

UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Martin Božič

**Prilagoditev velikih jezikovnih modelov
za popravljanje slovničnih napak v
slovenščini**

MAGISTRSKO DELO
MAGISTRSKI ŠTUDIJSKI PROGRAM DRUGE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: prof. dr. Marko Robnik Šikonja

Ljubljana, 2023

To delo je ponujeno pod licenco *Creative Commons Priznanje avtorstva-Deljenje pod enakimi pogoji 2.5 Slovenija* (ali novejšo različico). To pomeni, da se tako besedilo, slike, grafi in druge sestavine dela kot tudi rezultati diplomskega dela lahko prosto distribuirajo, reproducirajo, uporabljajo, priobčujejo javnosti in predelujejo, pod pogojem, da se jasno in vidno navede avtorja in naslov tega dela in da se v primeru spremembe, preoblikovanja ali uporabe tega dela v svojem delu, lahko distribuira predelava le pod licenco, ki je enaka tej. Podrobnosti licence so dostopne na spletni strani <http://creativecommons.si/creativecommons.si> ali na Inštitutu za intelektualno lastnino, Streliška 1, 1000 Ljubljana.



Izvorna koda diplomskega dela, njeni rezultati in v ta namen razvita programska oprema je ponujena pod licenco GNU General Public License, različica 3 (ali novejša). To pomeni, da se lahko prosto distribuira in/ali predeluje pod njenimi pogoji. Podrobnosti licence so dostopne na spletni strani <http://www.gnu.org/licenses/>.

ZAHVALA

Zahvaljujem se prof. dr. Marku Robniku Šikonji za izvrstno mentorstvo. Vedno si je vzел čas in mi kvalitetno svetoval pri pisanju naloge. Usmerjal me je tudi pri učenju, izpopolnjevanju in uporabi različnih jezikovnih modelov ter mi bil v podporo tekom izdelave celotne magistrske naloge.

Martin Božič, 2023

Moji družini, prijateljem in puncu Manci,
ki mi vedno stojijo ob strani.

Kazalo

Povzetek

Abstract

1	Uvod	1
2	Sorodna dela	5
3	Metode	7
3.1	Nevronske mreže perceptron	7
3.2	Rekurenčne nevronske mreže	9
3.3	Težava dolgoročnih odvisnosti	10
3.4	Enota z dolgim kratkoročnim spominom	11
3.5	Težave nevronskih mrež tipa LSTM	12
3.6	Aktivacijske funkcije	12
3.7	Preslikovalni modeli	14
3.8	Pozornost	15
3.9	Transformerji	18
3.10	Samo-pozornost	23
3.11	Večglava pozornost	26
3.12	Učenje s prenosom znanja	29
3.13	Model T5	30
3.14	Model BERT	34

4	Podatkovna množica	41
4.1	Korpus Šolar 3.0	43
4.2	Sloleks 2.0	43
4.3	Slovar besed	44
5	Arhitektura rešitve	45
5.1	Model T5 za označevanje napačno zapisanih besed	45
5.2	Model SloBERTa za črkovanje besed	53
5.3	Znakovni model T5 za črkovanje besed	58
5.4	Model T5 za črkovanje besed	60
5.5	Model T5 za popravljanje vrstnega reda besed	67
5.6	Model T5 namenjen pregibanju besed	69
5.7	Izdelava spletnega orodja	70
6	Evalvacija rešitve	73
6.1	Evalvacija modela T5 za označevanje napačno zapisanih besed	74
6.2	Evalvacija modela SloBERTa za črkovanje besed	76
6.3	Evalvacija znakovnega modela T5 za črkovanje besed	77
6.4	Evalvacija modela T5 za črkovanje besed	78
6.5	Evalvacija modela T5 za popravljanje vrstnega reda besed . .	80
6.6	Evalvacija modela T5 za pregibanje besed	82
6.7	Primerjava rezultatov naših modelov	83
6.8	Primerjava z drugimi jeziki	84
7	Zaključek	87

Seznam uporabljenih kratic

kratica	angleško	slovensko
T5	Text-to-Text Transfer Transformer	transformer za preslikavo iz besedila v besedilo
BERT	Bidirectional Encoder Representations from Transformers	predstavitev z dvosmernim kodirnikom arhitekture transformer
RoBERTa	Robustly optimized BERT pretraining approach	robustno optimizirano učenje modela BERT
SloBERTa	Slovene RoBERTa	model RoBERTa naučen za slovenski jezik
LSTM	Long short-term memory neural networks	nevronska mreža z dolgim kratkoročnim spominom
ReLU	Rectified Linear Unit	Usklajena linearna enota
GELU	Gaussian Error Linear Unit	Gaussova napaka linearna enota
Seq2Seq	Sequence to Sequence	zaporedje v zaporedje
BPE	Byte-Pair Encoding	bitno parno kodiranje
TP	true positive	dejansko pozitivni
FP	false positive	napačno pozitivni
FN	false negative	napačno negativni
TN	true negative	dejansko negativni

KAZALO

Povzetek

Naslov: Prilagoditev velikih jezikovnih modelov za popravljanje slovničnih napak v slovenščini

Strojno popravljanje slovničnih napak v slovenskem jeziku je še ne rešen problem. Rešitev bi olajšala pisno komunikacijo. Problem v okviru magistrske naloge razdelimo na podprobleme: popravljanje zapisa besed, zaznavanje napačno zapisanih besed, popravljanje pregibanja besed in popravljanje vrstnega reda besed. Najboljše rezultate dosežemo z izpopolnjevanjem slovenskega SloT5 modela. Najboljše modele uporabimo pri izdelavi spletne aplikacije. Ugotovimo, da je pri reševanju problema popravljanja slovničnih napak najbolj pomembna izbira ustreznega osnovnega jezikovnega modela in izgradnja kvalitetne učne množice. Pri izgradnji učne množice skušamo zajeti čim več kvalitetnih in realnih slovničnih napak, ne da bi pri tem spremenili ali pokvarili izvorni pomen besedila.

Ključne besede

model T5, transformerji, model BERT, model SloBERTa, slovnični popravki, nevronske mreže, strojno učenje

Abstract

Title: Adaptation of large language models for grammar correction in Slovene

Machine correction of grammatical errors in the Slovenian language is still an unsolved problem. Its solution would improve written communication. We divide the problem into subproblems: correcting word spelling, detecting misspelled words, correcting word inflection and correcting word order. The best results are achieved by finetuning the Slovenian SloT5 model. We use the best models in a web application. We conclude that in correcting grammatical errors, the most important consideration is the choice of a large language model and construction of a learning set. When building the learning set, we try to capture as many realistic grammatical errors as possible, without changing the meaning of the text.

Keywords

model T5, transformers, model BERT, model SloBERTa, grammatical corrections, neural networks, machine learning

Poglavje 1

Uvod

Razvoj arhitekture nevronske mreže transformer [50] je prispeval k razvoju velikega števila izboljšav na področju obdelave naravnega jezika. Pojavili so se boljši modeli za generiranje naravnega jezika [51], za oblikoskladenjsko označevanje besed [27] in odpravljanje posameznih jezikovnih napak [22]. Ker je na spletu na voljo največ besedilnih podatkovnih množic v angleškem jeziku, je ta jezik tehnološko najbolj razvit. Za slovenski jezik je bilo razvitih več modelov za procesiranje posameznih nalog, vendar splošnega pristopa za odpravljanje jezikovnih napak še nimamo.

Arhitektura transformer je v osnovi zgrajena iz kodirnika in dekodirnika. Naloga kodirnika je zapis vhodnega besedila v visokodimenzionalni vektor, ki predstavi pomen vhodnega besedila. Namen dekodirnika je pretvorba tega vektorja nazaj v razumljivo obliko z določenimi lastnostmi (npr. v drugem jeziku). To lastnost transformerjev bomo izkoristili za pretvorbo napačnega besedila v zapis v pravilni obliki.

T5 (Text to Text Transformer) [42] je model transformer tipa kodirnik-dekodirnik, ki ga bomo v okviru magistrskega dela izpopolnili za popravljanje slovničnih napak v slovenščini. Na jezikovnih problemih črkovanja, skladnje in vrstnega reda besed bomo izpopolnili slovenski model SloT5 [48]. Na problemu črkovanja bomo primerjali, kako se natančnost izpopolnjenega modela spremeni, če posamezno besedilo obravnavamo s celimi besedami oziroma

posameznimi znaki.

Skupno bomo prilagodili in izpopolnili pet različnih modelov T5. Pri vsakem modelu bomo uporabili prilagojeno učno množico, ki jo bomo zgradili iz korpusa Gigafida [28]. Model T5 bomo prilagodili za označevanje napačno zapisanih besed, dva modela T5 pa za črkovanje besed. Ubrali bomo dva pristopa. V prvem bomo napake skušali odpraviti z obravnavanjem besed v celoti, v drugem pristopu pa bomo besede razdelili na posamezne znake in pri zapisu besed popravljali le znake.

Model T5 bomo izpopolnili tudi za popravljanje vrstnega reda besed in za pregibanje besed. Pri izgradnji učne množice za pregibanje besed, bomo s pomočjo korpusa Gigafida ustvarili poseben slovar besed, ki bo omogočal, da bomo posamezne besede v učni in testni množici zamenjali z enakimi besedami, ki bodo drugače pregibane.

Pri učenju modela za popravljanje vrstnega reda besed se bomo osredotočili na pravilen vrstni red besed znotraj posameznih stavkov, ki jih bomo med seboj ločili glede na veznike in vejico.

V spletni aplikaciji bomo združili izpopolnjene modele T5 za označevanje napačno zapisanih besed, za črkovanje besed, za pregibanje besed in za popravljanje vrstnega reda besed.

Poleg izpopolnjevanja vnaprej naučenega modela SloT5, bomo na problemu črkovanja besed izpopolnili še vnaprej naučen slovenski model SloBERTa [49], ki ga bomo testirali s pomočjo korpusa Šolar 3.0 [4].

Delo je razdeljeno na sedem poglavij: V poglavju 2 bomo opisali sorodna dela. V poglavju 3 pa uporabljene tehnologije. Opisali bomo osnovno zgradbo nevronske mreže, predstavili njihove težave in izboljšave ter opisali različne aktivacijske funkcije. Predstavili bomo tudi zgradbo arhitekture *transformer*, koncept pozornosti in arhitekturo modela *transformer* tipa T5. Poglavje bomo zaključili z opisom zgradbe modela BERT.

V poglavju 4 bomo opisali učno množico za učenje in testiranje modelov. V poglavju 5 bomo opisali vse izpopolnjene modele. Pri vsakem modelu bomo opisali pripravo učne množice, proces učenja in testiranja modela. Na

koncu poglavja bomo opisali izdelavo spletnega orodja.

Rezultate testiranja vseh modelov bomo opisali v poglavju 6, kjer bomo rezultate komentirali in navedli možne izboljšave. V poglavju 7 bomo delo povzeli in zapisali najpomembnejše ugotovitve. Končali bomo z opisom omejitve dela in idejami za izboljšave.

Poglavje 2

Sorodna dela

Arhitektura transformer je v osnovi zgrajena iz kodirnika in dekodirnika, pri čemer lahko tako kodirnik kot dekodirnik ločimo in s pomočjo ločenega sklopa sestavimo jezikovni model. Model BERT je jezikovni model, zgrajen le iz kodirnika transformerja, in se je izkazal za uporabnega pri reševanju mnogih jezikovnih problemov. Deng in sod. [30] so prvi prenesli nevronske prevajalni model, osnovan na modelu BERT, na področje zaznave in popravljanja gramatičnih napak v kitajščini. Predlagan sistem je sestavljen iz dveh modelov, in sicer iz modela za pozicijsko označevanje besedila in iz modela za označevanje popravkov v besedilu. Sistem se je najbolje izkazal pri nalogah korekcije napak.

Pri prilagajanju jezikovnih modelov, namenjenih prepoznavi in popravljanju napak, dostikrat naletimo na problem premajhe količine kakovostnih učnih podatkov. Roman in sod. [18] so predlagali preprosto metodo za generiranje sinteničnih napak, s katero je mogoče povečati podatkovno množico, uporabljeno pri učenju modela. Metoda izkorišča konfuzne množice besed, pri čemer v isto konfuzno množico štejemo besede, ki jih pomensko med seboj hitro pomešamo. Takšne besede metoda pridobi iz gramatičnega popravljalnika in jih vključuje v učno množico.

V slovenskem jeziku imamo trenutno naučenih več jezikovnih modelov, baziranih na arhitekturi BERT, ki jih je mogoče uporabiti za popravljanje

jezikovnih napak. Mashiro in sod. [23] predlagajo način za uporabo predhodno naučenega modela BERT pri izdelavi modela, namenjenega popravljanju jezikovnih napak arhitekture kodirnik-dekodirnik. S predlagano metodo najprej izpopolnimo jezikovni model, naučen na podlagi maskiranja besed, nato pa izhode modela uporabimo kot dodatne parametre pri gradnji končnega modela za popravljanje slovničnih napak.

Z razvojem sistema za obvladovanje slovničnih napak se je ukvarjal Mokotar [36], ki je problem razdelil na tri podprobleme: na zaznavanje, prepoznavanje in popravljanje napak. Problem zaznavanja in prepoznavanja slovničnih napak je reševal z uporabo slovenskega modela SloBERTa, problem popravljanja slovničnih napak pa z uporabo slovenskega modela SloT5. Vsi predlagani sistemi so razdelili vhodno besedilo na povedi, pri tem pa popravljali slovnično pravilnost ločeno za vsako poved. Za učenje in evalviranje modelov je uporabil slovenska korpusa Šolar in Lektor.

Z uporabo modela tipa transformer pri popravljanju gramatičnih napak v korejskem jeziku so se ukvarjali Myunghoon in sod. [29]. Uporabili so model tipa transformer, pri tem pa so pri izgradnji učne in testne množice uporabili mehanizme kopiranja in generiranja gramatičnega šuma v besedilih. Ena izmed napak, ki so jih skušali odpraviti, je pisanje besed skupaj ali narazen, za kar so uporabili korejsko knjižnico ChatSpace. Z uporabo modela tipa transformer pri popravljanju gramatičnih napak so močno izboljšali prej obstoječe metode, ki so v osnovi delovale glede na gramatična pravila.

S popravljanjem slovničnih napak z uporabo modela BERT se je ukvarjal Petrič [41]. Osredotočil se je na popravljanje sklona roditeljski-tožilnik in števila množina-dvojina. Za učenje in testiranje modelov je uporabil korpus Lektor. Ugotovil je, da uporaba modela SloBERTaMCD v primerjavi z modelom SloBERTa pri problemu besednih zamenjav ni privedla do izboljšanja rezultatov.

Poglavje 3

Metode

Procesiranje naravnega besedila je doživelo razcvet s prihodom transformerjev [50], katerih glavni namen je bilo strojno prevajanje.

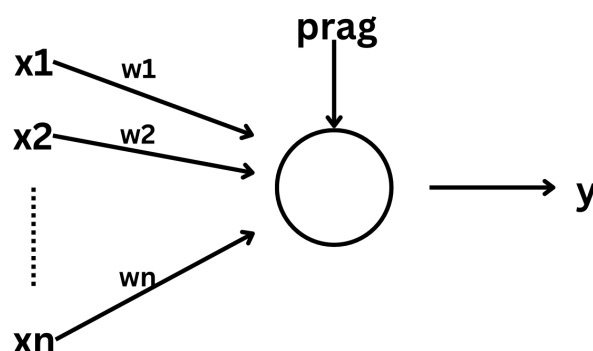
Pred uporabo transformerjev, so bili pri večini nalog najboljši rezultati doseženi z uporabo globokih rekurenčnih nevronske mreže. Trenutno na področju procesiranja naravnega jezika transformerji izpodrivajo ostale globoke nevronske mreže.

V tem poglavju bomo na kratko opisali nevronske mreže, predstavili njihove slabosti in izboljšave. Opisali bomo princip pozornosti, ki je osnovni gradnik modelov tipa transformer. V nadaljevanju bomo opisali zgradbo modela tipa transformer in modela T5 in BERT.

3.1 Nevronske mreže perceptron

Klasične nevronske mreže so zgrajene iz perceptronov, ki na osnovi večih uteženih realnih vhodov generirajo binarni izhod. Primer takega perceptrona vidimo na sliki 3.1.

Izhod posameznega nevrona izračunamo s skalarnim produktom vhodov in pripadajočih uteži, ki določajo pomembnosti in vpliv posameznega vhoda pri določanju vrednosti izhoda nevrona. Izhod posameznega nevrona izračunamo z enačbo 3.1.



Slika 3.1: Primer zgradbe preprostega nevrona z n vhodi in enim binarnim izhodom.

$$y = \begin{cases} 0 & \sum_{j=1}^n w_j x_j \leq \text{prag} \\ 1 & \sum_{j=1}^n w_j x_j > \text{prag} \end{cases} \quad (3.1)$$

Pri tem je

y izračunan binarni izhod perceptrona,

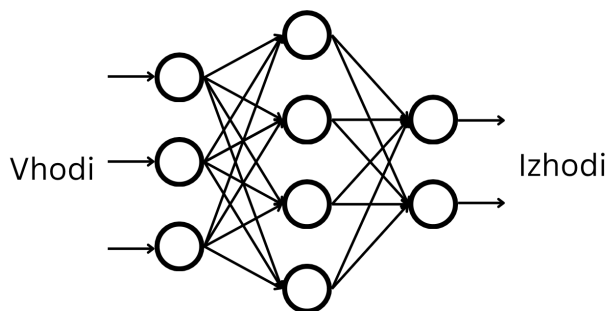
$w_{1...n}$ vrednost posamezne uteži nevrona,

prag vrednost, pri kateri perceptron spremeni vrednost izhoda.

V realni implementaciji vrednost praga (angl. threshold) nadomestimo z vrednostjo pristranskosti (angl. bias), ki jo prištejemo k vrednosti trenutnega izhoda nevrona, pridobljeno vrednost pa prepeljemo skozi aktivacijsko funkcijo (razdelek 3.6).

Pri izračunu izhodne vrednosti nevrona, vsak vhod pomnožimo z vrednostjo priležne uteži vhoda, vse pridobljene vrednosti seštejemo in prištejemo pristranskost (angl. bias). Celoten rezultat kasneje peljemo skozi aktivacijsko funkcijo. Izhod aktivacijske funkcije predstavlja izhodno vrednost nevrona.

En sam nevron nam daje možnost preprostega odločanja. Več nevronov povezanih v mrežo predstavlja nevronska mreža, kjer vsak nevron pripomore k odločanju izhoda. Primer nevronske mreže je prikazan na sliki 3.2.



Slika 3.2: Slika prikazuje primer nevronske mreže z izhodi obrnjenimi naprej. Prikazana nevronska mreža je zgrajena iz ene vhodne, ene skrite in ene izhodne plasti navronov.

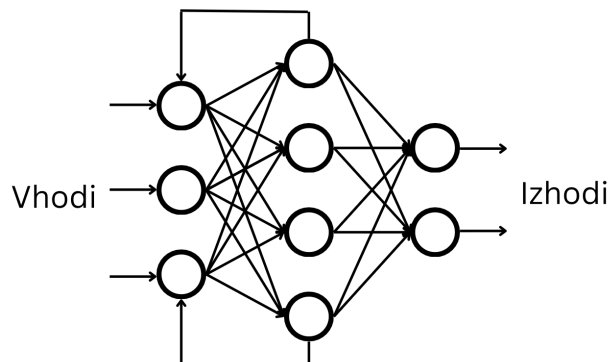
V kolikor ima nevronska mreža eno ali več skritih plasti, govorimo o globokih nevronskih mrežah.

3.2 Rekurenčne nevronske mreže

Ena izmed glavnih nalog rekurenčnih nevronskih mrež je zaporedno prevajanje, katerega temeljna zahteva je možnost kratkoročnega pomnjenja.

Model nevronskih mrež opisan v prejšnjem razdelku nam kratkoročnega pomnjenja ne omogoča, saj nobena celica nima določenega stanja, ki bi na podlagi trenutnega izračuna izhoda vplivalo na naslednji izračunan izhod.

Računanje spremenimo, če nekatere izhode iz posameznih celic peljemo na vhode nevronov, ki se v nevronske mreži nahajajo pred trenutnim nevronom in so na izhod trenutnega nevrona že vplivale. Nevronske mreže z izhodi nevronov obrnjenimi nazaj imenujemo rekurenčne nevronske mreže. Preprosta skica rekurenčne nevronske mreže je prikazana na sliki 3.3.



Slika 3.3: Slika prikazuje primer zgradbe rekurenčne nevronske mreže z izhodi nekaterih celic obrnjenimi nazaj.

Če prikaz rekurenčne nevronske mreže poenostavimo, jo lahko predstavimo kot celico z vhodom, izhodom in krožno zanko, ki predstavlja povezavo nazaj. Če namesto uporabe krožne zanke celico rekurenčne nevronske mreže razvijemo, dobimo preprost model verižne sekvence celic rekurenčne nevronske mreže.

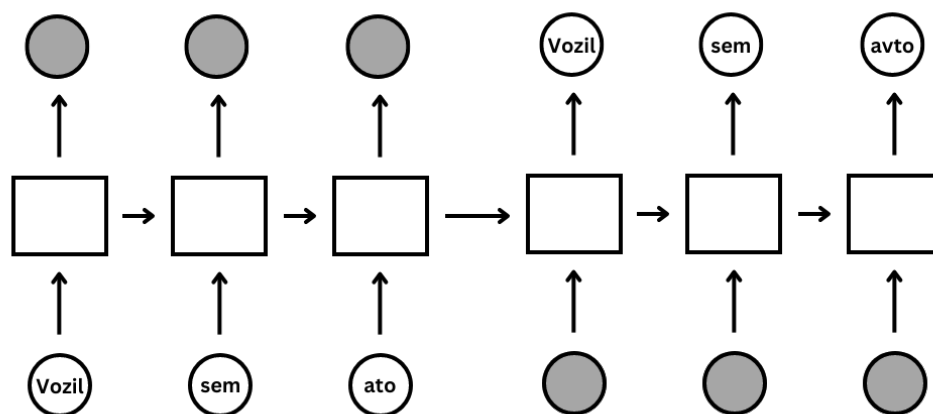
Verižna struktura celic rekurenčne nevronske mreže nam omogoča, da lahko celica pri izračunu trenutnega izhoda uporabi pridobljeno znanje v prejšnjih iteracijah računanja.

Če nas pri izračunu daljšega zaporednja vhodov zanima le končni izhod, pri izračunu stanja posamezne celice pa vedno uporabimo znanje pridobljeno v prejšnjih celicah, govorimo o sekvenčnih modelih. Primer preprostega sekvenčnega modela je prikazan na sliki 3.4.

