnjena njegova struktura, naveden broj primjera na svakom podskupu te objašnjeno što je njime testirano. Da bi model uspješno savladao te zadatke, mora moći pratiti dulji kontekst i dobro razumjeti sadržaj teksta. Opisan je postupak treniranja i nevedeni su hiperparametri korišteni za treniranje na svakom skupu. Provedena je detaljna analiza rezultata prije i nakon ugađanja modela. Komentirani su nedostatci i predloženi koraci za daljnji razvoj.

Ključne riječi: umjetna inteligencija, strojno učenje, duboko učenje, obrada prirodnog jezika, generativni modeli, kauzalno jezično modeliranje, transformator, GPT-2

Application of a generative language model to question answering and language modeling tasks

Abstract

This work studies the abilities of the generative language model GPT-2. The general idea behind generative language models and the principles on which they are based are explained in detail. The architecture of GPT-2 and the mechanisms used by it are also explained. The performance of the model is measured on three different datasets in the zero-shot setup and with fine-tuning on the training subsets. The datasets test the model's ability to handle more complex language tasks. The structure of each dataset is explained in detail. For each dataset, the size of each subset is listed, and it is explained which tasks they are testing. To succeed on these tasks, the model must be able to follow the longer context and understand it well. The training procedure is described, and hyperparameters used to train each dataset are listed. A detailed analysis of the results before and after fine-tuning was carried out. The deficiencies of training procedures are commented. The steps for further development are also suggested.

Keywords: artificial inteligence, machine learning, deep learning, natural language processing, generative models, causal language modeling, transformer, GPT-2