3.3 Težava dolgoročnih odvisnosti

Rekurenčne nevronske mreže nam omogočajo, da na podlagi preteklih vhodov predvidevamo trenutni izhod. Slika 3.4 prikazuje primer, kjer želimo v dveh stavkih popraviti napako pri pisanju besede avto. Predpostavljamo, da

rekurenčna mreža na vhod prejema besedo za besedo in na podlagi skritega stanja zadnje celice v verigi vhodov generira izhodne besede. Kot vidimo na sliki, besedo “ato” model popravi v besedo “avto”. Če namesto začetne besede “Vozil” uporabimo besedo “Raztrgal”, bi bil na primer stavek “Raztrgal sem vato” bolj pravilen od stavka “Raztrgal sem avto”, zato bi model besedo “ato” popravil v besedo “vato”.



Slika 3.4: Slika prikazuje uporabo rekurenčnega sekvenčnega modela pri popravljanju besede “ato”.

Težava pri rekurenčnih nevronske mrežah nastane pri reševanju dolgoročnih odvisnosti. Problem dolgoročnih odvisnosti nastane, ker se na vsakem koraku celotno znanje celice prenese v naslednji korak in daljša kot je veriga, več informacije se pri tem izgubi.

3.4 Enota z dolgim kratkoročnim spominom

Glavna težava rekurenčnih nevronske mrež je nezmožnost odločanja, katero informacijo bodo celice v sklopu odločanja ohranile in katero zavrgle. Vedno se vse informacije prenesejo naprej in del informacij, ki je pri trenutnem odločanju najbolj pomemben, lahko izgubi na pomembnosti.

Omenjeno težavo rešujejo celice LSTM (angl. Long Short Term Memory) [20], ki imajo možnost, da del informacije shranijo ali zavržejo. Vsa informacija potuje skozi mehanizem stanj v celici in s pomočjo operacij skalarne produkta celica izbere, katere dele informacije bo ohranila in katere zavrgla.

Vsaki LSTM celici podamo na vhod, ki je npr. beseda ali znak, stanje prejšnje celice in izhod prejšnje celice. Celica LSTM te vhode prebere, jih posodobi in na izhod vrne novo stanje celice ter izhod. Posodabljanje ali ohranjanje stanja celice iz enega koraka v drugi nam omogoča, da je določena informacija, ki je lahko pomembna pri odločanju celice, prisotna pri procesu izračuna izhoda v vseh celicah.

3.5 Težave nevronske mreže tipa LSTM

Tako kot pri rekurenčnih nevronske mreže, težave pri LSTM nevronske mreže nastanejo, ko so odvisnosti med vhodi daleč narazen. Glavni razlog za to je oteženo ohranjanje konteksta, ker vsaka celica trenutnemu kontekstu doda še informacijo, ki se ji zdi pomembna. Zaradi tega verjetnost ohranjanja pomembne informacije skozi več korakov eksponentno pada.

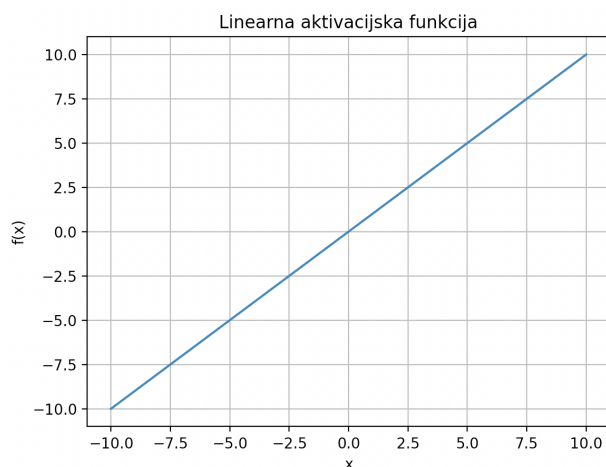
V nevronske mreže tipa LSTM obravnavamo vsak vhod v zaporednem vrstnem redu, saj informacija iz trenutnega vhoda neposredno vpliva na izračun izhoda naslednje celice. Ta lastnost otežuje paralelizacijo učenja.

Dodatna težava nevronske mreže tipa LSTM je nezmožnost razlikovanja med kratkoročnimi in dolgoročnimi odvisnostmi med vhodi.

3.6 Aktivacijske funkcije

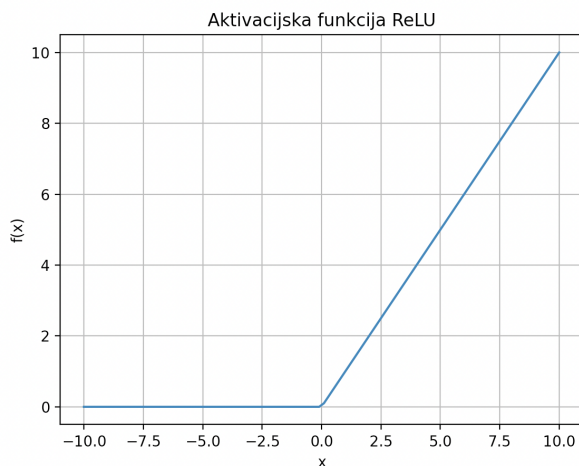
Aktivacijske funkcije predstavljajo pomemben del vsake nevronske mreže. Njihova naloga je preslikati izhod posameznega nevrona.

Najpreprostejša aktivacijska funkcija je linearna funkcija, ki na izhod preslika nespremenjeno vrednost nevrona. Njen graf prikazuje slika 3.5.



Slika 3.5: Slika prikazuje graf linearne aktivacijske funkcije.

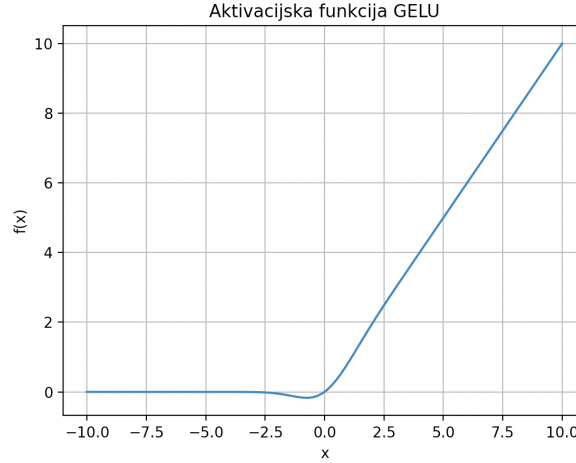
Težava linearne funkcije je, da upošteva tudi negativne vrednosti, ki lahko učenje nevronske mreže otežijo. Če pri linearni funkciji vsa negativna števila preslikamo v 0 dobimo aktivacijsko funkcijo ReLU, ki je v praksi pogosto uporabljena. Graf aktivacijske funkcije ReLU je prikazan na sliki 3.6.



Slika 3.6: Slika prikazuje graf aktivacijske funkcije ReLU.

Težava aktivacijske funkcije ReLU je, da se izhod spremeni glede na pred-

značenost vhoda. Če aktivacijsko funkcijo ReLU zgladimo in jo naredimo na vseh mestih odvedljivo, pridobimo aktivacijsko funkcijo GELU. Graf aktivacijske funkcije GELU je prikazan na sliki 3.7.



Slika 3.7: Slika prikazuje graf aktivacijske funkcije GELU.

Aktivacijska funkcija Softmax preslika vhodni vektor v vektor verjetnosti. Aktivacijsko funkcijo Softmax predstavlja enačba 3.2.

$$S(x)_i = \frac{e^{\beta(x_i)}}{\sum_{j=1}^n e^{\beta(x_j)}} \quad (3.2)$$

Pri tem je

- x vhodni vektor v aktivacijsko funkcijo Softmax,
- n število elementov v vhodnem vektorju x ,
- β parameter, ki je nastavljen in ga definiramo v času učenja nevronske mreže.

3.7 Preslikovalni modeli

Glavna lastnost preslikovalnih modelov (angl. Seq2Seq models) je preslikovanje nekega zaporednja elementov v novo zaporedje. Preslikovalni modeli

so se izkazali za uspešne pri prevajanju besedil in ostalih jezikovnih nalogah, ki temeljijo na prepisu vhodnega zaporedja besed v novo zaporedje.

V osnovi so preslikovalni modeli zgrajeni iz kodirnika in dekodirnika. Naloga kodirnika je preslikovanje vhodnega vektorja v nek več dimenzionalni skriti vektor. Dekodirnik nato ta več dimenzionalen vektor preslika v izhodni vektor.

Pri strojnem popravljanju pravopisnih napak si lahko delovanje kodirnika in dekodirnika predstavljamo tako, da na vhod kodirniku podamo poljubno besedilo, ki lahko vsebuje napake. To vhodno besedilo nato kodirnik preslika v večdimenzionalni vektor, dekodirnik pa pridobljeni vektor prepiše nazaj v besedilo brez napak.

3.8 Pozornost

Glavna naloga koncepta pozornosti [39] je ohranjanje in določanje pozornosti na določene dele vhodov, za katere tekom učenja določimo, da so pomembni pri izračunu izhodov.

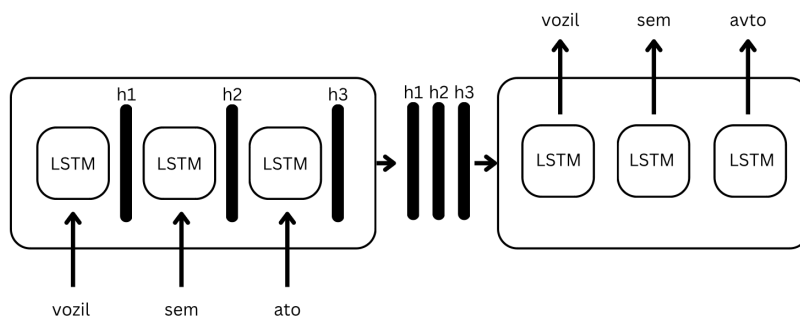
Pri strojnem popravljanju pravopisnih napak, opisanem v razdelku 3.7, močno izboljšamo predstavitev vhodnega besedila, če kodirniku v vhodnem besedilu označimo, na katerem mestu se najverjetneje nahaja napaka in katera mesta v besedilu so za odpravljanje napak najbolj pomembna. Isto informacijo lahko izkoristi dekodirnik, da ve, katere dele vhodnega besedila lahko samo prepiše in katere naj pri oblikovanju izhodnega besedila spremeni.

Na sliki 3.8 je prikazan preslikovalni model z rekurenčnimi celicami tipa LSTM, kjer v skrito stanje mreže zakodiramo vsako besedo ločeno in na koncu iz vektorja vseh skritih plasti generiramo izhodni vektor, ki ga posredujemo vsaki celici dekodirnika.

Ohranjanje skritih plasti nam omogoča, da ohranimo informacijo vseh vhodov, ki jo potem lahko uporabimo pri generiranju vseh izhodov. Pri tem nam koncept pozornosti omogoča, da lahko pri vsakem generiranju izhoda

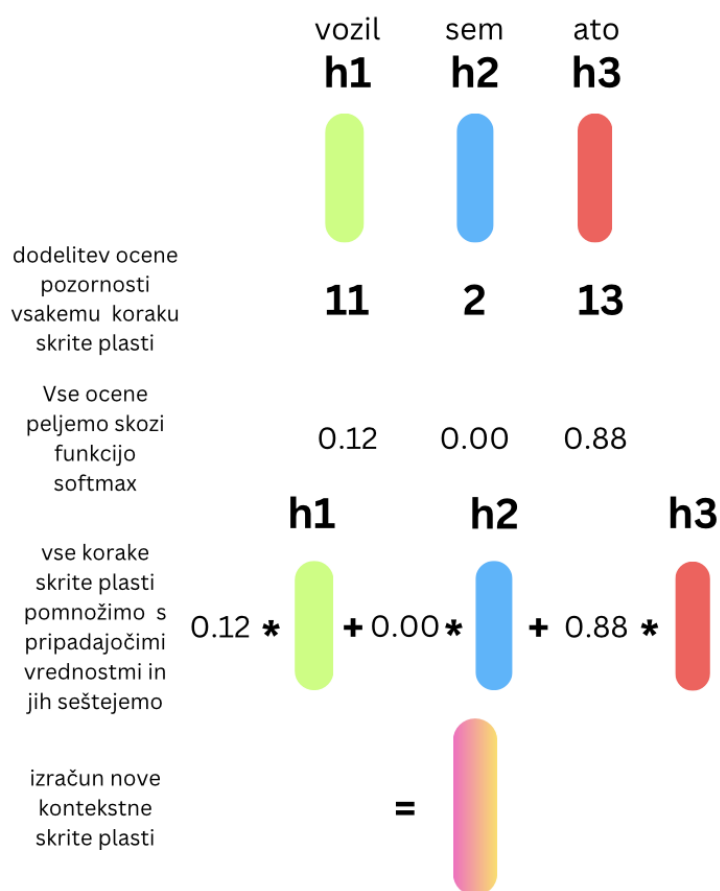
izberemo, katero skrito plast bomo upoštevali in katere ne.

Slika 3.9 prikazuje, kako s pomočjo koncepta pozornosti izračunamo skrito stanje, ki ga uporabimo pri izračunu vsakega izhoda dekodirnika. Pri vsakem izračunu izhoda ocenimo pomembnost vsake skrite plasti in vsaki oceni dodelimo vrednost. Vse pridobljene vrednosti peljemo skozi aktivacijsko funkcijo Softmax (razdelek 3.2), ki poskrbi, da je seštevek vseh vrednosti enak 1. Vsako skrito plast nato pomnožimo s priležno ocenjeno vrednostjo pozornosti in nato vse skrite plasti seštejemo. Na tak način pridobimo nov kontekstni vektor, ki ga uporabimo pri izračunu trenutnega izhoda.



Slika 3.8: Slika prikazuje primer preslikovalnega modela z mehanizmom pozornosti.

Koncept pozornosti močno zmanjša težave ohranjanja dolgoročnih odvisnosti.



Slika 3.9: Slika prikazuje primer izračuna skritega kontekstnega vektorja, ki ga uporabimo za popravek besede “ato” v “avto”. Iz slike je razvidno, da pri izračunu izhodnega kontekstnega vektorja uporabimo tri skrite vektorje (za besede “vozil”, “sem” in “ato”). Skrite vektorje uporabimo za popravek tretje besede, ki je beseda “ato”. Na sliki vidimo, da model najvišjo vrednost pozornosti dodeli tretjemu vektorju, kjer gre za napačen zapis besede “avto”, pri ostalih dveh besedah pa večjo vrednost dodeli besedi “vozil”, ker se beseda “vozil” bolj nanaša na popravek besede “ato”.

3.9 Transformerji

Transformerji [50] slonijo na tehnologiji kodirnikov in dekodirnikov in hkrati uporabljajo tehnologijo pozornosti. Ena izmed pomembnejših lastnosti arhitekture transformerjev je, da ne uporabljajo rekurenčnih nevronske mreže, kljub temu pa dosežejo dobre rezultate pri prevajanju in drugih nalogah procesiranja naravnega jezika.

Transformer je zgrajen iz kodirnika in dekodirnika. Kodirnik in dekodirnik sta zgrajena iz več zaporednih modulov. Zgradbo transformerja prikazuje slika 3.10.

Kodirnik je zgrajen iz N (v arhitekturi, predstavljeni v [50], je $N=6$) modulov. Vsak posamezni modul je zgrajen iz komponente za izračun večglave pozornosti in polno povezane nevronske mreže. Delovanje komponente za izračun večglave pozornosti podrobneje razložimo v razdelku 3.11.

Polno povezana nevronska mreža je zgrajena iz dveh plasti nevronov. Pri tem uporablja povezave naprej, ne uporablja pa povratnih povezav, kar pomeni, da ne omogoča pomnenja. Primer izračuna izhoda omenjene nevronske mreže prikazuje enačba 3.3.

$$y = ReLU(W_1x + b_1)W_2 + b_2 \quad (3.3)$$

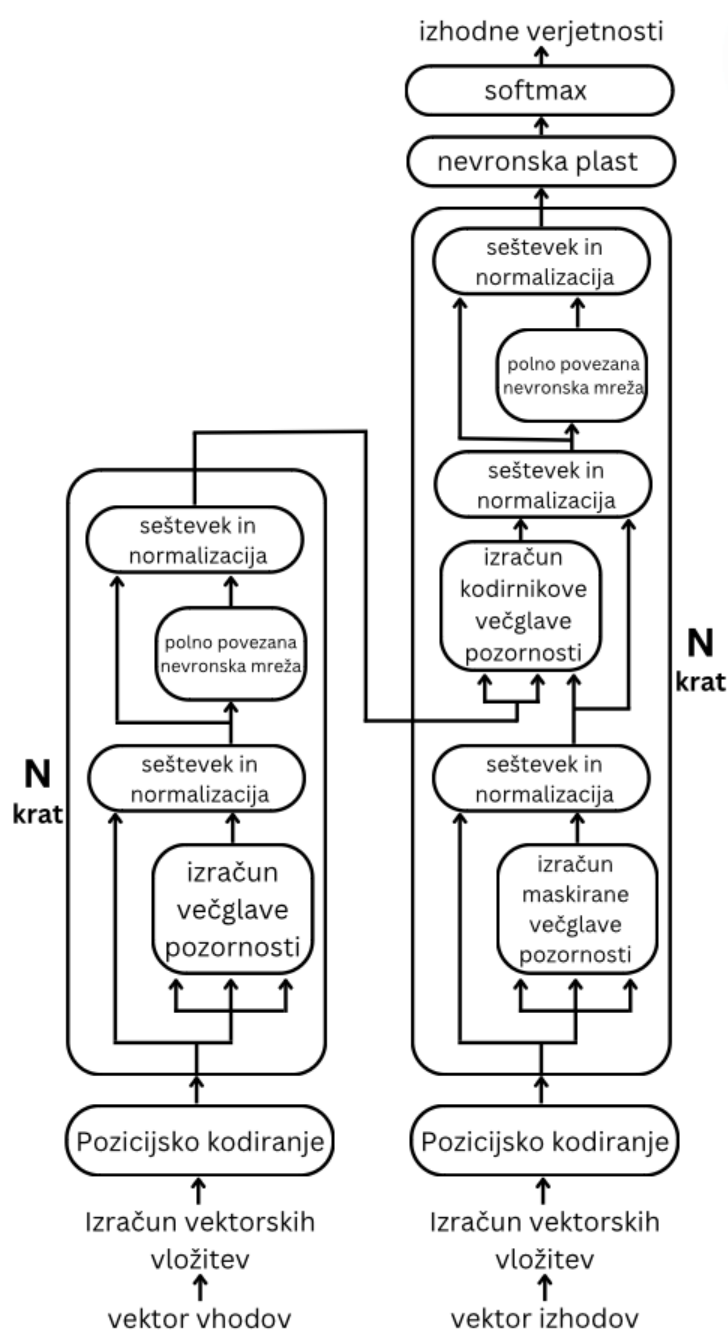
Pri tem je

y izhod nevronske plasti,

$ReLU$ aktivacijska funkcija ReLU,

b vrednost pristranskosti,

W_1 in W_2 matrika uteži.



Slika 3.10: Slika prikazuje zgradbo arhitekture transformerja. Na levi strani je primer kodirnika, na desni pa primer dekodirnika. Izhodi zadnjega kodirnika so usmerjeni v vse dekodirnike in so uporabljeni pri izračunu večglave pozornosti. Vsak transformer je zgrajen iz N kodirnikov in dekodirnikov, ki so naloženi drug na drugega.

Delovanje vseh N kodirnikov je identično, vendar pa vsak kodirnik pri transformacijah uporablja različne uteži (angl. weights) in pristranskosti (angl. bias), ki jih določi skozi proces učenja. Vsaka komponenta kodirnika ima rezidualno povezavo, ki omogoča prost pretok informacij v višje plasti kodirnika in v ostale kodirnike. Vsaki komponenti dodamo plast za normalizacijo (angl. normalization layer), kjer najprej seštejemo oba vhoda, nato pa njun seštevek normaliziramo. Prvi vhod predstavlja vhod v komponento, ki se nahaja pod normalizacijsko plastjo, drugi vhod pa izhod iz te komponente. Enačba 3.4 prikazuje primer izračuna izhoda komponente za normalizacijo. Izhod zadnje normalizacijske plasti predstavlja izhod kodirnika.

$$\begin{aligned}
 v &= x + \text{podPlast}(x) \\
 \text{normalizacijskaPlast}(v) &= \gamma \frac{v - \mu}{\sqrt{\sigma^2}} + \beta \\
 \mu &= \frac{1}{d} \sum_{k=1}^d v_k \\
 \sigma^2 &= \frac{1}{d} \sum_{k=1}^d (v_k - \mu)^2
 \end{aligned} \tag{3.4}$$

Pri tem je

- x vhod v posamezno komponento pod plastjo za normalizacijo,
- v vektor seštevka vhodov v normalizacijsko plast,
- $\text{podPlast}(x)$ izračunan izhod posamezne komponente, ki stoji pred normalizacijsko plastjo,
- d dolžina vektorja na vhodu,
- μ povprečna vrednost vektorja, ki ga želimo normalizirati,
- σ varianca vektorja, ki ga želimo normalizirati,
- γ parameter, ki se ga transformer nauči v fazi učenja,
- β parameter, ki se ga transformer nauči v vazi učenja.

Zaradi paralelizacije vhodov in s tem hitrejšega računanja, kodirnik ne hrani informacije o vrstnem redu vhodov. Da se vrstni red besed ohrani skrbi komponenta za pozicijsko kodiranje vhodov, ki vsakemu vhodu doda pozicijsko

oznako in s tem kasnejšim plastem poda informacijo o vrstnem redu vhodov. Pozicijsko kodiranje je bolj natančno opisano v razdelku 3.11.1.

Dekodirnik je v osnovi zgrajen podobno kot kodirnik. Sestavlja ga N (v osnovni različici je $N=6$) identičnih plasti, pri tem pa je vsaka plast razdeljena na 3 komponente.

Prva komponenta predstavlja maskirano večglavo pozornost, ki jo imenujemo komponenta za izračun samopozornosti dekodirnika. Za razliko od kodirnika, ki v vsakem koraku obdela vse vhode v model, se dekodirnik osredotoča le na preteklih n vhodov. Slednje mu omogoča koncept maskiranja besed. Maskiranje izvede z nastavljanjem določenih vrednosti izračunanega produkta matrik Q in K na negativno neskončnost. Kasneje, ko izračunane vrednosti peljemo skozi aktivacijsko funkcijo Softmax, to privede do tega, da funkcija Softmax maskiranim vrednostim poda nično vrednost. Matriki Q in K sta podrobneje opisani v razdelku 3.10. Enačba 3.5 prikazuje način maskiranja produkta matrik Q in K .

$$maskiraj(QK^T) = maskiraj\left(\begin{bmatrix} e_{1,1} & e_{1,2} & \cdots & e_{1,n} \\ e_{2,1} & e_{2,2} & \cdots & e_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ e_{m,1} & e_{m,2} & \cdots & e_{m,n} \end{bmatrix}\right) = \begin{bmatrix} e_{1,1} & -\infty & \cdots & -\infty \\ e_{2,1} & e_{2,2} & \cdots & -\infty \\ \vdots & \vdots & \ddots & \vdots \\ e_{m,1} & e_{m,2} & \cdots & e_{m,n} \end{bmatrix} \quad (3.5)$$

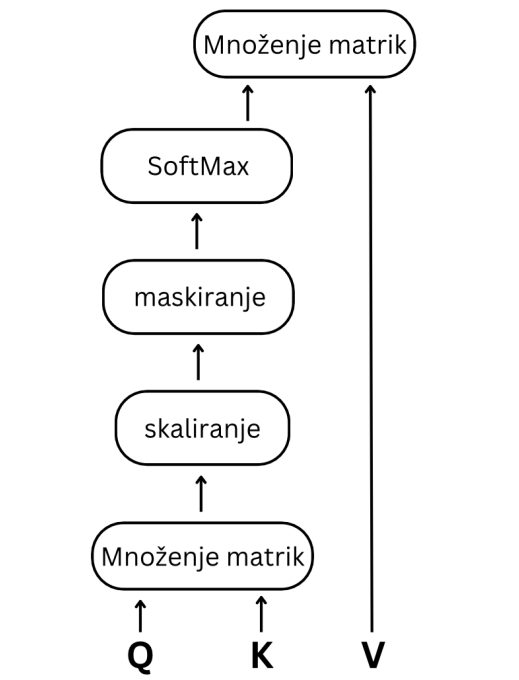
Pri tem je

Q matrika poizvedbenih (query) vektorjev,

K matrika ključnih (key) vektorjev,

$maskiraj(...)$ funkcija maskiranja vhodov.

Koncept večglave pozornosti je definiran z več posameznimi izračuni maskirane pozornosti. Postopek izračuna maskirane pozornosti prikazuje slika 3.11.



Slika 3.11: Slika prikazuje postopek izračuna maskirane pozornosti.

Druga komponenta dekodirnika je zgrajena iz večglave samopozornosti, ki je podobna večglavi samopozornosti kodirnika z razliko, da dekodirnik izračuna proizvedbeno matriko (Q) iz izhoda prejšnjega sloja dekodirnika, matriko ključev (K) in matriko vrednosti (V) pa iz izhoda zadnjega kodirnika. To omogoča dekodirniku, da pri izračunu izhoda upošteva vse vhode v kodirnik.

Tretja komponenta dekodirnika je sestavljena iz dvoslojne nevronske mreže, ki je po zgradbi enaka nevronske mreži kodirnika.

Vse komponente dekodirnika so obdane z residualnimi povezavami. Vsaki komponenti sledi normalizacijska plast, ki združi in normalizira vrednost residualne povezave in vrednost izhoda trenutne komponente.

Vsi vhodi v dekodirnik so prepeljani skozi plast pozicijskega kodiranja, opisanega v razdelku 3.8.

Izhod zadnjega dekodirnika predstavlja vektor, ki ga nato prepeljemo

skozi polno povezano nevronske mrežo. Ta vektor preslika v vektor logit vrednosti. Njegova dolžina je enaka velikosti slovarja besed, uporabljenega v fazi učenja modela. Vsak posamezni element vektorja predstavlja vrednost istoležne besede v slovarju. Vektor logit vrednosti na koncu prepeljemo skozi aktivacijsko funkcijo Softmax, ki vrednosti besed spremeni v verjetnosti. Pri tem se verjetnosti seštejejo v 1.

3.10 Samo-pozornost

Vsak vhod v kodirnik pošljemo skozi plast za izračun vektorske vložitve (angl. embedding vectors).

Iz vsakega vhodnega vektorja nato izračunamo tri vektorje (vektor poi-
zvedbe (q), vektor ključa (k) in vektor vrednosti (v)). Izračunamo jih tako,
da vložitve vhodnih vektorjev množimo z matrikami priležnih uteži (W_q , W_k
in W_v). Te matrike izračunamo v procesu učenja transformerja.

Dimenzija vektorjev q , k in v je 64, medtem ko je dimenzija posamezne
vektorske vložitve vhoda enaka 512. Enačbe 3.6 prikazujejo izračun vektorjev
 q , k in v .

$$\begin{aligned} q_1 &= W_q x_1 \\ k_1 &= W_k x_1 \\ v_1 &= W_v x_1 \end{aligned} \tag{3.6}$$

Pri tem je

- x_1 vhodni vektor,
- q_1 poizvedbeni vektor vhoda x_1 ,
- k_1 vektor ključa vhoda x_1 ,
- v_1 vektor vrednosti vhoda x_1 ,
- W_q matrika uteži za izračun vrednosti vektorja poizvedbe,
- W_k matrika uteži za izračun vrednosti vektorja ključa,
- W_v matrika uteži za izračun vrednosti vektorja vrednosti.

V naslednjem koraku izračunamo vrednost trenutnega vhodnega vektorja. Pri izračunu upoštevamo vse ostale vhodne vektorje, ki sestavljajo trenutni vhod.

To storimo tako, da pomnožimo poizvedbeni vektor trenutnega vhoda z vsemi vektorji ključev vseh vhodnih vektorjev. Vse izračunane vrednosti vektorjev nato delimo s kvadratnim korenem dolžine vektorjev q , k in v . Vse rezultate peljemo skozi aktivacijsko funkcijo Softmax. Rezultat funkcije Softmax nam pove, kako bo posamezen vhod vplival na izračun določenega izhoda.

Vsak vektor vrednosti nato pomnožimo s pripradajočo verjetnostjo, ki jo vrne funkcija Softmax. Pridobljene rezultate na koncu seštejemo in pridobimo vektor izhoda (z).

Celoten proces izračuna vektorja izhoda za popravek besede “ato”, v stavku “vozil sem ato” je prikazan na sliki 3.12.













Celoten proces izračuna samopozornosti zaradi hitrejšega računanja opravimo z uporabo matrik. Vektorje poizvedb, ključev in vrednosti zlepimo skupaj v matrike Q , K in V , vektorske vložitve vhodnih vektorjev pa v matriko X . Enačba 3.7 prikazuje končni izračun samopozornosti.

$$\begin{aligned}
 Q &= XW_q \\
 K &= XW_k \\
 V &= XW_v \\
 Z &= \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)
 \end{aligned} \tag{3.7}$$

Pri tem je

- X matrika vektorjev vhodov,
- Q matrika vektorjev poizvedbe,
- K matrika vektorjev ključev,
- V matrika vektorjev vrednosti,
- Z matrika izhodnih vektorjev,
- Softmax aktivacijska funkcija Softmax,

d_k dolžina vektorja ključ.

	vozil	sem	ato
vektorska vložitev besede	x1 	x2 	x3 
vektor poizvedbe	q1 	q2 	q3 
vektor ključa	k1 	k2 	k3 
vektor vrednosti	v1 	v2 	v3 
izračun vrednosti posameznega vhoda	q3*k1=110	q3*k2=90	q3*k3=120
rezultat delimo s korenom dolžine vektorjev (q, v in k)	110/8=13,75	110/8=11.25	110/8=15
rezultate pošljemo skozi aktivacijsko funkcijo Softmax	0,219	0,018	0,763
vrednost vhoda pomnožimo z vektorjem vrednosti	v1 = v1*0,219	v2 = v2*0,018	v3 = v3*0,763
seštejemo vse na novo izračunane vektorje vrednosti			z3 = v1+v2+v3

Slika 3.12: Slika prikazuje postopek izračuna samopozornosti za vhodni vektor “ato”, stavka “vozil sem ato”. Uteži transformerja vnaprej prilagodimo tako, da popravlja napake v stavku. Pri vrednostih aktivacijske funkcije Softmax vidimo, da na izračun izhoda za besedo “ato” najbolj vplivata besedi “vozil” in “avto” najmanj pa beseda “sem”. Beseda “vozil” transformerju pove, da pri popravljanju besede “ato” gre za besedo “avto” in ne recimo za besedo “vato”.

3.11 Večglava pozornost

Komponento samopozornosti razširimo v večglavo pozornost, s čimer povečamo njeno kapaciteto. To nam omogoča, da komponenta usmeri pozornost na več delov besede hkrati. Ena glava samopozornosti se tako lahko nauči usmerjati pozornost na sklanjatve, druga se na primer osredotoča na čas uporabljen v trenutni povedi, spet druge pa se lahko osredotočijo na ostale dele povedi, ki vplivajo na rezultat, ki ga komponenta vrne.

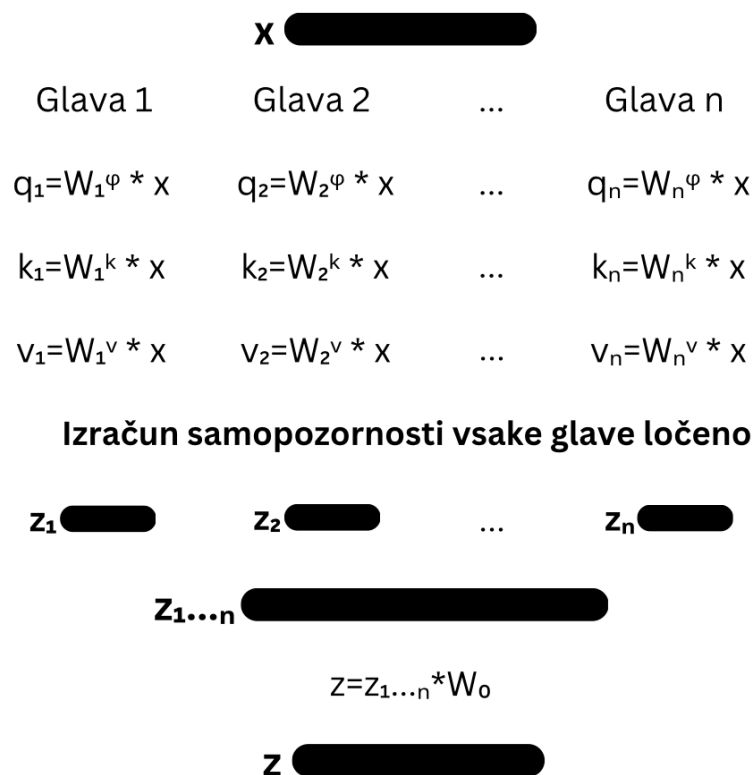
Vzrok za neenakost osredotočanja različnih glav samopozornosti je v času inicializacije in učenja modela. Pred začetkom učenja naključno nastavimo vse vrednosti matrik uteži, kar povzroči, da pri učenju vsaka komponenta poskuša maksimizirati svoj rezultat glede na različno začetno stanje. To v povezavi z ostalimi komponentami privede do boljših rezultatov.

Pri izračunu večglave pozornosti poteka izračun pozornosti vhoda x na enak način, kot je opisano na sliki 3.12. Samopozornost hkrati računamo n -krat, pri tem pa uporabimo različne matrike uteži. Po izračunu vseh n -glav pridobimo n izhodnih vektorjev (z_1, z_2, \dots, z_n) . Vse izhodne vektorje nato združimo v izhodni vektor Z , ki ga pomnožimo z matriko uteži W^o . Na ta način pridobimo izhodni vektor komponente večglave pozornosti.

Celoten proces izračuna večglave pozornosti prikazuje slika 3.13.

3.11.1 Pozicijsko kodiranje v arhitekturi transformer

Pozicijsko kodiranje transformerju omogoča predstavo, kako si sledijo besede na vhodu in kakšna je razdalja med njimi. V tehnologiji transformer preslikamo vsak vhod s pozicijskim vektorjem. Na ta način pri določanju pozicije vhodov izračunamo pozicijsko matriko.



Slika 3.13: Slika prikazuje postopek izračuna večglave pozornosti. V prvem delu izračunamo samopozornost nad vhodnim vektorjem x vsake glave ločeno in izračunamo vektorje z_1, \dots, z_n . Vektorje z_1, \dots, z_n nato združimo v vektor $z_{1...n}$ in ga pomnožimo z matriko uteži W_0 . Tako izračunamo izhodni vektor z , ki je enake dolžine kot vhodni vektor x . Vektor z predstavlja izračunano vrednost večglave pozornosti za vhodni vektor x .

Pozicijski vektor nekega vhoda izračunamo z enačbama 3.8.

$$\begin{aligned}
 P(k, 2i) &= \sin\left(\frac{k}{n^{\frac{2i}{d}}}\right) \\
 P(k, 2i + 1) &= \cos\left(\frac{k}{n^{\frac{2i}{d}}}\right) \\
 0 \leq i &< \frac{d}{2}
 \end{aligned} \tag{3.8}$$

Pri tem je

k pozicija v vhodnem vektorju,

n vnaprej definirana vrednost, ki jo definiramo v času implementacije,

d dimenzija izhoda,

i pozicija v izhodnem pozicijskem vektorju, kjer posamezna vrednost predstavlja kosinusno in sinusno pozicijo.

Vhodni vektor preoblikujemo v dveh korakih. Iz vsake besede najprej izračunamo vektorsko vložitev vhoda, šele na to uporabimo pozicijski vektor vhoda. Pozorni smo na to, da je dolžina vhodnega pozicijskega vektorja (d) enaka dolžini njegove vektorske vložitve. Na koncu oba izračunana vektorja seštejemo in pridobljeni vektor uporabimo kot vhod v kodirnik ali dekodirnik.

3.11.2 Vektorske vložitve v arhitekturi transformer

Modelu transformer kvalitetna predstavitev vektorjev na vhodu omogoča, da že iz besed na vhodu delno razbere njihov pomen.

Najpreprostejša oblika vektorskih vložitev deluje po principu eničnega kodiranja (angl. one-hot encoding), kjer vsak vhod predstavlja vektor, ki je enake dolžine kot celoten slovar vhodov v model. Na mesto, ki ustreza mestu vhodne besede v slovarju, v vektor postavimo enico, na ostala mesta pa 0. Težava takšnega kodiranja je, da so tovrstni vektorji dolgi in nam o pomenu vhoda ali besede, ki jo predstavljamo, ne povedo veliko.

Vektorske vložitve v arhitekturi transformer uporabljajo izboljšano verzijo eničnega kodiranja vektorskih vložitev, pri čemer vsak enično kodiran vhodni vektor pomnožijo z matriko uteži W_v . Dimenzija matrike W_v je enaka dimenziji slovarja transformerja in dimenziji vhodnega vektorja. V originalni arhitekturi transformerjev je dimenzija enaka 512. Produkt vhodnega eno-bitno kodiranega vektorja z matriko uteži je manjši vektor, ki modelu bolje predstavi pomen vhoda. V arhitekturi transformer se matrika W_v napolni z naključnimi vrednostmi, te vrednosti pa se v fazi učenja sprotno prilagajajo. Enačba 3.9 prikazuje končni izračun vhodnega vektorja v arhitekturi

transformer.

$$\begin{aligned} z &= v_0 W_v \\ x &= z + P \end{aligned} \tag{3.9}$$

Pri tem je

v_0 enično zakodiran (one-hot encoded) vhodni vektor,

W_v matrika, ki enično kodiran vhodni vektor preslika v vektorsko vložitev vhoda,

z izračunana vektorska vložitev za vhod v_0 ,

P pozicijski vektor,

x izračunan vhodni vektor za vhod v_0 .

3.12 Učenje s prenosom znanja

Veliki jezikovni modeli, kot je na primer transformer, dosežejo dobre rezultate, vendar potrebujejo veliko procesorske moči in veliko učno množico. Osnovna arhitektura transformerja [50] vsebuje 110 milijonov parametrov, ta številka pa se s kasnejšimi modifikacijami in nadgradnjami še povečuje. Učenje velikih modelov je zaradi tega oteženo, zato so po objavi osnovnega modela transformer razvili veliko modificiranih verzij s poudarkom na učenju s prenosom znanja (angl. transfer learning).

Učenje s prenosom znanja nam omogoča, da znanje, ki ga model pridobi z reševanjem nekega problema prenesemo in uporabimo pri reševanju novega problema. Pri prilagajanju modela na nov problem uporabimo že nastavljene uteži. Le-te še prilagodimo, tako da izboljšamo delovanje modela na novem problemu. Večino računskih virov tako uporabimo za učenje modela na nekem splošnem problemu, model pa nato le prilagodimo novemu problemu.

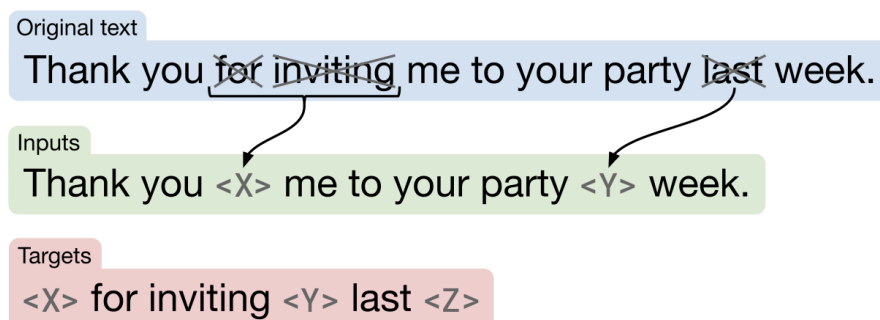
3.13 Model T5

Osnovni model transformerja je bil razvit za preverjanje različnih besedil. T5 (angl. Text-to-text transformer) model [43] skuša združiti več nalog procesiranja naravnega jezika v enem modelu, kjer vhodne probleme in njegove rešitve predstavimo v obliki besedila. To nam omogoča uporabnost modela za reševanje več različnih jezikovnih problemov.

Model je zgrajen na principu učenja s prenosom znanja, opisanem v razdelku 3.12, kjer so avtorji velik poudarek namenili procesu učenja osnovnega modela, ki ga lahko nato prilagodimo reševanju različnih problemov. Za učenje osnovnega modela so uporabili vnaprej pripravljeno podatkovno množico, ki so jo poimenovali C4 [9] (angl. Colossal Clean Crawled Corpus). Podatkovna množica je v osnovi zgrajena spletnih besedil, iz katere so izključili vsa neangleška besedila, izbrisali besedila, ki so vsebovala programsko kodo, odstranili podvojene stavke in drugo.

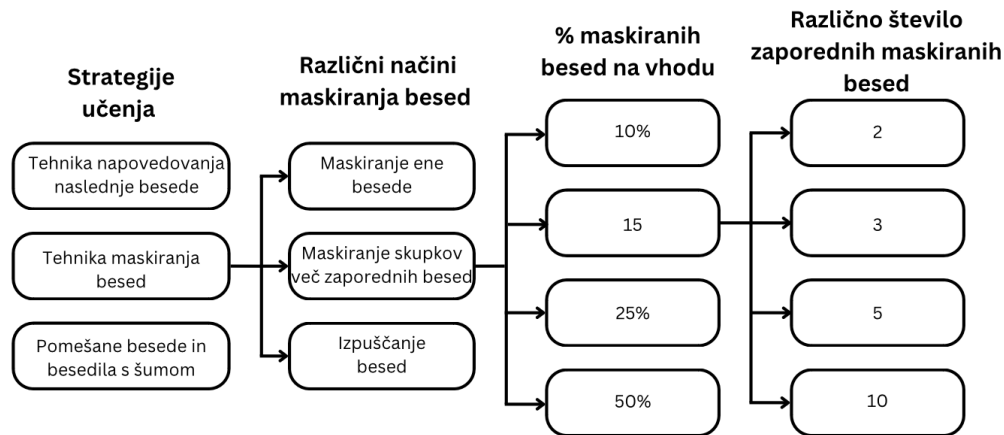
Ena izmed učnih tehnik modela T5 je maskiranje besed, kjer za razliko od učenja modela BERT (razdelek 3.14), namesto ene besede, maskiramo eno ali več zaporednih besed hkrati. Slika 3.14 prikazuje maskiranje besed, ki se uporablja pri osnovnem učenju T5 modela. V procesu učenja iz vhodnega besedila sestavimo besedilo, ki predstavlja vhod v model, in besedilo, ki predstavlja pričakovan izhod.

Avtorji so po opravljenem osnovnem učenju, pri katerem so uporabili korpus C4, model doučili z izpopolnjevanjem na različnih jezikovnih problemih. To so storili z oblikovanjem vhodnih in izhodnih besedil, ki so jih oblikovali glede na izhodno nalogo, ki jo je model moral rešiti. Model so izpopolnili za naloge prevajanja, pisanja povzetkov, primerjanja stavkov in druge. Pri tem so pred vsakim vhodom dodali predpono, ki modelu pove, katero jezikovno nalogo naj opravi. Posledično model prilagodi uteži za reševanje omenjenega problema. Avtorji trdijo, da pri evalvaciji modela ni bilo primera, ko bi model pomotoma na izhod podal rešitev drugega jezikovnega problema, ampak je vedno reševal le problem, ki so ga navedli v predponi vhodnega besedila.



Slika 3.14: Slika je povzeta po Raffel in sod. [43]. Pod oznako “original text” je prikazano besedilo namenjeno učenju modela. Iz tega besedila kreiramo vhodno besedilo v model “Inputs” in pričakovano izhodno besedilo “Targets”. Pri kreiranju vhodov maskiramo besede “for”, “inviting” in “last”. Ker besede “for” in “inviting” pogosto stojijo skupaj, besedi združimo v en žeton <X>, besedo “last” pa v žeton <Y>. Pričakovane izhode iz modela sestavimo tako, da po vrsti napišemo vse uporabljene žetone, za vsakim žetonom pa podamo besedo ali zaporedje besed, ki bi morale stati na mestu žetona. Na koncu vektorja, ki predstavlja pričakovan izhod, podamo žeton <Z>.

Pri učenju osnovnega modela so avtorji preizkusili več različnih strategij, ki so jih razdelili v 4 kategorije. Strategije so prikazane na sliki 3.15.



Slika 3.15: Slika prikazuje različne tehnike učenja, ki so jih avtorji modela T5 uporabili in preizkusili pri učenju osnovnega modela. Pri strategijah učenja so najboljše rezultate dosegli s tehniko maskiranja besed. Pri preizkušanju različnih načinov maskiranja besed so najboljše rezultate dosegli z maskiranjem skupkov večih zaporednih besed. Pri povečevanju odstotka maskiranih besed, so najboljše rezultate dosegli, če so maskirali 15 odstotkov vhodov. Pri preizkušanju maskiranja različnega števila zaporednih besed, so najboljše rezultate dosegli pri maskiranju dveh besed.

Preizkušali so različne tehnike osnovnega učenja modela. V prvem koraku so model poskušali naučiti s tehniko, kjer je model na podlagi prejšnjih besed skušal napovedati naslednjo besedo v nizu. Naslednja tehnika, ki so jo preizkusili je tehnika maskiranja besed, kjer so nekatere besede na vhodu maskirali, pri tem je moral model v fazi učenja napovedovati njihove vrednosti. Na koncu so skušali model učiti z besedili, kjer so bili deli besed pomešani ali pa so med besede vrinili šum. Izmed vseh strategij učenja vnaprej so najboljše rezultate dosegli s tehniko maskiranja besed. V naslednjem koraku so eksperimentirali z različnim načini maskiranja besed. Preizkusili so tehniko maskiranja ene besede, tehniko maskiranja skupkov več zaporednih besed in tehniko izpuščanja besed. Med temi tehnikami so najboljše rezultate dosegli s tehniko maskiranja večih zaporednih besed. V naslednjem koraku so sku-

šali maskirati 10%, 15%, 25% in 50% razponov besed. Najboljše rezultate so dosegli z maskiranjem 15 odstotkov razponov besed. Nazadnje so testirali različne dolžine maskiranih razponov besed, kjer so ugotovili, da z daljšanjem razpona besed dobijo slabše rezultate. Najboljše rezultate so dobili z maskiranjem dveh zaporednih besed.

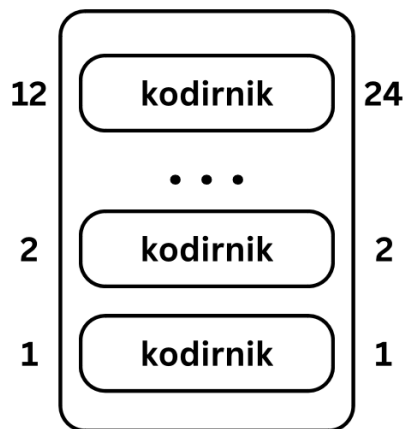
3.13.1 Arhitektura modela T5

Zgradba modela T5 temelji na originalni arhitekturi predstavljeni v [50] z nekaterimi spremembami. Prva dopolnitev je v zgradbi normalizacijske plasti, ki jo podrobneje opišemo v razdelku 3.9, kjer pri izračunu izhoda normalizacijske plasti ne prištejejo vrednost pristranskosti (β). Poleg tega prestavijo normalizacijsko plast iz rezidualne povezave. Naslednja nadgradnja je v implementaciji nevronske mreže, kjer uporabljajo tehniko opustitve nekaterih nevronov. Ta tehnika v splošnem nevronske mreži preprečuje, da bi se preveč prilagodila učni množici. Naslednja dopolnitev je v zgradbi polno povezane nevronske mreže, ki izhod zadnjega dekodirnika preslika v izhod modela. Omenjena nevronske mreže si deli vhodne uteži z vložitveno matriko vhodov, kar pomeni, da so vložitve vhodnih besed in uteži te nevronske mreže povezane in se med učenjem skupaj prilagajajo. Pri generiranju vhodov v model, namesto fiksne pozicijskega kodiranja, ki ga opišemo v 3.11.1, model T5 uporablja poenostavljeno relativno pozicijsko kodiranje besed. Pozicijski vektorji besed se pri izračunu samopozornosti med seboj razlikujejo glede na oddaljenost vektorja ključ (k) in vektorja poizvedbe (q).

Osnovni model T5 je zgrajen iz 12 modulov kodirnikov in dekodirnikov. Nevronska mreža v vsakem modulu na izhod poda vektor dimenzije 3072. Vsa ostala skrita stanja in vektorske vložitve modela imajo dimenzionalnost 768. Vsaka komponenta večglave pozornosti je zgrajena iz 12 glav samopozornosti. Celoten model ima tako 220 milijonov parametrov.

3.14 Model BERT

Enega večjih korakov pri razvoju predstavitvenih modelov za naloge procesiranja naravnega jezika predstavlja model BERT [10]. Osnovna zgradba modela BERT temelji na arhitekturi transformerjev, pri čemer pa je vključen le del s kodirniki. Na sliki 3.16 je prikazana preprosta zgradba modela BERT.



Slika 3.16: Slika prikazuje preprosto zgradbo modela BERT, ki je zgrajen iz kodirnikov, ki so naloženi drug na drugega. Struktura posameznega kodirnika je opisana v razdelku 3.9. Števila levo predstavljajo model $BERT_{BASE}$, ki je zgrajen iz 12ih kodirnikov, števila desno pa model $BERT_{LARGE}$, ki je zgrajen iz 24ih kodirnikov.

Avtorji so predstavili dva modela $BERT_{BASE}$ in $BERT_{LARGE}$. Modela se razlikujeta predvsem po številu in velikosti komponent. $BERT_{BASE}$ je zgrajen iz 12ih kodirnikov, $BERT_{LARGE}$ pa iz 24ih kodirnikov, ki so naloženi drug na drugega. Poleg števila kodirnikov se modela razlikujeta v številu skritih nevronov v nevronske mreži, ki se nahaja v vsakem kodirniku. Nevronske mreže v $BERT_{BASE}$ modelu vsebujejo 768, v $BERT_{LARGE}$ pa 1024 skritih nevronov. Modela se razlikujeta tudi v številu glav pozornosti. Vsaka komponenta večglave pozornosti $BERT_{BASE}$ vsebuje 12 glav pozornosti, mo-

dela $BERT_{LARGE}$ pa 16 glav pozornosti. Posledično $BERT_{LARGE}$ vsebuje večje število parametrov, kar poveča čas učenja modela. V kolikor imamo na razpolago dovolj računskih virov in podatkov, prinaša uporaba večjega modela načeloma boljše rezultate.

3.14.1 WordPiece kodiranje

Vektorske vložitve pri modelu BERT so narejene po principu izboljšanega algoritma združevanja parov (BPE ali Byte-pair encoding algoritem), ki pripravi predstavitev podbesed (angl. subwords). Glavna lastnost algoritmov, ki generirajo podbesede je v tem, da namesto vhoda iz celih besed uporabljajo njihove dele.

BPE algoritem deluje tako, da med vsemi pari črk v trenutnem slovarju vzame tisti par, ki se v besedilu največkrat nahaja skupaj in ga združi v novo črko, postopek pa ponavlja. Težava pri algoritmu BPE se pojavi, ko imata dva para v učnem besedilu enako število ponovitev. V tem primeru algoritem nima definirane dobre strategije, kateri par izbrati, kar lahko privede do tega, da algoritem na vsakem koraku ne izbere optimalne možnosti in s tem generira slabše vektorske vložitve.

Algoritem WordPiece [45] deluje na podoben princip kot algoritem BPE. Za razliko od algoritma BPE WordPiece združuje tiste podbesede, katerih združitve bodo privedle do največjega povečanja verjetnosti pri združevanju. Primer: če se v našem slovarju nahajata črki "a" in "v" in je verjetnost, da se v besedilu naključno pojavi črka "a", ki ji sledi črka "v", manjša od verjetnosti, da se pojavita črki "av" skupaj, potem črki lahko združi. Izračun povečanja verjetnosti združitve dveh delčkov besed prikazuje enačba 3.10.

$$y = \frac{p(k)p(z)}{p(kz)} \quad (3.10)$$

Pri tem je

- y vrednost povečanja verjetnosti po združitvi dveh delčkov besed,
- $p(k)$ verjetnost, da se v besedilu pojavi delček k ,

$p(z)$ verjetnost, da se v besedilu pojavi delček z ,
 $p(kz)$ verjetnost, da se v besedilu delčka k in z pojavita skupaj.

Algoritem črke združuje tako, da na vsakem koraku združi delčke besed, katerih verjetnost je največja.

3.14.2 Kreiranje vhodov v model BERT

Modelu BERT na vhodu podamo zaporedje žetonov, ki istočasno potujejo skozi model. Ker model vse vhode obravnava istočasno, mu to omogoča, da se pri obravnavi posameznega vhoda osredotoči na vhode, ki stojijo pred in po trenutno obravnavanem vhodu.

Rezultat algoritma WordPiece 3.14.1 je slovar črk in delčkov besed, ki se v učni množici največkrat pojavljajo skupaj. Pri obravnavanju posameznega vhoda model BERT vsako besedo razdeli na posamezne delčke, pri čemer vsako besedo predstavi s čim večjimi delčki besed, ki so prisotni v slovarju. V primeru, da se v slovarju nahaja celotna beseda, model uporabi to besedo. V primeru, da se v slovarju ne nahaja noben delček trenutne besede, besedo razdeli na posamezne črke. V primeru, da določenega delčka ali znaka besed ne more najti v slovarju, ta delček označi s posebnim žetonom [UNK]. Vsak delček besede, ki prapravlja nadaljevanje besede, vsebuje predpono, ki jo označimo z dvema znakoma “#”. Poleg omenjenih žetonov, model BERT uporablja še žeton [CLS], ki nakazuje začetek celotnega vhoda in žeton [SEP], ki predstavlja konec vhoda. V naslednjem koraku model BERT vsakemu žetonu dodeli identifikacijsko število, ki je enako zaporednemu številu žetona v slovarju. Tabela 3.14.2 prikazuje ločitev vhodnega niza modela BERT na posamezne žetone.

Vhodni stavek	Vozil sem avto.
Tokeniziran stavek	[CLS] Vozil sem av ##to . [SEP]

Zaporedno število posameznega žetona predstavlja mesto v matriki vektorskih vložitev, ki jo model v času učenja prilagaja. Vsaka vrstica v matriki vektorskih vložitev predstavlja vektorsko vložitev žetona.

3.14.3 Pozicijsko kodiranje pri modelu BERT

Model vsaki vektorski vložitvi poda informacijo, na kateri poziciji se posamezen žeton nahaja. Za izračun pozicijske informacije model BERT uporablja kombinacijo trigonometričnih funkcij sinus in kosinus. Primer izračuna pozicijskega vektorja za nek žeton v modelu BERT prikazuje enačba 3.11.

$$p_{i,j} = \begin{cases} \sin\left(\frac{i}{10000 \left(\frac{j}{d_{emb_dim}}\right)}\right) \text{ j \% } 2 == 0 \\ \cos\left(\frac{i}{10000 \left(\frac{j-1}{d_{emb_dim}}\right)}\right) \text{ j \% } 2 != 0 \end{cases} \quad (3.11)$$

Pri tem je

$p_{i,j}$ j-ta vrednost i-tega pozicijskega vektorja,

d_{emb_dim} dolžina vektorske vložitve vhodnega vektorja.

3.14.4 Tehnike učenja modela BERT

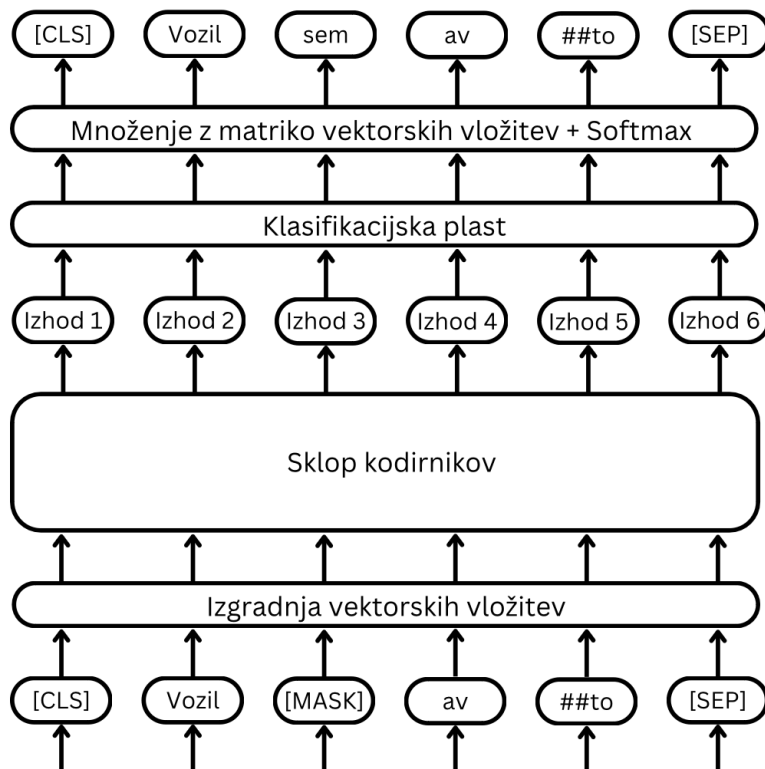
Model BERT temelji na arhitekturi kodirnika z nalogo, da iz vhodnega besedila razbere kontekstne informacije, ki se v besedilu nahajajo. Model prebere in obdela vse vhode hkrati, zato na vsakem koraku gleda na vhod iz obeh strani.

Pred objavo arhitekture modela BERT se je pri učenju večine jezikovnih modelov uporabljalo napovedovanje naslednje besede v besedilu. Ta tehnika je za model BERT slaba, saj naj bi model na vsakem koraku za dobro predstavitev besedila videl vse vhode v model. Model zato uporablja dve tehniki učenja.

Prva tehnika učenja je maskiranje besed, pri kateri 15% naključnih žetonov označijo z možnostjo maskiranja. Od teh 15% označenih žetonov jih nato 80% zamenjajo z žetonom [MASK], 10% žetonov pustijo nespremenjenih, ostalih 10% pa zamenjajo z naključnim žetonom iz slovarja. Na podlagi ostalih žetonov v fazi učenja model poskuša ugotoviti prvotno vrednost.

Pri učenju modela s tehniko maskiranja besed pripnemo na zadnjo plast modela, ki generira izhode, še klasifikacijsko plast. To sestavlja polnopravna

zana mreža, ki ji sledi aktivacijska funkcija GELU in normalizacijska plast. Izhode klasifikacijske plasti množimo z matriko vektorskih vložitev in rezultat peljemo skozi aktivacijsko funkcijo Softmax. Slika 3.17 prikazuje celoten proces klasifikacije maskiranih žetonov pri modelu BERT.

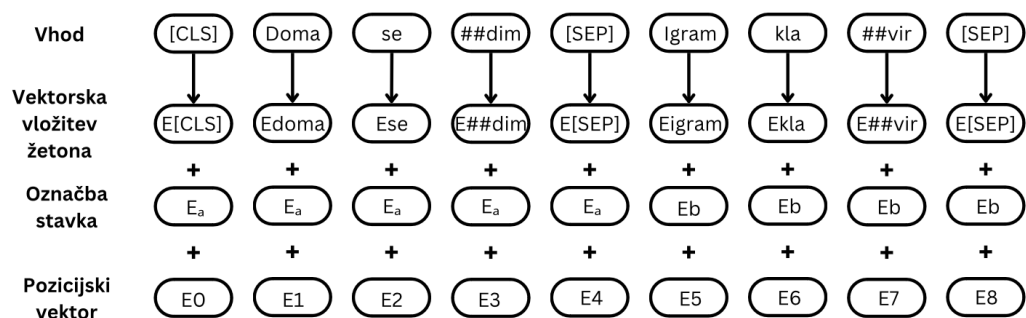


Slika 3.17: Potek izračuna izhodnega vektorja pri modelu BERT s tehniko maskiranja besed.

Naslednja tehnika učenja modela BERT je napovedovanje sosledja stavkov. Pri tem modelu na vходу podamo dva stavka, drugega za drugim. Model skuša ugotoviti ali drugi stavek v originalnem besedilu sledi prvemu ali ne.

Pri izgradnji učne množice za učenje modela BERT so avtorji iz učne množice izbrali stavke tako, da 50% stavkov ni sledilo, 50% pa je sledilo

prvemu stavku. Pri učenju modela z napovedovanjem sosledja stavkov so vhode v model pripravili tako, da so konec vsakega stavka označili z žetonom [SEP]. Poleg tega so vsak vhod v model zgradili iz treh vektorjev. Pri izračunu vhoda so sešteli vektorsko vložitev vhoda, vložitev stavka, ki modelu pove ali pri vhodu gre za prvi ali drugi stavek, in pozicijski vektor. Primer izgradnje vhoda v model pri napovedovanju naslednjega stavka prikazuje slika 3.18.



Slika 3.18: Slika prikazuje izgradnjo vhodnih vektorjev pri učenju modela z napovedovanjem naslednjega stavka. Vhodni vektor v model pridobimo s seštevkom vektorske vložitve vhoda, pozicijskega vektorja in vektorja označbe stavka, ki označuje ali gre za stavek A ali stavek B.

Za izhod modela avtorji vzamejo izhodni vektor začetnega žetona [CLS], ki ga peljejo skozi linearno nevronske plast. Izhod nevronske plasti je 2×1 vektor. Vektor nato peljejo skozi aktivacijsko funkcijo Softmax, katere izhod predstavlja dve vrednosti. Prva vrednost predstavlja verjetnost, da druga poved sledi prvi povedi, druga vrednost pa predstavlja verjetnost, da druga poved ne sledi prvi povedi.

3.14.5 Izpopolnjevanje modela BERT

Enako kot pri arhitekturi transformer igra izpopolnjevanje vnaprej naučenega modela BERT pomembno vlogo. Omogoča hitrejšo pot do kvalitetnejših

rezultatov, saj v fazi izpopolnjevanja uporabimo že vnaprej naučen osnovni model, ki ga na koncu le še izpopolnimo za določeno nalogo.

Pri prilagajanju modela BERT lahko uporabimo več tehnik izpopolnjevanja modela. Pri vsaki tehniki na osnovno zgradbo modela pripnemo dodatno plast. Ta nam omogoča izgradnjo izhodov, ki jo pri določeni tehniki izpopolnjevanja potrebujemo.

Ena izmed nalog izpopolnjevanja modela je klasifikacija. Deluje na podoben princip kot učenje napovedovanja naslednjega stavka. V tem primeru lahko nek vhod klasificiramo tudi v več kategorij.

Naslednja naloga je odgovarjanje na vprašanja. Model na vhod dobi besedilo in vprašanje. Na izhodu poda lokacijo odgovora na to vprašanje v vhodnem besedilu.

Tretja naloga je prepoznavanje imenskih entitet. Model lahko skuša tudi oblikoskladensko označiti vhodno besedilo.

Smiselna naloga je tudi maskiranje besed. Model izpopolnjujemo s podobno tehniko, kot je bil naučen. V tem primeru se model samo douči, kaj mora napovedati na mestih, ki jih označujejo maskirni žetoni.

3.14.6 RoBERTa

Po objavi modela BERT so bile predstavljene številne izboljšave osnovnega modela. Liu in sod. [31] predstavijo model RoBERTa, ki se od osnovnega modela BERT razlikuje predvsem po tem, da je bil naučen na veliko večji podatkovni množici. Za razliko od tehnike maskiranja besed pri originalnem modelu BERT, predstavljenem v razdelku 3.14.4, so avtorji pri modelu RoBERTa uporabili dinamično maskiranje besed. Pri tem so vseh 15% žetonov označenih za maskiranje, zamenjali z žetonom [MASK]. Ta proces so ponovili 10-krat. Pri vsaki iteraciji so izberali novih 15% žetonov.

Model RoBERTa za razliko od osnovnega modela BERT iz procesa učenja povsem odstrani napovedovanje naslednjega stavka.

Poglavje 4

Podatkovna množica

Podatkovno množico za obvladovanje jezikovnih napak zgradimo iz korpusa Gigafida [28]. Korpus vsebuje okoli 1.1 milijarde besed, ki so razdeljene med 39.427 dokumentov in 99 sklopov. Korpus je zgrajen iz besedil časopisov, revij, spletnih besedil, knjižnih publikacij in drugih virov.

Vsak dokument v korpusu je zapisan v xml obliki. V prvem delu dokumenta so prisotne informacije o izvoru in lokaciji besedila, nato sledi vsebina dokumenta, ki je ločena na stavke. Začetek in konec stavka sta označena z oznakama `<s xml:id="enolični identifikator stavka">` in `</s>`. Znotraj stavka zaporedoma sledijo posamezne besede označene z oznako `"w"`, števila z oznako `"f"`, presledki z oznako `"c"` in vezniki z oznako `"pc"`. Vsaka beseda vsebuje dva atributa. Atribut `"ana"`, v katerem je zapisana analiza besede, in atribut `"lemma"`, kjer je zapisana lematizirana oblika besede. Slika 4.1 prikazuje primer stavka "Vsaka prestolnica ima svoj Panteon, počivališče pomembnih mož."

```

<s xml:id="GF7690878.5.1">
  <w ana="mte:Zc-zei" lemma="vsak">Vsaka</w><c> </c>
  <w ana="mte:Sozei" lemma="prestonica">prestonica</w><c> </c>
  <w ana="mte:Ggnste-n" lemma="imeti">ima</w><c> </c>
  <w ana="mte:Zp-met" lemma="svoj">svoj</w><c> </c>
  <w ana="mte:Sometr" lemma="panteon">Panteon</w>
  <pc ana="mte:U">,</pc><c> </c>
  <w ana="mte:Sosei" lemma="počivališče">počivališče</w><c> </c>
  <w ana="mte:Ppnmmr" lemma="pomembni">pomembnih</w><c> </c>
  <w ana="mte:Sommr" lemma="mož">mož</w>
  <pc ana="mte:U">.</pc><c> </c>
</s>

```

Slika 4.1: Primer stavka “Vsaka prestonica ima svoj Panteon, počivališče pomembnih mož.”, ki je zapisan v izvornem xml dokumentu korpusa Gigafida.

Iz zbirke pripravimo učno, validacijsko in testno podatkovno množico. Za izgradnjo testne množice uporabimo sklope dokumentov 1-4, za izgradnjo validacijske in učne podatkovne množice pa sklope dokumentov 5-99. Posamezni sklopi dokumentov se med seboj vsebinsko ne povezujejo. Testno in učno množico gradimo po enakem principu. V podatkovno množico vključimo vse dokumente posameznega sklopa dokumentov, pri tem pa iz vsakega dokumenta vzamemo 20 naključnih stavkov. Učna in validacijska množica tako vsebujeta 2,413.478 povedi. 2000 povedi uporabimo za validacijsko množico, ostale povedi pa za učno množico.

Za procesiranje xml datotek uporabimo knjižnico glob. Informacije o posamezni besedi in njeni lemi pri izgradnji podatkovne množice zavrzemo. Iz povedi odstranimo podvojene presledke, presledke pred ločili, izločimo stavke, kjer večji del stavka sestavljajo števila, in zamenjamo nekatere posebne znake. Zamenjane znake in njihove zamenjave prikazuje tabela 4.

Sklop znakov	zamenjava
[!@#\$%\:; '""'“”●\«\»%&ø ¹]	
öó	o
áä	a
ë	e
í	i
ü	u
ć	c

Podatkovno množico sestavljajo povedi, ki so zapisane v vrsticah. Vsaka poved se konča z znakom za novo vrstico. Podatkovno množico naprej dodatno prilagodimo za učenje posameznega jezikovnega modela.

4.1 Korpus Šolar 3.0

Za testiranje nekaterih modelov poleg testne množice zgrajene iz korpusa Gigafida uporabimo korpus Šolar 3.0 [4].

Korpus je zgrajen iz pisnih besedil srednješolcev in osnovnošolcev ter vsebuje več kot milijon besed. Skoraj polovica besedil vsebuje ročne popravke učiteljev. Vsak popravek v besedilih je označen, pri čemer je označena tudi vrsta napake.

Za branje in procesiranje korpusa uporabimo knjižnico datasets iz zbirke Hugging Face. Testno podatkovno množico iz zbirke Šolar gradimo le na ročno popravljenih in pregledanih besedilih. Ta besedila imajo v korpusu posebno oznako (is_manually_validated) postavljeno na true.

4.2 Sloleks 2.0

Sloleks 2.0 [11], je leksikon slovensih besednih oblik. Vsebuje 100.802 iztočnic in 2.792.003 posameznih besednih oblik. V leksikonu največji delež obsegajo samostalniki, sledijo jim pridevniki, nato glagoli, prislovi, števniki in ostale

besedne vrste. Leksikon uporabimo predvsem za preverjanje, ali je neka beseda, ki jo model napove, pravilna ali ne.

4.3 Slovar besed

Korpus Gigafida uporabimo za izgradnjo lastnega slovarja besed. Pri izgradnji slovarja pregledamo celotno podatkovno zbirko Gigafida in sproti polnemo dva slovarja. V prvem slovarju, ki ga poimenujemo slovar lem, kot ključ uporabimo besedo, za vrednost pa njeno lemo. Če je neka beseda že v slovarju, besede ne zapisujemo.

V drugem slovarju, ki ga poimenujemo slovar besed, kot ključ uporabimo lemo besede, za vrednost pa besedno obliko. Vrednost tako predstavlja seznam vseh besednih oblik za lemo. Če je neka beseda že v slovarju, jo v slovar besed vseeno podamo, saj s tem ohranimo informacijo o pogostosti besed za lemo.

Slovar besed uporabimo za učenje nekaterih jezikovnih modelov.

Poglavje 5

Arhitektura rešitve

V razdelku 5.1 predstavimo izpopolnjen T5 model namenjen označevanju napačno zapisanih in napačno črkovanih besed. Opišemo, kako iz učne množice pripravimo vhode v model in na kaj smo pri pripravi posameznih vhodov pozorni. Nato predstavimo potek učenja modela in potek testiranja. V nadaljevanju predstavimo model SloBERTa namenjen popravljanju označenih besed ter pripravo učne množice, proces učenja in testiranja modela.

V razdelku 5.3 predstavimo znakovni model T5 namenjen popravljanju napačno črkovanih besed. Nato opišemo model T5 namenjen popravljanju vrstnega reda besed v stavkih. Na koncu predstavimo še model T5 namenjen pregibanju besed. Pri vsakem modelu opišemo pripravo učne množice, postopke učenja in testiranja modelov.

5.1 Model T5 za označevanje napačno zapisanih besed

V tem razdelku opišemo pripravo učne množice za učenje modela T5 za označevanje napačno zapisanih besed. Opišemo tudi proces učenja in testiranja modela.

5.1.1 Priprava učne množice

Učno množico gradimo iz pripravljene zbirke besedil opisane v poglavju 4. Zbirko besedil sestavljajo povedi, ki so zapisane v vrsticah. Pri sestavi učne množice beremo vsako poved posebej. V vsaki povedi poskrbimo, da se pred in za vsakim ločilom nahaja presledek. Pri oblikovanju učne množice gradimo dve zbirki povedi. Prva zbirka predstavlja napačne (pokvarjene) povedi, ki jih v času učenja in testiranja podamo na vhod modela. Druga zbirka predstavlja željeno (pravilno) obliko povedi, ki predstavljajo pričakovan izhod modela.

Iz povedi odstranimo podvojene presledke. V vsaki povedi začnemo naključno izbirati besede, ki jih spreminjamo z naključno izbranim postopkom. Naključno izbrane postopke opišemo spodaj. Vsakemu postopku dodelimo verjetnostni faktor, ki sporoča verjetnost izbora postopka pri spreminjanju besed.

• Naključna ločitev besede

Pri naključni ločitvi besede izberemo naključno mesto v besedi in na izbrano mesto vrinemo presledek. Na ta način iz ene pravilne besede pridobimo dve napačni besedi. V zbirko napačnih povedi zapišemo dve besedi, v zbirko pravilnih povedi pa dvakrat zapišemo oznako `<splited_text>`. Primer naključne ločitve besede "avto" v stavku "Vozil sem avto." prikazuje spodnja tabela. Z zaznavo ločenih besed model skozi proces učenja naučimo zapisa skupaj ali narazen.

Osnovna poved	Vozil sem avto .
Pokvarjena poved	Vozil sem av to .
Pravilna poved	Vozil sem <code><splited_text></code> <code><splited_text></code> .

Pri pripravi učne in testne podatkovne množice metodo naključne ločitve pri obravnavi določene besede uporabimo s 3% verjetnostjo, ki jo določimo s poskušanjem. Pri določanju verjetnosti upoštevamo, da hkrati v besedilu simuliramo tudi druge napake. Ker želimo kljub velikemu številu napak čimbolj ohraniti izviren pomen besedila, verjetnosti simuliranja posameznih napak ohranimo nizke.

• Naključno združevanje besed

Pri metodi naključnega združevanja besed trenutno obravnavano besedo združimo z besedo, ki ji sledi. V zbirko napačnih povedi zapišemo združeno besedo, v zbirko pravih povedi pa oznako `<concatated_word>`. Primer naključnega združevanja besed “Vozil” in “sem” v povedi “Vozil sem avto.” prikazuje spodnja tabela. S pomočjo omenjene metode model naučimo zaznati, ali bi določeno besedo morali zapisati narazen.

Osnovna poved	Vozil sem avto .
Pokvarjena poved	Vozilsem avto .
Pravilna poved	<code><concatated_word></code> avto .

Pri pripravi učne in testne podatkovne množice metodo naključnega združevanja pri obravnavi določene besede uporabimo s 3% verjetnostjo.

• Zamenjava besed “nagajivk”

Besede nagajivke so besede, ki jih veliko ljudi piše napačno. V večini primerov te besede drugače izgovorimo, kot jih zapišemo. Zbrano zbirko besed zapišemo v dva slovarja. V prvi slovar zapišemo pravilno zapisane besede, v drugi slovar pa najbolj pogoste napačno zapisane različice. Primer prvih desetih besed “nagajivk” prikazuje spodnja tabela. Slovar v celoti vsebuje 85 besed.

Napačen zapis besede	Pravilen zapis besede
golop	golob
vrak	vrag
hrip	hrib
blaš	blaž
poglet	pogled
krok	krog
mras	mraz
zemljevit	zemljevid
matevš	matevž
nagopčnik	nagobčnik
.

V primeru, da se trenutno obravnavana beseda nahaja v slovarju “nagajivk”, v zbirko napačnih povedi zapišemo napačen zapis besede, v zbirko pravih povedi pa oznako <mask>.

Pri pripravi učne in testne podatkovne množice, to metodo pri obravnavi določene besede uporabimo z 10% verjetnostjo.

- **Zamenjava pogosto napačno uporabljenih črk**

Pri zamenjavi pogosto napačno uporabljenih znakov si pomagamo s tabelo “nagajivk”, ki jo opišemo v razdelku 5.1.1. Iz vseh zbranih pogosto napačno zapisanih besed izluščimo znake, ki se pri napačnem zapisovanju besed pogosto zamenjajo. Zbrane znake zberemo v preslikovalno tabelo, ki jo uporabimo pri gradnji zbirke napačno zapisanih povedi. To storimo tako, da iz preslikovalne tabele izberemo naključno zamenjavo in če se znak za zamenjavo v trenutno obravnavani besedi nahaja, znak zamenjamo. V primeru, da se znak za zamenjavo v trenutno obravnavani besedi ne nahaja, naključno iz preslikovalne tabele izberemo nov znak. To ponovimo štirikrat in v primeru, da po štirih poskusih zamenjave ne naredimo, obravnavano besedo pustimo nespremenjeno.

V primeru, da besedo spremenimo, v zbirko vseh nepravilnih povedi zapišemo napačno besedo, v zbirko pravih povedi pa zapišemo oznako <mask>.

Spodnja tabela prikazuje seznam vseh uporabljenih zamenjav črk. Uporabljena tabela je enkrat večja od prikazane, saj zamenjave črk opravljamo v obeh smereh.

črka	zamenjava
nj	n
l	lj
t	d
v	u
u	el
i	j
k	kj
k	h
k	g
s	z
p	b
š	ž
v	l
u	l
t	tj

Metodo zamenjave pogosto napačno uporabljenih črk pri obravnavi posameznih besed uporabimo s 70% verjetnostjo. Verjetnost izbire metode je višja, saj se v večini primerov ena izmed štirih izbranih črk ne nahaja v trenutno obravnavani besedi. Dejanjskih zamenjav črk je tako manj.

• Zamenjava šumnikov

Pri metodi naključne zamenjave šumnikov v trenutno obravnavani besedi nadomestimo vse šumnike. Tako na primer besedo “smučišče” nadomestimo z besedo “smucisce”. V primeru, da v besedi nadomestimo šumnike, v zbirko napačnih povedi zapišemo besedo brez šumnikov, v zbirko pravih povedi pa oznako <mask>. Primer zamenjave šumnikov v povedi “Danes smučam.” prikazuje spodnja tabela.

Osnovna poved	Danes smučam .
Pokvarjena poved	Danes smucam .
Pravilna poved	Danes <mask> .

Metodo zamenjave šumnikov uporabimo s 5% verjetnostjo.

• Zamenjava naključnih znakov v besedi

Metodo zamenjave naključnih znakov razdelimo na 5 delov. Najprej v trenutno obravnavani besedi z verjetnostjo 5% zamenjamo naključno izbrani samoglasnik. V trenutno obravnavani besedi kasneje s 5% verjetnostjo zamenjamo naključno izbrani soglasnik. Nato z 2% verjetnostjo v besedi zamenjamo naključno izbrano črko. Kasneje v besedi s 4% verjetnostjo naključno črko izbrišemo in na koncu s 3% verjetnostjo na naključno mesto dodamo naključno črko. Vsako od omenjenih metod z enako verjetnostjo ponovimo. Če na primer v besedi “avto” dvakrat zamenjamo naključni samoglasnik in enkrat na naključno mesto dodamo naključno črko, lahko iz besede avto dobimo besedo “avita”. Spodnji izraz prikazuje verjetnost, da v neki besedi dvakrat zamenjamo naključno izbran samoglasnik in enkrat na poljubno mesto dodamo poljubno črko. V enačbi privzamemo, da izbrana beseda vsebuje soglasnike in samoglasnike.

$$0.05 \cdot 0.05 \cdot 0.95 \cdot 0.95 \cdot 0.98 \cdot 0.96 \cdot 0.03 \cdot 0.97 = 0,000062$$

Če v trenutno izbrani besedi zamenjamo nekatere črke, v zbirko napačnih povedi zapišemo spremenjeno besedo, v zbirko pravih povedi pa zapišemo oznako <mask>.

• Končna podatkovna množica

Končna učna množica namenjena učenju modela za označevanje napačno zapisanih besed vsebuje 2,417.200 povedi. Vsako poved obdelamo z metodami opisanimi zgoraj. Učno množico naprej razdelimo na učno in validacijsko, pri čemer validacijsko množico sestavlja prvih 2000 povedi.

Testno podatkovno množico sestavimo na enak način kot učno množico. Ta množica vsebuje 214.000 povedi.

Spodnja tabela prikazuje primer naključnih stavkov, ki jih uporabimo pri učenju modela T5. Prvi stavek predstavlja vhod, ki ga podamo v model, drugi stavek pa njegov pričakovan izhod.

Izvirna poved	In prav vsi gredo čez enak postopek obravnave .
Vhod v model	In prau usi gredo čez enak postojek obravnave .
Pričakovan izhod	In <mask> <mask> gredo čez enak <mask> obravnave .

V tabeli lahko vidimo, da v postopku priprave vhodne povedi, popačimo besede “prav”, “vsi” in “postopek”. V prvih dveh besedah črko “v” zamenjamo s črko “u”, v drugi besedi pa črko “p” s črko “j”. Vse popačene besede so v pričakovani izhodni povedi modela označene z oznako <mask>.

5.1.2 Izpopolnjevanje modela T5 za označevanje napačno zapisanih besed

Učenje modela izvedemo z izpopolnjevanjem vnaprej naučenega slovenskega T5-sl-small modela [48]. Osnoven model je zgrajen iz osmih kodirnikov in osmih dekodirnikov in vsebuje okrog 60 milijonov parametrov. Pri učenju modela so uporabili besedila korpusov Gigafida 2.0 [28], Kas 1.0 [14], Janes 1.0 [15], siParl 2.0 [40] in slWac [16].

V prvem delu učenja pripravimo dve csv datoteki (“train.csv” in “validate.csv”). Prva datoteka predstavlja učno druga pa validacijsko množico. Vsaka csv datoteka vsebuje dva stolpca z imeni (“input” in “target”). Stolpec “input” predstavlja vhode v model, ki so zapisani v vrsticah. Vsaka vrstica predstavlja ločeno poved. Stolpec “target” predstavlja pričakovane izhodne povedi modela.

Pri procesu izpopolnjevanja modela uporabljamo knjižnice za delo z modeli tipa transformer iz zbirke Pytorch. Najprej iz vhodnih vektorjev pripravimo generator, ki modelu med procesom učenja podaja vhode. Generator modelu ob vsaki iteraciji poda 4 vektorje, in sicer vektor identifikatorjev žetonov vhoda, vektor pozornosti kodirnika, vektor identifikatorjev žetonov vektorja pričakovanega izhoda in vektor pozornosti dekodirnika. Vektor iden-

tifikatorjev žetonov predstavlja vektor zaporednih števil žetonov v slovarju modela.

Namesto obravnavanja besed v celoti model T5 besede razbije na posamezne žetone. Pri tem uporablja žetone, ki so zapisani v slovarju modela. Pri razbitju besede na več žetonov se prvi žeton vedno začne z znakom `_`. Razbitje besed na več žetonov modelu omogoča obravnavanje neznanih besed in boljšo predstavo o podobnosti besed.

V spodnji tabeli prikažemo primer izgradnje vseh vektorjev, ki jih potrebujemo pri pripravi vhodov za učenje modela. Kot vhodni vektor vzamemo poved “Vozil sm avto.”, kot njen pričakovani izhodni vektor pa “Vozil <mask> avto.”.

Vhodni vektor	Vozil sm avto .
Vektor žetonov vhoda	<code>_Vo zil _sm _avto</code> <code>_. </s> <pad> <pad></code>
Vektor identifikatorjev žetonov vhoda	1518 5053 3310 670 1662 1 0 0
Vektor pozornosti kodirnika	1 1 1 1 1 1 0 0
Vektor pričakovanega izhoda	Vozil <mask> avto .
Vektor žetonov izhoda	<code>_Vo zil _< ma sk ></code> <code>_avto </s></code>
Vektor identifikatorjev žetonov izhoda	1518 5053 413 44 1676 31415 670 1
Vektor pozornosti dekodirnika	1 1 1 1 1 1 1 1 1

Vektorje zaradi lažjega prikaza omejimo na dolžino 9, v končni implementaciji pa uporabimo vektorje dolžine 64 znakov. Če je vektor na vhodu v model daljši od 64 žetonov, ga porežemo na to dolžino. Če je vektor na vhodu v model krajši od 64 žetonov, vektor podaljšamo s posebnimi žetoni `<pad>`.

Za namen sprotnega osveževanja in izboljševanja uteži, pred učenjem pripravimo optimizator. Uporabimo optimizator AdamW, ki je razširjena oblika

optimizatorja Adam. Optimizatorju doda L2 regularizacijo, ki zmanjšuje stopnjo prekomernega prilagajanja modela na učni množici.

Model učimo v več korakih. Na vsakem koraku generator modelu na vhod poda paket enajstih četveric vektorjev. Postopek ponavljamo, dokler generator na vhod modela ne poda vseh vektorjev iz učne podatkovne množice. Učenje modela izvedemo v treh iteracijah, pri čemer vsaka iteracija predstavlja obdelavo vseh elementov učne množice.

Pred učenjem modela nastavimo parameter, ki označuje skupno izgubo pri učenju. Na vsakem koraku nato izračunamo trenutno izgubo modela, ki jo prištejemo k skupni izgubi (angl. total loss). Na vsak 500 korak izračunamo povprečno izgubo modela. Enačba 5.1 prikazuje izračun povprečne izgube modela.

$$avg_{loss} = total_{loss}/d_u \quad (5.1)$$

Pri tem je

avg_{loss} trenutno izračunana povprečna izguba modela,
 $total_{loss}$ celotna izguba modela,
 d_u dolžina trenutno obravnavane učne množice.

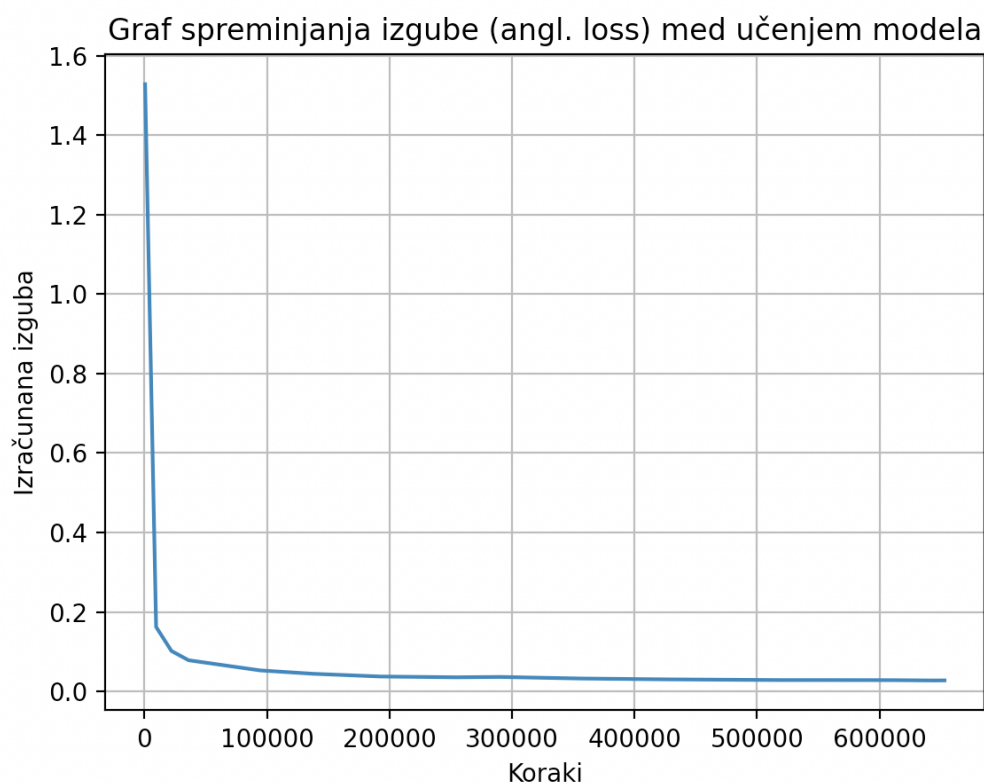
Model skupno učimo v 629.237 korakih. Na sliki 5.1 prikažemo spreminjanje izgube v odvisnosti od števila korakov pri izpopolnjevanju modela.

5.2 Model SloBERTa za črkovanje besed

V poglavju opišemo proces priprave vhodov in postopek izpopolnjevanja modela SloBERTa za črkovanje besed. Model testiramo v kombinaciji z modelom T5 za označevanje napačno zapisanih besed.

5.2.1 Priprava vhodov

Učno množico za model SloBERTa v začetni fazi gradimo po enakem postopku, kot za model T5, opisan v poglavju 5.1.1. Zgradimo množico vektor-



Slika 5.1: Graf prikazuje zniževanje izgube v odvisnosti od števila narejenih korakov pri izpopolnjevanju modela T5 za označevanje napačno zapisanih besed.

jev povedi, ki vsebujejo napačne in popačene besede, ter množico vektorjev povedi, kjer so pokvarjene besede označene z oznako `<mask>`.

Na podlagi zbirke vektorjev, ki vsebujejo napačne in popačene besede, napačno besedo obdamo z oklepaji, levo od nje pa postavimo oznako `[MASK]`. S tem kasneje modelu sporočimo, na katerih mestih naj poskuša predvidevati besedo. Iz zbirke vektorjev povedi, kjer so pokvarjene besede označene z oznako `<mask>`, napačno besedo ovijemo z oklepaji, pravilno besedo pa zapišemo pred napačno besedo. Spodnja tabela prikazuje primer zapisa povedi "Vozil sm avto .", z vsemi štirimi vektorji. V izvirni povedi popačimo besedo "avto" in jo zapišemo z besedo "auto".

Vhodni vektor	Vozil sem avto .
Vektor z napačnimi besedami	Vozil sem auto .
Vektor z maskiranimi besedami	Vozil sem <mask> .
Vhodni vektor (SloBERTa)	Vozil sem [MASK] (auto) .
Pravilni izhodni vektor	Vozil sem avto (auto) .

Podobno kot T5 model, tudi model SloBERTa pri učenju ne uporablja celotnih besed, ampak besede razdeli na žetone. Model vsako besedo deli na žetone, dokler v slovarju ne najde ustreznega zapisa žetona. Pred žeton, ki označuje začetek besede model vrine oznako `##`. Ker model lahko na mestu, ki ga označimo z oznako [MASK] predvidi samo en žeton, v naslednjem koraku oznako [MASK] podaljšamo na pričakovano dolžino popravka napačno zapisane besede. Dolžino pravilnega zapisa napačno zapisane besede pridobimo iz dolžine tokenizirane napačno zapisane besede. Če je dolžina tokenizirane napačno zapisane besede manjša od 4 prištejemo 2, v nasprotnem primeru pa 1. Pridobljena dolžina nam pri pripravi vhodov v model sporoča število maskirnih žetonov.

V vektorju, ki predstavlja pričakovan izhod modela SloBERTa, pri tokenizaciji pravilne verzije besede prav tako upoštevamo pričakovano dolžino besede. Če je ta večja od dolžine tokenizirane pravilne besede, pravilno besedo podaljšamo s posebnimi žetoni [UNK], ki so v osnovi namenjeni za označevanje neznanih znakov. Primer priprave končnega vhodnega in izhodnega vektorja v model SloBERTa prikažemo v spodnji tabeli.

Vektor z napačnimi besedami	Vozil sem auto .
Vhodni vektor	Vozil sem [MASK] (auto) .
Pravilni izhodni vektor	Vozil sem avto (auto) .
Tokeniziran vhodni vektor	Vozil sem [MASK] [MASK] [MASK] (auto) .
Tokeniziran izhodni vektor	Vozil sem avto [UNK] [UNK] (auto) .

V tabeli zgoraj lahko vidimo, da tokenizator za besedo “auto” in za besedo “avto” uporabi en žeton. Ker je beseda dolga 4 znake, dolžini besede

prištejemo 2 in oznako [MASK] nadomestimo s tremi makirnimi oznakami. V tokeniziranem izhodnem vektorju pa besedi “avto” dodamo dva žetona [UNK], katerih osnovna funkcionalnost je označevanje neznanih znakov.

5.2.2 Učenje modela SloBERTa

Pred učenjem modela pripravimo učno množico, katere pripravo smo opisali v razdelku 5.2.1. Pred pričetkom učenja modela pripravimo generator vhodov, ki modelu sproti v paketih podaja vektorje. V realni implementaciji uporabimo pakete velikosti 30.

Generator modelu na vsakem koraku poda tri vektorje: vektor identifikatorjev žetonov, izhodni vektor identifikatorjev žetonov in vektor pozornosti. Vektor identifikatorjev žetonov in izhodni vektor identifikatorjev žetonov predstavljata zaporedna števila žetonov v slovarju modela. Vektor pozornosti modelu pri učenju pove, kje se v vhodnem vektorju nahajajo vhodi, ki so pomembni pri računanju uspešnosti modela, in kateri del vhodnega vektorja predstavljajo žetoni za podaljševanje vektorja.

V spodnji tabeli prikažemo primer izgradnje vseh treh vektorjev, ki jih potrebujemo pri učenju modela. Zaradi lažjega prikaza vektorjev njihove dolžine omejimo na 8. V realni implementaciji smo dolžine vektorjev omejili na 512 žetonov.

Vhodni (Maskirani) vektor	Vozil sem [MASK] [MASK] [MASK] (auto) .
Izhodni vektor	Vozil sem avto [UNK] [UNK] (auto) .
Vhodni vektor žetonov	[CLS] Vozil sem [MASK] [MASK] [MASK] (auto) . [SEP] [PAD]
Izhodni vektor žetonov	[CLS] Vozil sem avto [UNK] [UNK] (auto) . [SEP] [PAD]
Vh. vektor identifikatorjev žetonov	103 5035 1185 105 105 105 46946 3421 45073 47105 104 2
Izh. vektor identifikatorjev žetonov	103 5035 1185 4504 102 102 46946 3421 45073 47105 104 2
Vektor pozornosti	1 1 1 1 1 1 1 1 1 1 0

Tako kot pri učenju modela T5, za namen osveževanja in izboljševanja uteži, pred učenjem pripravimo optimizator. Uporabimo optimizator AdamW. Optimizatorju pred učenjem nastavimo stopnjo učenja (angl. learning rate) na vrednost 5e-5.

Celoten proces učenja izvedemo v dveh prehodih skozi celotno učno množico. Učenje izvajamo po korakih. Na vsakem koraku obdelamo paket tridesetih trojic vektorjev.

Na vsakem koraku najprej preberemo vhodni vektor identifikatorjev žetonov, pričakovan izhodni vektor in vektor pozornosti. Nato vse vektorje prestavimo na grafični procesor in izračunamo izhod modela. Na podlagi izračunanega izhoda in pričakovanega izhoda modela izračunamo izgubo (angl. loss), ki jo uporabimo pri popravljanju uteži. Korak zaključimo s posodobitvijo zapisa trenutnega stanja učenja in korak ponovimo.

5.2.3 Testiranje modela SloBERTa

Testiranje modela SloBERTa izvedemo v dveh korakih. Najprej model testiramo na avtomatsko generirani podatkovni množici, kjer so napačno zapisane

besede označene.

Model nato testiramo skupaj z izpopolnjenim modelom T5, namenjenim označevanju napačno zapisanih besed. Model T5 uporabimo za označevanje napačnih besed, nato pa te besede skušamo popraviti z modelom SloBERTa.

Za vsako napačno besedo skušamo predvideti, iz kakšnega števila žetonov bo sestavljena. To število žetonov potrebujemo za zapis napačne besede. Pridobljenemu številu prištejemo 2, če je beseda krajša od 5 znakov, in 1, če je beseda daljša.

Pri predvidevanju besed za vsak maskirni žeton predvidimo 5 najbolj verjetnih žetonov. Med vsemi pridobljenimi žetoni nato tvorimo kombinacije. Ko se neka kombinacija od napačno zapisane besede razlikuje v največ dveh črkah in se ta kombinacija nahaja v našem slovarju (razdelek 4.3) ter v slovarju Sloleks 2.0 (razdelek 4.2), pridobljeno besedo označimo kot popravek napačne besede.

5.3 Znakovni model T5 za črkovanje besed

Znakovni model T5 omogoča, da se model pri popravljanju nekaterih napak v črkovanju osredotoča na posamezne črke, namesto da se osredotoča na celotno besedo. Osredotočanje modela na posamezno črko modelu omeji možnost parafraziranja posameznih besed. Model spodbuja zamenjavo posameznih črk, kar je koristno pri črkovanju.

5.3.1 Priprava vhodov

Učno množico za učenje modela pripravimo iz podatkovne množice opisane v poglavju 4. Pri pripravi učne množice vsako poved razdelimo na znake. Nekateri znaki, kot so ločila, števila in presledki, nadomestimo s posebnimi znaki. Med vse pridobljene znake nato vrinemo presledek. V spodnji tabeli prikažemo vse zamenjave znakov, ki jih naredimo pri pripravi vhodov.

Znak	Zamenjava
Presledki	—
Števila	#
Ločila	#
Vsak znak, ki ni v slovenski abecedi	#

V naslednjem koraku se pri obravnavi vsake povedi trikrat sprehodimo skozi vse znake v povedi. Pri prvem sprehodu skozi znake vsak znak z verjetnostjo 3% zamenjamo z naključnim znakom slovenske abecede. V drugem sprehodu vsak znak z verjetnostjo 3% izpustimo, pri čemer lahko izpustimo tudi zamenjave znakov. V zadnjem sprehodu z verjetnostjo 3% za trenutno obravnavan znak vrinemo nov znak, pri tem pa lahko vrinemo tudi znak za presledek (_).

Ker pri izpustitvi in dodajanju znakov izpuščamo tudi znake za presledek, model poleg pravilnega črkovanja učimo tudi, kdaj se neka beseda piše skupaj ali narazen.

V spodnji tabeli prikažemo primer izgradnje vhodnega in pričakovanega izhodnega vektorja pri izgradnji učne podatkovne množice.

Osnovna poved	Vozil sem avto .
Vektor znakov	V o z i l _ s e m _ a v t o #
Vektor modificiranih znakov	V o z i l _ s u m _ a v t l o #
Pričakovan izhodni vektor	V o z i l _ s e m _ a v t o #

5.3.2 Izpopolnjevanje znakovnega modela T5

Učenje pričnemo z izgradnjo generatorja, ki modelu na vhod podaja pakete štirih vektorjev: vektor identifikatorjev žetonov vhoda, vektor pozornosti kodirnika, vektor identifikatorjev žetonov izhoda in vektor pozornosti dekodirnika.

Pred pričetkom učenja pripravimo dve csv datoteki (input.csv in target.csv). V vsaki datoteki so v ločenih vrsticah zapisane povedi ločene na posamezne znake.

Znakovni model učimo z izpopolnjevanjem vnaprej naučenega T5-sl-small modela [48].

Končna podatkovna množica namenjena učenju je zgrajena iz podmnožice celotne pripravljene podatkovne množice. Učno množico pri učenju razdelimo na pakete velikosti 11. Model učimo v treh iteracijah skozi celotno učno množico.

5.3.3 Testiranje znakovnega modela T5

Testiranje izpopolnjenega znakovnega modela T5 opravimo na ločeni podatkovni množici. Pri testiranju modela posamezne vektorje, razdeljene na znake, združimo. Na ta način iz posameznih znakov pridobimo besede in v fazi testiranja preverjamo, koliko besed je model pravilno popravil. Iz dobljenih popravkov besed izračunamo natančnost, pristranskost in F_1 -oceno modela.

5.4 Model T5 za črkovanje besed

Zgradba modela T5 za črkovanje besed je podobna zgradbi modela T5 za označevanje napačno zapisanih besed. Modela ločimo z namenom večjega nadzora pri detektiranju napačnih besed. Z ločitvijo modelov se hkrati izognemo prekomernem popravljanju besed v stavkih.

5.4.1 Priprava vhodov

Vhode v model pripravimo na podoben način kot v razdelku 5.1.1. Pri tem prilagodimo postopke namenjene spreminjanju zapisa posameznih besed.

- **Naključna ločitev besede**

Pri naključni ločitvi besed namesto oznake `<splited_text>` v zbirko pravih besed zapišemo pravilno besedo. Slednji primer prikazuje spodnja tabela.

Osnovna poved	Vozil sem avto .
Pokvarjena poved	Vozil sem av to .
Pravilna poved	Vozil sem avto .

• Naključno združevanje besed

Tako kot pri postopku naključne ločitve besed pri postopku naključnega združevanja besed namesto oznake <concatenated_word> v zbirko napačnih povedi zapišemo pravilni ločeni besedi. Primer postopka naključnega združevanja besed prikažemo v spodnji tabeli.

Osnovna poved	Vozil sem avto .
Pokvarjena poved	Vozilsem avto .
Pravilna poved	Vozil sem avto .

• Ostali postopki

Postopke zamenjave besed “nagajivk”, zamenjave pogosto zamenjanih črk, zamenjave šumnikov in zamenjave naključnih znakov v besedi uporabimo na podoben način kot v razdelku 5.1.1. V primeru spremembe zapisa besede v zbirko pravih povedi ne zapišemo oznake <mask> temveč pravilno besedo.

• Končna podatkovna množica

Končna učna množica vsebuje 2.417.200 povedi. Pred učenjem modela učno množico razdelimo na učno in validacijsko, pri čemer validacijska množica predstavlja prvih 2.000 povedi, ostale pa porabimo pri učenju modela.

Testno množico gradimo na enak način kot učno množico, pri čemer testna množica vsebuje 214.000 povedi.

V spodnji tabeli prikažemo primer naključno izbranega stavka iz učne množice, primer izvirne povedi, ki jo uporabimo za pripravo vhodov, ter primer vhoda v model in njegovega pričakovanega izhoda.

Izvirna poved	In prav vsi gredo čez enak postopek obravnave .
Vhod v model	In prau usi gredo čez enak postojek obravnave .
Pričakovan izhod	In prav vsi gredo čez enak postopek obravnave .

Za razliko od pričakovanega izhoda modela T5 za označevanje napačno zapisanih besed, v pričakovanem izhodu modela zapišemo pravilne oblike besed. S tem model učimo zaznave in direktnega popravljanja napak v posameznih besedah.

5.4.2 Izpopolnjevanje modela T5

Model učimo z izpopolnjevanjem vnaprej naučenega slovenskega T5-sl-small modela [48]. Izpopolnjevanje modela poteka na enak način kot v razdelku 5.1.2. Najprej pripravimo dve csv datoteki (“train.csv” in “validate.csv”). Vsaka datoteka vsebuje dva stolpca (“input” in “target”).

V naslednjem koraku pripravimo generator, ki modelu med učenjem postopoma na vhod podaja pakete skupin štirih vektorjev. Velikost paketov, zaradi omejitve grafičnega pomnilnika, nastavimo na 11.

V spodnji tabeli prikažemo primer izgradnje vseh vektorjev, ki jih uporabimo pri učenju modela T5.

Vhodni vektor	Vozil sm avto .
Vektor žetonov vhoda	_Vo zil _sm _avto _. </s> <pad> <pad>
Vektor identifikatorjev žetonov vhoda	1518 5053 3310 670 1662 1 0 0
Vektor pozornosti kodirnika	1 1 1 1 1 1 0 0
Vektor pričakovanega izhoda	Vozil sem avto .
Vektor žetonov izhoda	_Vo zil _sem _avto _. </s> <pad> <pad>
Vektor identifikatorjev žetonov izhoda	1518 5053 309 670 1662 1 0 0
Vektor pozornosti dekodirnika	1 1 1 1 1 1 1 1

Zaradi lažjega prikaza omejimo vse vektorje na dolžino 8. V dejanski implementaciji dolžine vektorjev omejimo na dolžino 64. Kot lahko razberemo iz tabele, vektorje, ki so krajši od dolžine 8, dopolnimo z žetoni <pad>.

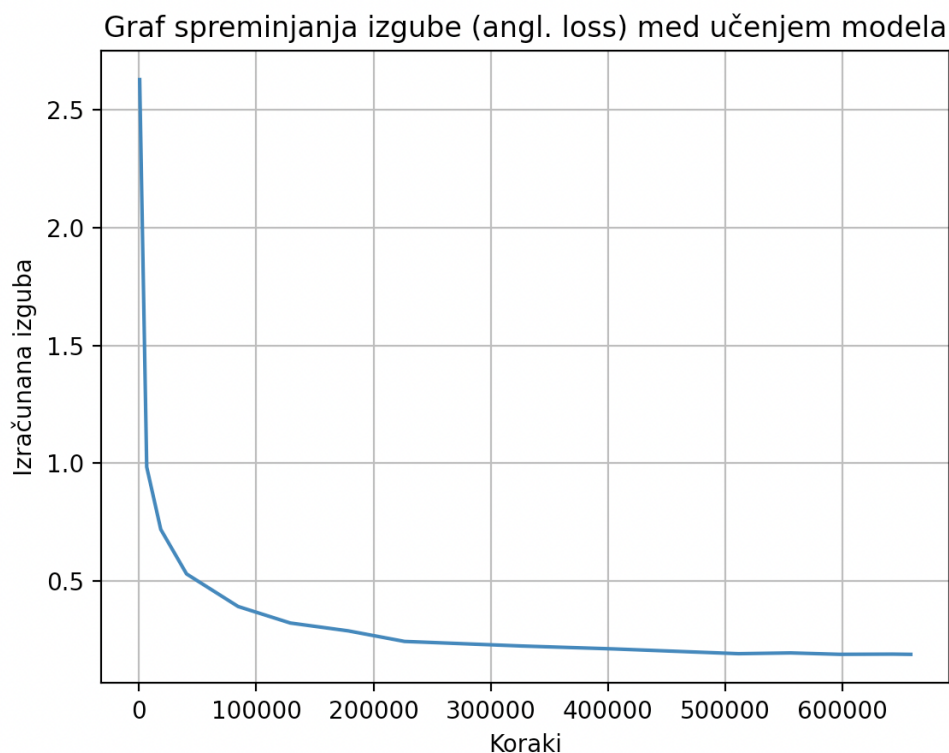
Model učimo v korakih. Pri tem nam generator na vsakem koraku pripravi paket enajstih četveric vektorjev. Na vsakem koraku računamo izgubo modela, ki jo zapišemo v skupno izgubo (angl. total loss). Na vsak 500 korak izračunamo povprečno izgubo modela.

Slika 5.2 prikazuje spreminjanje izgube med učenjem. Izgubo prikažemo v odvisnosti od korakov učenja. V primerjavi z grafom učenja modela T5 za označevanje napačno zapisanih besed lahko vidimo, da izračunana izguba pada počasneje in stagnira pri višji vrednosti. To nakazuje na večjo kompleksnost problema.

5.4.3 Testiranje modela T5

Testiranje izvedemo v dveh korakih. V prvem koraku testiramo le model, pri čemer za testno množico uporabimo množico, zgrajeno na enak način, kot učno množico.

V drugem delu model testiramo skupaj z modelom T5 namenjenim ozna-



Slika 5.2: Graf prikazuje zniževanje izgube v odvisnosti od števila narejenih korakov pri izpopolnjevanju modela T5 za črkovanje besed.

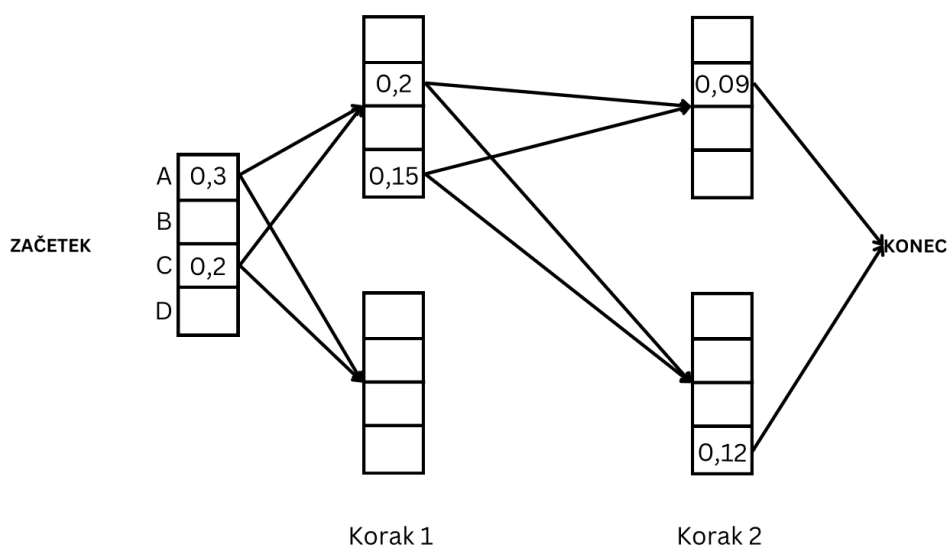
čevanju napačno zapisanih besed. Testno množico pripravimo iz korpusa Šolar, ki ga opišemo v razdelku 4.1.

Pri vsakem generiranju izhodnega vektorja modela paralelno generiramo 15 vektorjev, med katerimi izberemo 10 najbolj primernih. Generiranje večih izhodnih vektorjev nam omogoča tehnika iskanja v snopu (angl. beam search).

Dekodirnik pri generiranju vsake besede na izhod poda vektor dolžine celotnega slovarja modela, pri čemer vsakemu žetonu pripiše verjetnost. Pri požrešnem iskanju se pri izbiri žetona omejimo le na trenutno generiran vektor in vedno izberemo žeton z največjo verjetnostjo. Za razliko od požrešnega iskanja, iskanje v snopu pri računanju vsakega posameznega izhoda vzame N

najbolj verjetnih žetonov. Pri računanju verjetnosti upošteva tudi verjetnosti vseh ostalih žetonov, ki jih je izbral v prejšnjem koraku.

Primer poenostavljenega delovanja iskanja v snopu prikazuje slika 5.3. Zaradi lažje predstave se na sliki pri velikosti slovarja omejimo na 4 žetone (A, B, C in D). Prikazana tehnika izbiranja vedno izbere dva najbolj verjetna žetona. Dva najbolj verjetna končna niza sta "ABB" in "ABD". Izračun več možnih izhodnih nizov s tehniko iskanja v snopu je računsko zahtevnejši od požrešnega iskanja, omogoča pa nam hkratno generiranje večih izhodov in boljše rezultate.



Slika 5.3: Primer iskanja v snopu. Tehnika vedno izbere dva najbolj verjetna žetona, pri tem pa upošteva vse verjetnosti žetonov v prejšnjih korakih.

V množici desetih pridobljenih izhodnih vektorjev, ki jih generiramo za vsak vhod, se omejimo le na besede, ki jih model T5 za označevanje napačno zapisanih besed označi z žetoni <mask>, <concatated_word> ali <splitted_text>. V primeru, da model določeno vhodno besedo označi z žetonom <mask>, besedo iz najbolj verjetnega vektorja desetih pridobljenih izhodnih vektorjev, ki se nahaja v našem generiranem slovarju (razdelek 4.3) in v

slovarju Sloleks 2.0, uporabimo kot popravek.

Pri upoštevanju novih zamenjav besed izračunamo tudi Lavenshteinovo razdaljo med napačno besedo in predlaganim popravkom besede. Lavenshteinova razdalja predstavlja najmanjše število zamenjav, dodajanj ali pa izbrise črk, ki jih moramo narediti, da iz ene besede preidemo v drugo.

Računanje Lavenshteinove razdalje opišemo z rekurzivno enačbo 5.2.

$$lev(b1, b2) = \begin{cases} \max(|b1|, |b2|) & \min(|b1|, |b2|) = 0 \\ \min \begin{cases} lev(b1[2:], b2[2:]) + 1 & \text{(izbris)} \\ lev(b1[2:], b2) + 1 & \text{(dodajanje)} \\ lev(b1[2:], b2[2:]) + 1_{(b1[0] \neq b2[0])} & \text{(zamenjava)} \end{cases} & \text{drugače} \end{cases} \quad (5.2)$$

Pri tem je

$b1$ prva beseda,

$b2$ druga beseda,

$\max(a, b)$ funkcija, ki med števili a in b izbere večje število,

$\min(a, b)$ funkcija, ki med števili a in b izbere manjše število,

$|b1|$ dolžina besede $b1$ in

$|b2|$ dolžina besede $b2$.

Pri izbiranju besed se omejimo le na popravke besed, ki se od napačno zapisane besede razlikujejo v največ treh črkovnih modifikacijah.

V primeru, da model T5 za označevanje napačno zapisanih besed označi dve zaporedni besedi z žetonoma `<splited_text>`, besede združimo. Če model besedo označi z žetonom `<concatd_word>`, besedo zamenjamo s prvo besedo in z besedo, ki ji sledi. Obe zamenjavi naredimo le, če se vse besede nahajajo v našem generiranem slovarju (razdelek 4.2) in slovarju Sloleks 2.0.

Vsak zapis v korpusu Šolar 3.0 je zgrajen iz treh vektorjev: `src_tokens`, `tgt_tokens` in `corrections`. Vektor “`src_tokens`” predstavlja izvirno poved, ki ne vsebuje popravkov. Vektor “`tgt_tokens`” predstavlja popravljeno izvirno

poved, kjer so vključeni vsi popravki. Vektor “corrections” vsebuje vse popravke, ki so jih opravili popravljanci teksta. Vsak popravek ima označen tudi tip napake. Tipi napak so zabeleženi v vektorju “corr_types”.

Pri testiranju modela T5 upoštevamo le napake pri črkovanju, ki so v vektorju “corr_types” označene z oznako “Č”. Poleg vektorja “corr_types” vektor popravkov vsebuje tudi vektorja “idx_src”, kjer je zapisano mesto napačne besede v vektorju “src_tokens”, in vektor “idx_tgt”, kjer je zapisano mesto popravljen besede v vektorju “tgt_tokens”.

Pri testiranju modela preverjamo koliko popravkov, ki so zabeleženi v testni učni množici, je model uspel popraviti. Če model pravilno popravi besedo, ki je označena kot napačna v testni množici, to štejemo kot pravilen pozitiven popravek (TP). Če model popravi neko besedo in ta popravek ni označen kot črkovni popravek, to štejemo kot pravilen negativen popravek (TN). Če model besede ne popravi in tudi med popravki ni zabeležena kot črkovna napaka, to štejemo kot pravilen negativen popravek (TN). V kolikor model popravi besedo, ki v testni množici ni označena kot napaka ali pa model napačno besedo popravi, vendar je ne popravi pravilno, popravek štejemo kot napačen pozitiven popravek (FP).

5.5 Model T5 za popravljanje vrstnega reda besed

Model T5 za popravljanje vrstnega reda besed naučimo tako, da popravlja vrstni red znotraj stavkov. Končni naučeni model spreminja le vrstni red besed ne pa tudi njihove oblike.

5.5.1 Priprava vhodov

Učno množico za učenje modela pripravimo iz podatkovne množice opisane v poglavju 4. Pri pripravi učne množice, vsako poved razdelimo na stavke. Privzamemo, da se vsak stavek konča z ločilom ali z vejico. Vsak stavek v

povedi nato obravnavamo ločeno.

V 60 odstotkih vseh stavkov, ki vsebujejo dve besedi ali več, izberemo n naključnih besed, ki jih med seboj naključno zamenjamo. Število n določimo z uteženo naključno izbiro, kjer izbiramo med številom 2 in dolžino celotnega obravnavanega stavka. Pri uteževanju posameznih števil večjo utež pripišemo manjšim vrednostim za n. To pomeni, da funkcija večkrat zamenja 2 besedi kot vse besede v stavku.

V naslednjem koraku v posameznem stavku med seboj zamenjamo n naključno izbranih besed. Vse stavke nato združimo in pridobimo vhodni vektor za učenje modela. V vektor, ki predstavlja pričakovan izhod modela zapišemo poved, ki vsebuje pravilen vrstni red besed.

V tabeli spodaj prikažemo primer menjave vrstnega reda v povedi “Videl sem konja in odšel jahat.”. V tabeli lahko vidimo, da poved razdelimo na dva stavka, ki ju ločuje veznik “in”. Pri obravnavi vsakega stavka ločeno, stavek “Videl sem konja” pustimo takšen kot je, v stavku “odšel jahat” pa zamenjamo vrstni red.

Izvirna poved	Videl sem konja in odšel jahat.
Vhodni vektor	Videl sem konja in jahat odšel .
Pričakovan izhodni vektor	Videl sem konja in odšel jahat .

5.5.2 Izpopolnjevanje modela T5

Model učimo z izpopolnjevanjem vnaprej naučenega modela T5-sl-small. Postopek učenja je enak učenju modela za označevanje napačno zapisanih besed (razdelek 3.14) in učenju modela za popravljanje besed (razdelek 5.4).

Pred učenjem modela pripravimo generator, ki modelu med učenjem sproti na vhode podaja vektorje. Vektorje podaja v paketih velikosti 11.

Model učimo v korakih, kjer na vsakem koraku računamo celotno izgubo modela. Na vsak 500 korak izpišemo povprečno izgubo modela.

5.5.3 Testiranje modela T5

Model testiramo z učno množico, ki jo pripravimo iz podatkovne množice opisane v poglavju 4. Pri testiranju modela primerjamo poved na vhodu, pravilno poved in poved, ki jo pridobimo na izhodu modela. Vsako besedo vseh povedi obravnavamo istočasno. V primeru, da so vse besede enake, popravek štejemo kot pravilno negativen (TN). Če se beseda iz vhodne povedi v model razlikuje od besede v pravilni povedi, pri tem pa je beseda v izhodni povedi modela enaka besedi v pravilni povedi, popravek štejemo kot pravilno pozitiven (TP). Če se besedi iz izhodne in pravilne povedi ne ujemata, popravek označimo kot napačno pozitiven (FP). V kolikor se besedi iz vhodne in pravilne povedi ujemata, besedi iz izhodne povedi in pravilne povedi pa ne, popravek označimo kot napačno negativen (FN).

5.6 Model T5 namenjen pregibanju besed

Model T5 izpopolnimo tudi z namenom popravljanja pravilne oblike besed v besedilih. Ta naučeni model spreminja sklanjatve in spreganja besed ne pa tudi njihovega črkovanja in zapisa.

5.6.1 Priprava vhodov

Učno množico za učenje modela pripravimo iz podatkovne množice opisane v poglavju 4. Pri učenju modela uporabimo tudi slovar besed, opisan v poglavju 4.3.

Pri gradnji učne množice se sprehodimo skozi vsako besedo v učni množici in z verjetnostjo 20% besedo spremenimo. Spreminjamo le besede, ki niso števila in ne vsebujejo ločil.

Za vsako besedo, ki jo spremenimo, v slovarju lem poiščemo njeno lemo. Za dobljeno lemo v slovarju besed izberemo naključno besedno obliko. Izbrano besedo nato nadomestimo z novo besedno obliko.

V spodnji tabeli prikažemo primer pravilnega vektorja, ki ga uporabimo

kot vhod v model in primer vektorja, ki ima nekatere besede sklanjane napačno.

Pravilna poved	Vozil sem avto.
Napačna poved	Vozil sva avto.

5.6.2 Učenje modela T5

Model učimo z izpopolnjevanjem v naprej naučenega modela T5-sl-small. Postopek učenja je enak, kot ga opišemo v zgornjih poglavjih.

Model učimo po korakih, pri čemer na vsakem koraku generator modelu poda paket štirih vektorjev. Velikost paketa nastavimo na 11. Velikost posameznih vektorjev omejimo na dolžino 64 žetonov. Pri tem en žeton predstavlja besedo, oziroma del besede, ki se nahaja v slovarju modela.

Med učenjem modela računamo skupno izgubo učenja. Povprečno izgubo pri učenju izračunamo na vsak 500 korak.

5.6.3 Testiranje modela

Za testiranje modela uporabimo ločeno podatkovno množico. Pri testiranju modela štejemo, koliko napačnih oblik besed je model glede na testno podatkovno množico pravilno preoblikoval.

V primeru, da je model napačno obliko besede popravil pravilno, popravek označimo kot pozitivno pravilen (TP). Če je model pravilno obliko besede spremenil v napačno, popravek označimo kot napačno negativen (FN). V kolikor model pravilne oblike besede ne spremeni, popravek označimo kot pravilno negativen (TN). Če model ne popravi napačne oblike besede, popravek označimo kot napačno pozitiven (FP).

5.7 Izdelava spletnega orodja

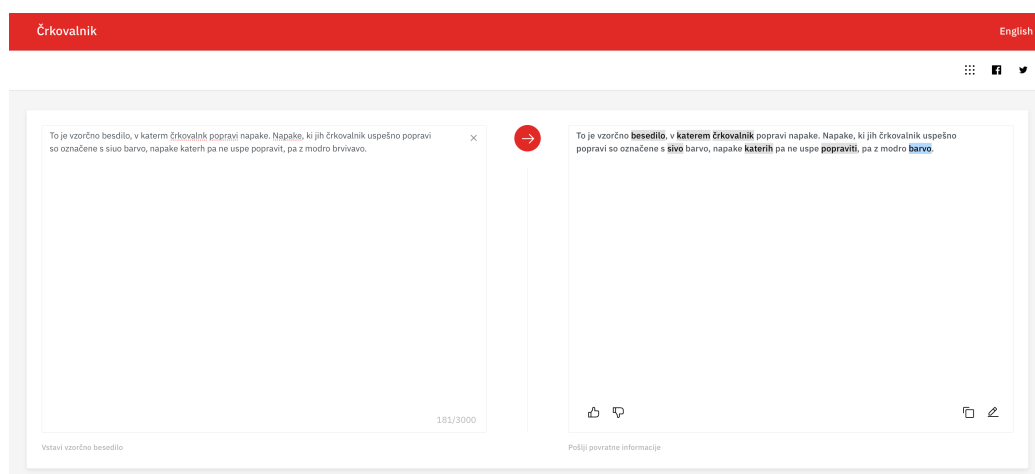
V okviru magistrske naloge pripravimo spletni vmesnik in vanj vključimo modele za popravljanje črkovanja besed. Poleg črkovalnika izdelamo spletni

vmesnik za popravljanje vrstnega reda besed v stavkih in še nekatere druge popravke, ki niso bili obravnavani v tej nalogi.

5.7.1 Črkovalnik

Osnovna naloga črkovalnika je analizirati vhodno besedilo in v njem popraviti napačno zapisane besede. Pri tem uporabimo model T5 za označevanje napačno zapisanih besed, model T5 za popravljanje napak v besedah in model T5 namenjen pregibanju besed.

Slika 5.4 prikazuje izgled črkovalnika. Črkovalnik deluje tako, da uporabnik v levo okno vnese besedilo, ki ga omejimo na dolžino 3000 znakov. S pritiskom na puščico na sredini se vnešeno besedilo posreduje v zaledni sistem, kjer se obdela. Uporabniku nato vrnemo besedilo z označenimi popravki pri črkovanju in skladnji besed.



Slika 5.4: Slika prikazuje izgled črkovalnika. Besede, ki jih je črkovalnik uspel popraviti, so označene s sivo barvo, besede, za katere črkovalnik meni, da so napačne, vendar ni prepričan, kako jih popraviti, pa z modro barvo.

V zalednem sistemu se vsako večje besedilo in besedilo, ki vsebuje znake za novo vrstico, obdela postopoma. Besedilo najprej razdelimo na odseke, ločene z znakom za novo vrstico. V kolikor je posamezni odsek daljši od 300

znakov, v odseku poiščemo zadnjo piko in glede nanjo odsek ločimo na dva manjša dela. Vsak odsek nato obravnavamo ločeno.

Vsak odsek besedila najprej peljemo skozi model T5 za označevanje napačno zapisanih besed. Mesta, ki jih model na izhodu označi z oznakami `<mask>`, `<splited_text>` ali pa `<concated_word>`, si zabeležimo. Vhodni odsek besedila peljemo še skozi T5 model za popravljanje napačno zapisanih besed.

Kot opišemo v poglavju 5.4.3, pri generiranju izhoda modela T5 za črkovanje besed uporabimo iskanje besed v snopu, pri čemer vedno izbiramo med 15 najbolj verjetnimi besedami. Izmed 15 pridobljenih izhodnih vektorjev izberemo 10 najbolj verjetnih.

Besede, ki jih model T5 za označevanje napačno napisanih besed označi z žetonom `<mask>`, nadomestimo z besedo iz najbolj verjetnega izhodnega vektorja, ki se nahaja v našem generiranem slovarju (razdelek 4.3) in v slovarju SloLeks 2.0. Vse ostale besede, ki se nahajajo v obeh slovarjih dodamo v listo mogočih zamenjav in jih uporabniku prikažemo, če na uporabniškem vmesniku pritisne na popravljeno besedo. V primeru, da model določeno besedo označi z žetonom `<concated_word>` ali pa dve zaporedni besedi označi z žetonom `<splited_text>`, besede razdružimo oziroma združimo.

Pri izbiranju besed se omejimo le na popravke, ki se od napačno zapisane besede razlikujejo v največ treh črkovnih popravkih. Črkovne popravke izračunamo s pomočjo Lavenshteinove razdalje. Ostale besede, ki se nahajajo v obeh slovarjih, dodamo v možnosti. Možnosti za izbiro popravkov besed so urejene glede na število črkovnih popravkov. Pri vrhu so besede, ki se od napačno zapisane besede razlikujejo le v eni črki, najnižje pa besede, ki se od napačno zapisane besede razlikujejo v vseh črkah.

Poglavje 6

Evalvacija rešitve

V poglavju predstavimo rezultate testiranja naučenih modelov. Pri vsakem modelu navedemo, na kako veliki učni množici smo ga testirali in izračunamo točnost, pristranskost in F_1 -oceno modela. Pri vsakem modelu navedemo nekaj testnih primerov in predstavimo, na katerih primerih delujejo dobro in na katerih ne. Pri vsakem modelu opišemo področja, kjer bi lahko posamezen model izboljšali.

Najprej predstavimo rezultate testiranja izpopolnjenega modela T5 za označevanje napačno zapisanih besed, nato predstavimo rezultate izpopolnjenega modela SloBERTa za popravljanje napak pri zapisu besed, sledi predstavitev rezultatov izpopolnjenega modela T5 za popravljanje napak pri zapisu besed. Nadaljnje predstavimo rezultate znakovnega modela T5 namenjenega črkovanju besed, modela T5 za popravljanje vrstnega reda besed in modela T5 za pregibanje besed.

Pri testiranju vseh modelov pri vsakem izhodu modela izračunamo število besed v kategorijah TP, TN, FP in FN.

S pomočjo pridobljenih vrednosti TP, FP, TN in FN izračunamo točnost, priklic in F_1 -ocena posameznih modelov.

Točnost (angl. precision) nam pove, koliko besed je posamezni model popravil pravilno. Izračunamo jo z enačbo

$$tocnost = \frac{TP}{TP + FP} \quad (6.1)$$

Poleg točnosti izračunamo priklic (angl. recall), ki nam pove, kolikšen delež besed je model popravil pravilno v primerjavi z deležem vseh napačnih besed. Priklic izračunamo z enačbo

$$priklic = \frac{TP}{TP + FN} \quad (6.2)$$

V primeru, da model prekomerno popravlja besede, se mu bo povečalo število FP primerov in zmanjšalo število FN primerov. To bo vodilo do boljšega priklica modela, ki pa bo imel manjšo točnost. V primeru, da bo model popravljaj besede, za katere bo res prepričan, da jih zna popraviti, se mu bo zmanjšal delež FP primerov, kar bo povečalo točnost modela, povečalo pa se bo število FN primerov, kar bo zmanjšalo priklic.

Model deluje najbolje, ko uspemo obdržati visoko točnost modela, ne da bi s tem močno zmanjšali priklic. F_1 -ocena predstavlja razmerje med točnostjo in priklicom modela. F_1 -oceno izračunamo z enačbo

$$F_1 - ocena = \frac{2 * točnost * priklic}{točnost + priklic} \quad (6.3)$$

6.1 Evalvacija modela T5 za označevanje napačno zapisanih besed

Model testiramo z lastno testno podatkovno množico. Testna množica vsebuje 2132 besed. Od napačnih besed jih model pravilno označi 714 za napačne, 3 besede označi za napačne, vendar so pravilne, 1373 pravilnih besed ne označi in 42 besed, ki so napačne, ne označi kot napačne. Rezultate testiranja prikazuje spodnja tabela.

TP	714
FP	3
TN	1373
FN	42

Iz pridobljenih rezultatov lahko izračunamo točnost, priklic in F_1 -oceno modela. Rezultate prikažemo v spodnji tabeli.

točnost	0,9958
priklic	0,9444
F_1-ocena	0,9694

Pri testiranju modela opazimo, da model v več primerih določene besede ne označi kot napačno črkovane, čeprav je ta zapisana nepravilno. Ko neko besedo označi za napačno, je zapis besede v večini primerov nepravilen. Če želimo z istim modelom zmanjšati število napak, ko model neke napačne besede ne označi kot napačne, lahko pri računanju izhoda modela uporabimo iskanje v snopu. S to tehniko lahko naenkrat pridobimo več izhodnih vektorjev, ki jih nato združimo. Iskanja v snopu smo podrobneje opisali v razdelku 5.4.3 in bi zmanjšalo število napak tipa FN. S tem pa bi zmanjšali tudi točnost modela, saj bi model lahko več pravih besed označil kot napačne in bi se s tem povečalo število napak tipa FP. Seveda želimo točnost modela ohranjati visoko, sicer bi uporabnika zavajali.

Pri analizi rezultatov opazimo, da je večina neoznačenih nepravilnih besed takšnih, ki so same zase pravilne, v kontekstu povedi pa ne. Primer takšne povedi, je uporaba besede "sem" namesto besede "jem", prikazana spodaj. V tabeli lahko vidimo, da model pravilno označi besedo "auto" kot napačno, besede "jem" pa ne.

vhod v model	Vozil jem auto .
rezultat	Vozil jem <mask> .

Model v večini primerov zna odpraviti napake, kjer pri napačnem črkovanju besede pridobimo novo pravilno besedo, ki pa je glede na kontekst povedi

napačna. Takšen primer je prikazan spodaj, kjer je beseda "otok" napačno črkovana kot "otrok". V danem primeru, naš model besedo pravilno označi kot napačno.

vhod v model	Na otrok smo odšli z avtomobilom .
rezultat	Na <mask> smo odšli z avtomobilom .

6.2 Evalvacija modela SloBERTa za črkovanje besed

Izpopolnjen model SloBERTa sprva evalviramo na testni množici, kjer imamo že označene napačne besede. Zaradi tega se pri prvi evalvaciji osredotočimo le na število besed, ki jih je model popravil pravilno, in na število besed, ki jih model ni uspel pravilno popraviti. Vseh napačnih besed v učni množici je 2106. Od teh jih je model uspel uspešno popraviti 1643, kar predstavlja približno 78% vseh napačnih besed.

V drugem delu model evalviramo na testni množici, ki jo zgradimo iz podatkovne množice Šolar 3.0. Pri testiranju najprej z modelom T5 za označevanje napačno zapisanih besed označimo vse napačno zapisane besede in pripravimo vhode za izpopolnjen model SloBERTa. Za pravilno pozitiven (TP) primer tako štejemo napačno črkovano besedo, ki jo je model T5 označil in model SloBERTa pravilno popravil. Podobno velja za napake tipa FP, TN in FN.

Testna množica, zgrajena iz podatkovne množice Šolar 3.0, vsebuje 28.979 besed. Spodaj prikažemo pridobljene rezultate, kjer lahko vidimo, da model popravi veliko besed, ki so v vhodni povedi zapisane pravilno.

TP	97
FP	149
TN	61
FN	28666

6.3 Evalvacija znakovnega modela T5 za črkovanje besed

Model testiramo z lastno podatkovno množico. Testna množica vsebuje 7.921 znakov, od tega je 3.027 znakov napačnih. Od napačnih znakov naš model pravilno popravi 1459 znakov, 1568 nepravilnih znakov pa ne popravi. 480 znakov model popravi, kljub temu da so zapisani pravilno in 4.414 znakov ne popravlja. Rezultate testiranja prikazuje tabela spodaj.

TP	1459
FP	480
TN	4414
FN	1568

Iz pridobljenih rezultatov lahko izračunamo točnost, F_1 -oceno in pristranskost modela. Rezultate prikazuje tabela spodaj.

točnost	0,7525
priklic	0,4773
F_1-ocena	0,5841

Pri analizi rezultatov opazimo, da glede na ostale črkovalnike, znakovni model T5 deluje najslabše. Menimo, da je vzrok za to drugačna oblika pripravljanja vhodov v model. Modela ne učimo s celotnimi besedami in pri tem ne izkoriščamo prednaučene besedne predstave modela, ki si jo ustvari z razbijanjem besed na žetone. Kljub temu model v dosti primerih pravilno popravi črkovne napake. Tak primer prikažemo v tabeli spodaj.

Vhodna poved	A tisčo # kar še osebej vlivna upanje na kpkovostno sezono # je trenersko zaupanje # .
Pravilna poved	A tisto # kar še posebej živi upanje na kako vlivno sezono # je trenersko zaupanje #
Rezultat modela	A tisto # kar še osebej vlivna upanje na kako vostno sezono # je trenersko zaupanje #

6.4 Evalvacija modela T5 za črkovanje besed

V prvem delu model testiramo z lastno podatkovno množico. Testna množica vsebuje 2106 besed. Izmed napačno zapisanih besed model T5 pravilno popravi 885 besed, 1 besede ne popravi pravilno, 1173 besed, ki so zapisane pravilno, ne popravi, prav tako ne popravi 47 napačno zapisanih besed. Rezultate testiranja modela prikazuje tabela spodaj.

TP	885
FP	1
TN	1173
FN	47

Iz pridobljenih rezultatov nato izračunamo točnost, priklic in F_1 -oceno modela. Rezultate prikazuje spodnja tabela.

točnost	0,9988
priklic	0,9496
F_1-ocena	0,9736

Iz pridobljenih rezultatov lahko vidimo, da ima model veliko točnost. V primeru, da besedo popravi, obstaja velika verjetnost, da jo popravi pravilno. Kljub veliki točnosti ima model nekaj težav pri zaznavi in popravljanju nekaterih besed, ki so v besedilu zapisane napačno. To se odraža na povečanem številu napak tipa FN in manjšem priklicu modela.

Kljub temu, da model včasih ne popravi vseh napak, pogosto popravi napake, kjer je v povedi napačno črkovana beseda sama zase pravilna. Takšen primer prikazuje tabela spodaj, kjer model pravilno popravi besedo "jem" z besedo "je", čeprav je beseda "jem" sama po sebi pravilna.

vhod v model	Vozil jem auto .
rezultat	Vozil je avto .

V drugem delu model testiramo s testno podatkovno množico, ki jo zgradimo iz korpusa Šolar 3.0. Testiranje opravimo skupaj z modelom T5 za

označevanje napačno zapisanih besed. Pri testiranju smo pozorni le na napake pri črkovanju besed. Postopek testiranja je podrobneje opisan v poglavju 5.4.3. Pri testiranju izberemo naključen izsek celotne testne množice. Pri analizi napak ugotovimo, da je v podatkovni množici veliko primerov, kjer so besede črkovane napačno a niso popravljene. Zaradi tega vse napake, ki jih pri testiranju zaznamo, tudi ročno pregledamo. Primer vhodne in ročno popravljene povedi, v kateri je beseda “mislili” napačno črkovana, prikazuje tabela spodaj. V tabeli lahko vidimo, da je naš model napačno črkovano besedo popravil pravilno.

Vhodna poved	Mislim , da so ves obup in razmišljanja po vseh grozotah moralili izliti na papir .
Popravljen poved	Mislim , da so ves obup in razmišljanja po vseh grozotah moralili izliti na papir .
Rezultat modela	Mislim , da so ves obup in razmišljanja po vseh grozotah morali izliti na papir .

Testna množica, ki jo zgradimo iz korpusa Šolar 3.0, vsebuje 16.597 besed. Od teh je 176 besed črkovanih napačno. Model pravilno popravi 155 besed, ostalih 21 besed pa pusti nepopravljenih. 33 besed popravi, čeprav so črkovane pravilno, 16388 pravilno črkovanih besed pa ne popravlja. Spodnja tabela prikazuje rezultate modela.

TP	155
FP	33
FN	21
TN	16388

Iz pridobljenih rezultatov lahko izračunamo točnost, priklic in F_1 -oceno modela kot prikažemo v tabeli

točnost	0,8245
priklic	0,8807
F_1-ocena	0,8516

Pri drugem testiranju modela opazimo, da ima testna množica zgrajena iz korpusa Šolar 3.0 veliko več pravilno zapisanih besed. To je pričakovano, saj je prva testna množica avtomatsko generirana in vsebuje več napak pri črkovanju besed.

Opazimo tudi, da se poveča število besed, ki jih model popravi napačno. V tabeli 6.4 prikažemo primer povedi, ki vsebuje napačno črkovano ime “Terezijas”. Naš model pravilno zazna, da je črkovanje imena napačno. Pri popravljanju model besedo popravi z besedo “Terezija”, moral pa bi jo popraviti z besedo “Teiresijas”. Po analizi številnih primerov ugotovimo, da je vzrok za tovrstne napake, da model takšnih imen v času učenja ni videl ter da se številna lastna imena ne nahajajo v našem slovarju besed (razdelek 4.3) oziroma v slovarju Sloleks 2.0.

Vhodna poved	Pri pokopu svojega brata jo ovirata tudi Terezijas in Hajmon , katera tudi zastopata državno pravo , v katerem ni prostora za ljubezen .
Popravljen poved	Pri pokopu njenega brata jo ovirata tudi Teiresijas in Hajmon , ki tudi zastopata državno pravo , v katerem ni prostora za ljubezen .
Rezultat modela	Pri pokopu svojega brata jo ovirata tudi Terezija in Hajmon , katera tudi zastopata državno pravo , v katerem ni prostora za ljubezen .

6.5 Evalvacija modela T5 za popravljanje vrstnega reda besed

Model testiramo z lastno podatkovno množico. Testna množica vsebuje 2125 besed. 475 besedam spremenimo vrstni red v stavkih. Model uspešno popravi vrstni red 367 besed, 108 napačno postavljenim besedam ne spreminja vrstnega reda, 86 besedam popravi vrstni red, čeprav stojijo na pravem mestu in 1564 besedam, ki stojijo na pravem mestu v stavku, vrstnega reda ne

spremeni. Rezultate testiranja prikažemo v tabeli spodaj.

TP	367
FP	86
TN	1564
FN	108

Iz pridobljenih rezultatov lahko izračunamo točnost, priklic in F_1 -oceno modela. Rezultate prikazuje tabela spodaj.

točnost	0,8102
priklic	0,7726
F_1-ocena	0,7910

Ena izmed težav ocenjevanja točnosti modela za popravljanje vrstnega reda besed so povedi, ki imajo lahko več pravih vrstnih redov. Pri teh povedih model lahko popravi vrstni red besed pravilno, vendar vrstnega reda ne popravi na enak način, kot je to pričakovano iz vhodne povedi. Ta težava je prisotna predvsem v povedih, ki imajo več besed pomešanih. V teh primerih se vse besede označijo kot napačne, čeprav model vrstni red popravi pravilno. Primer takšne povedi je prikazan v tabeli spodaj. V povedi lahko vidimo, da je vrstni red besed, ki jih vrne model pravilen, enako je pravilen vrstni red besed v pravilni povedi.

Vhodna poved	lokalne so danes oblasti tudi objavile seznam z imeni vseh 19 žrtev .
Pravilna poved	Danes so lokalne oblasti tudi objavile seznam z imeni vseh 19 žrtev .
Rezultat modela	Lokalne oblasti so danes tudi objavile seznam z imeni vseh 19 žrtev .

Veliko je tudi primerov, kjer model vrstni red besed popravi bolj od vrstnega reda besed v izvorni povedi. Primer takšne povedi prikazuje spodnja tabela.

Vhodna poved	drugače bo letos , veva ker , kako skupaj videti vse je in kako poteka turnir .
Pravilna poved	Letos bo drugače , ker veva , kako je videti vse skupaj in kako turnir poteka .
Rezultat modela	Letos bo drugače , ker veva , kako je vse skupaj videti in kako poteka turnir .

6.6 Evalvacija modela T5 za pregibanje besed

Model testiramo z lastno podatkovno množico. Testna množica vsebuje 2125 besed. 312 besed v testni podatkovni množici je napačno pregibanih. Od teh jih naš model uspešno popravi 276. Ostalih 36 ne popravi in jih pusti nespremenjene. 9 besed model popravi, čeprav so pregibane pravilno. 1804 besede, ki so zapisane pravilno, model ne popravi. Rezultate modela prikažemo v tabeli spodaj.

TP	276
FP	9
TN	1804
FN	36

Iz pridobljenih rezultatov lahko izračunamo točnost, priklic in F_1 -oceno modela. Rezultate prikazuje spodnja tabela.

točnost	0,9684
priklic	0,8846
F_1-ocena	0,9246

Pri analizi rezultatov ugotovimo, da ima model veliko točnost, največ napak pa naredi pri nepopravljanju napačno pregibanih besed. V nekaterih primerih do tega pride zaradi zgradbe testne in učne množice, saj pri simuliranju napačnega pregibanja besed določeno besedo zamenjamo z drugo obliko te besede, ki je lahko v trenutnem kontekstu tudi pravilna. Primer takšne povedi prikazuje tabela spodaj. Vidimo lahko, da smo pri gradnji

testne množice besedo “bo” zamenjali z besedo “je”, ki je prav tako pravilna in je naš model ni popravil.

Vhodna poved	S pomoč teh točk je avtomobil izpisoval na cesti različne grafike .
Pravilna poved	S pomočjo teh točk bo avtomobil izpisoval na cesto različne grafike .
Rezultat modela	S pomočjo teh točk je avtomobil izpisoval na cesti različne grafike .

6.7 Primerjava rezultatov naših modelov

V magistrskem delu smo skupno izpopolnili 3 različne modele za črkovanje besed. Najslabše rezultate pridobimo z znakovnim modelom T5. Iz tega sklepamo, da je pri uporabi modelov T5 podatkovno množico bolje zgraditi iz celotnih besed namesto posameznih znakov.

Boljše rezultate pridobimo z modeloma SloBERTa in T5. Oba modela testiramo v kombinaciji z modelom T5 za označevanje napačno zapisanih besed. Za testiranje uporabimo podatkovno množico Šolar 3.0. Izračunano točnost, priklic in F_1 -oceno obeh modelov prikažemo v tabeli spodaj.

	Model T5 za črkovanje besed	Model SloBERTa za črkovanje besed
točnost	0,8245	0,3943
priklic	0,8807	0,6139
F_1-ocena	0,8516	0,4802

Iz tabele je razvidno, da bistveno boljše rezultate dosežemo z modelom T5. Model ima občasno težave pri popravljanju nekaterih besed, ki so črkovane pravilno, vendar so v kontekstu povedi napačne. Takšnih besed občasno ne popravi. To težavo bi lahko rešili s posodobitvijo učne množice, v kateri bi v nekaterih primerih določene besede “popačili” tako, da bi jih zamenjali z besedami, ki bi se od izvirnih besed razlikovale le v manjšem številu črk,

zamenjave besed pa bi bile še vedno pravilno črkovane. Menimo, da bi število omenjenih napak zmanjšali tudi z daljšim časom izpopolnjevanja modela in uporabo večje učne množice.

Pri evalvaciji modela T5 za popravljanje vrstnega reda besed opazimo, da ima model težave, kadar je potrebno poleg vrstnega reda besed spremeniti tudi oblike besed. Vzrok za to je neuporaba takšnih primerov pri izgradnji učne množice. Težavo bi lahko odpravili s posodobitvijo učne množice, pri kateri bi poleg menjave vrstnega reda besed v stavku zamenjali tudi nekatere oblike besed. Model bi moral nato popraviti oblike in vrstni red besed.

Pri evalvaciji modela T5 za pregibanje besed opazimo, da model naredi največ napak pri zaznavi napačno pregibanih besed. V teh primerih napačno pregibanih besed ne spreminja. Menimo, da bi to lahko izboljšali z uporabo večje učne množice in daljšim časom izpopolnjevanja modela.

Izmed vseh modelov za posamezne podprobleme, najboljše rezultate pridobimo z modelom T5 za zaznavanje napačno zapisanih besed, modelom T5 za črkovanje besed, modelom T5 za popravljanje vrstnega reda besed in modelom T5 za pregibanje besed. Omenjene modele zaradi dobrih rezultatov uporabimo v spletni aplikaciji.

6.8 Primerjava z drugimi jeziki

Sahu in sod. v [44] primerjajo rezultate pridobljene s testiranjem petih različnih slovničnih popravljalnikov v angleščini. Najboljše rezultate dosežejo s popravljalnikoma Grammarly [52] in LanguageTool [46]. Vse popravljalnike testirajo na 500 povedih, ki jih popačijo s petimi različnimi vrstami napak: napake v strukturi povedi, napake v črkovanju besed, sintaktične napake, napake pri uporabi končnih ločil in semantične napake. Za vsako vrsto napake izračunajo točnost posameznega modela pri zaznavi in popravljanju napak.

V tabeli spodaj prikažemo primerjavo točnosti v črkovanju besed slovničnih popravljalnikov Grammarly, LanguageTool in našega izpopolnjenega modela T5 za črkovanje besed, ki deluje v kombinaciji z modelom T5 za ozna-

čevanje napačno zapisanih besed. Primerjava ni povsem objektivna, saj naš model deluje na realnih napakah iz korpusa Šolar, druga dva popravljalnika pa na umetno generiranih napakah. V tabeli lahko vidimo, da je točnost našega modela T5 slabša, vendar še vedno primerljiva s točnostjo obeh popravljalnikov. Kljub slabši točnosti menimo, da je naš model T5 primeren za uporabo v praktičnih aplikacijah. Omenimo, da popravljanje črkovanja besed predstavlja večji problem v slovenskem kot v angleškem jeziku, saj poznamo v slovenskem jeziku več posebnosti, ki se pojavljajo pri rabi besed v stavkih. Omenimo tudi, da naš model T5 za črkovanje besed testiramo na množici šolskih spisov, kjer veliko povedi vsebuje večje število napak, kar pri črkovanju modelu otežuje delo.

model	točnost
Model T5 za črkovanje besed	0,83
Grammarly (angleščina)	0,99
LanguageTool (angleščina)	0,95

Rezultate modela T5 za pregibanje besed primerjamo tudi z rezultati popravljalnikov Grammarly in LanguageTool pri popravljanju sintaktičnih napak (ang. Syntax error). V tabeli spodaj prikažemo primerjavo točnosti obeh angleških modelov in našega modela T5 za pregibanje besed. Omenimo, da tukaj naš model T5 za pregibanje besed testiramo na avtomatsko generirani množici, kjer so tipi napak enaki tipom napak, s katerimi model naučimo, kar posledično vodi do boljših rezultatov. Kljub temu točnost 96,8% nakazuje na uporabnost modela tudi v praktičnih aplikacijah.

model	točnost
Model T5 za pregibanje besed	0,97
Grammarly (angleščina)	0,69
Language Tool (angleščina)	0,44

Pri modelu T5 za popravljanje vrstnega reda besed dosežemo točnost 81%, vendar kljub visoki točnosti menimo, da bomo za praktično uporabnost, model morali še izboljšati.

Za model SloBERTa za črkovanje besed in znakovni model T5 za črkovanje besed menimo, da še nista primerna za uporabo v praktičnih aplikacijah. Za dejansko rabo ju bomo v nadaljnjem delu združili s klasičnim črkovalnikom na osnovi leksikona besed Sloleks.

Poglavje 7

Zaključek

V magistrskem delu analiziramo, izpopolnimo in prilagodimo velike jezikovne modele za popravljanje slovničnih napak v slovenščini. Osredotočimo se na tri vrste napak: napake pri črkovanju besed, napake pri pregibanju besed in na pravilen vrstni red besed v povedih. Skupaj izpopolnimo in prilagodimo šest jezikovnih modelov. En model je namenjen zaznavanju napačno zapisanih besed, trije modeli so namenjeni popravljanju črkovanja besed, eden je namenjen popravljanju pregibanja besed in eden popravljanju vrstnega reda besed v povedih.

Pri evalviranju modela T5 za označevanje napačno zapisanih besed ugotovimo, da ima model občasno težave z označevanjem besed, ki so zapisane pravilno, vendar so v kontekstu povedi napačne. Omenjeno težavo bi lahko odpravili z uporabo večjega deleža takšnih besed pri izgradnji učne množice.

Pri analizi rezultatov ugotovimo, da najboljše rezultate pridobimo z izpopolnjevanjem modela T5-sl-small. Najboljšo primerjavo med modelom T5 in modelom SloBERTa vidimo pri primerjavi obeh črkovalnikov, kjer boljše rezultate pridobimo z izpopolnjenim modelom T5.

Pri pripravi izpopolnjevanja modela namenjenega pregibanju besed na manjši množici preizkusimo izpopolniti modela SloBERTa in T5. Ugotovimo, da z modelom T5 pridobimo občutno boljše rezultate.

Pri izpopolnjevanju znakovnega modela T5 za popravljanje črkovanja

ugotovimo, da z njim dobimo slabše rezultate. Iz tega sklepamo, da je bolje, da pri modelih T5 operiramo s celotnimi besedami, saj v tem primeru izkoristimo pridobljeno znanje o morfologiji besed, ki jo model T5 gradi skozi osnovni proces učenja.

Model za označevanje napačnih besed, izpopolnjen model T5 za črkovanje besed in izpopolnjen model za pregibanje besed uporabimo v preprosti spletni aplikaciji. V ločeni spletni aplikaciji objavimo model za popravljanje vrstnega reda besed.

Tekom izdelave magistrske naloge smo ugotovili, da ima največji vpliv na uspešnost in delovanje jezikovnih modelov izbira jezikovnega modela in priprava učne množice. Zaradi tega smo pri izdelavi naloge največ časa posvetili izdelavi dobre učne množice, kjer smo skušali simulirati čim več kvalitetnih in realnih jezikovnih napak. Pri testiranju in izpopolnjevanju modelov se je pogosto izkazalo, da je kvalitetnejša podatkovna množica pomembnejša od njene velikosti.

Večjo uspešnost pri delovanju posameznih jezikovnih modelov dosežemo tudi z razdelitvijo problemov na manjše in bolj obvladljive. To izboljša preglednost nad delovanjem modelov in omogoča njihovo lažjo integracijo v spletno aplikacijo.

Modele uporabljene v spletni aplikaciji bomo izboljševali še naprej. Pri tem se bomo osredotočili predvsem na izboljšanje kakovosti uporabljenih učnih množic ter modele izpopolnili z uporabo večje in kakovostnejše učne množice.

V prihodnje bomo razvili tudi model za popravljanje vrstnega reda besed, kjer bomo poleg vrstnega reda poizkusili popravljati tudi obliko besed. Problem vrstnega reda besed v stavkih bomo poizkusili razširiti na problem vrstnega reda stavkov v povedi.

Opomba: Vsa koda uporabljena pri izdelavi magistrske naloge je dostopna na url naslovu: https://github.com/mb0484/magistrska_naloga.

Literatura

- [1] Jay Alammar. *The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning)*. <http://jalammar.github.io/illustrated-bert/>. Obiskano: Julij 2023. 2018.
- [2] Jay Alammar. *The Illustrated Transformer*. <https://jalammar.github.io/illustrated-transformer/>. Obiskano: Julij 2023. 2018.
- [3] Jay Alammar. *Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models With Attention)*. <https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/>. Obiskano: Julij 2023. 2018.
- [4] Špela Arhar Holdt, Tadeja Rozman, Mojca Stritar Kučuk, Simon Krek, Irena Krapš Vodopivec, Marko Stabej, Eva Pori, Teja Goli, Polona Lavrič, Cyprian Laskowski, Polonca Kocjančič, Bojan Klemenc, Luka Krtnik in Iztok Kosem. *Developmental corpus Šolar 3.0*. Slovenian language resource repository CLARIN.SI. 2022. URL: <http://hdl.handle.net/11356/1589>.
- [5] Arthur Arnx. *First neural network for beginners explained (with code)*. <https://towardsdatascience.com/first-neural-network-for-beginners-explained-with-code-4cfd37e06eaf>. Obiskano: Julij 2023. 2019.
- [6] Jason Brownlee. *A Gentle Introduction to Positional Encoding in Transformer Models*. <https://machinelearningmastery.com/a-gentle->

- introduction-to-positional-encoding-in-transformer-models-part-1/. Obiskano: Julij 2023. 2023.
- [7] Jason Brownlee. *Encoder-Decoder Recurrent Neural Network Models for Neural Machine Translation*. <https://machinelearningmastery.com/encoder-decoder-recurrent-neural-network-models-neural-machine-translation/>. Obiskano: Julij 2023. 2019.
- [8] Miguel Romero Calvo. *Dissecting BERT Part 1: The Encoder*. <https://medium.com/dissecting-bert/dissecting-bert-part-1-d3c3d495cdb3>. Obiskano: Julij 2023. 2018.
- [9] *Common Crawl*. <https://commoncrawl.org/>. Obiskano: Avgust 2023.
- [10] Jacob Devlin, Ming-Wei Chang, Kenton Lee in Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". V: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Jun. 2019, str. 4171–4186. DOI: 10.18653/v1/N19-1423.
- [11] Kaja Dobrovoljc, Simon Krek, Peter Holozan, Tomaž Erjavec, Miro Romih, Špela Arhar Holdt, Jaka Čibej, Luka Krsnik in Marko Robnik-Šikonja. *Morphological lexicon Sloleks 2.0*. Slovenian language resource repository CLARIN.SI. 2019. URL: <http://hdl.handle.net/11356/1230>.
- [12] Ketan Doshi. *Foundations of NLP Explained Visually: Beam Search, How It Works*. <https://towardsdatascience.com/foundations-of-nlp-explained-visually-beam-search-how-it-works-1586b9849a24>. Obiskano: Avgust 2023. 2021.
- [13] Rachel Draelos. *The Transformer: Attention Is All You Need*. <https://glassboxmedicine.com/2019/08/15/the-transformer-attention-is-all-you-need/>. Obiskano: Julij 2023. 2019.

- [14] Tomaž Erjavec, Darja Fišer, Nikola Ljubešić, Marko Ferme, Mladen Borovič, Borko Boškovič, Milan Ojsteršek in Goran Hrovat. *Corpus of academic Slovene KAS 1.0*. Slovenian language resource repository CLARIN.SI. 2019. URL: <http://hdl.handle.net/11356/1244>.
- [15] Tomaž Erjavec, Nikola Ljubešić in Darja Fišer. “Korpus slovenskih spletnih uporabniških vsebin Janes”. V: *Viri, orodja in metode za analizo spletne slovenščine*, edited by Darja Fišer, Znanstvena založba Filozofske fakultete, Ljubljana (2018), str. 16–43.
- [16] Tomaž Erjavec, Nikola Ljubešić in Nataša Logar. “The Slwac corpus of the Slovene web”. V: *Informatica* 39.1 (2015).
- [17] Giuliano Giacaglia. *How Transformers Work*. Obiskano: Julij 2023. 2019. URL: <https://towardsdatascience.com/transformers-141e32e69591>.
- [18] Roman Grundkiewicz, Marcin Junczys-Dowmunt in Kenneth Heafield. “Neural grammatical error correction systems with unsupervised pre-training on synthetic data”. V: *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*. 2019, str. 252–263.
- [19] Dan Hendrycks in Kevin Gimpel. “Gaussian error linear units (gelus)”. V: *arXiv preprint arXiv:1606.08415* (2016).
- [20] Sepp Hochreiter in Jürgen Schmidhuber. “Long short-term memory”. V: *Neural computation* 9.8 (1997), str. 1735–1780.
- [21] Rani Horev. *BERT Explained: State of the art language model for NLP*. <https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270>. Obiskano: Julij 2023. 2018.
- [22] Oleksii Hrinchuk, Mariya Popova in Boris Ginsburg. “Correction of Automatic Speech Recognition with Transformer Sequence-To-Sequence Model”. V: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, str. 7074–7078. DOI: 10.1109/ICASSP40776.2020.9053051.

- [23] Masahiro Kaneko, Masato Mita, Shun Kiyono, Jun Suzuki in Kentaro Inui. “Encoder-Decoder Models Can Benefit from Pre-trained Masked Language Models in Grammatical Error Correction”. V: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Jul. 2020, str. 4248–4254. DOI: 10.18653/v1/2020.acl-main.391.
- [24] Chetna Khanna. *Byte-Pair Encoding: Subword-based tokenization algorithm*. <https://towardsdatascience.com/byte-pair-encoding-subword-based-tokenization-algorithm-77828a70bee0>. Obiskano: Julij 2023. 2021.
- [25] Chetna Khanna. *WordPiece: Subword-based tokenization algorithm*. <https://towardsdatascience.com/wordpiece-subword-based-tokenization-algorithm-1fbd14394ed7>. Obiskano: Julij 2023. 2021.
- [26] Kiprono Elijah Koech. *Softmax Activation Function — How It Actually Works*. <https://towardsdatascience.com/softmax-activation-function-how-it-actually-works-d292d335bd78>. Obiskano: Julij 2023. 2020.
- [27] Dan Kondratyuk. “Cross-Lingual Lemmatization and Morphology Tagging with Two-Stage Multilingual BERT Fine-Tuning”. V: *Proceedings of the 16th Workshop on Computational Research in Phonetics, Phonology, and Morphology*. Avg. 2019, str. 12–18. DOI: 10.18653/v1/W19-4203.
- [28] Simon Krek, Špela Arhar Holdt, Tomaž Erjavec, Jaka Čibej, Andraz Repar, Polona Gantar, Nikola Ljubešić, Iztok Kosem in Kaja Dobrovoljc. “Gigafida 2.0: The Reference Corpus of Written Standard Slovene”. English. V: *Proceedings of the Twelfth Language Resources and Evaluation Conference*. Maj 2020, str. 3340–3345. URL: <https://aclanthology.org/2020.lrec-1.409>.

-
- [29] Myunghoon Lee, Hyeonho Shin, Dabin Lee in Sung-Pil Choi. “Korean grammatical error correction based on transformer with copying mechanisms and grammatical noise implantation methods”. V: *Sensors* 21.8 (2021), str. 2658.
- [30] Deng Liang, Chen Zheng, Lei Guo, Xin Cui, Xiuzhang Xiong, Hengqiao Rong in Jinpeng Dong. “BERT enhanced neural machine translation and sequence tagging model for Chinese grammatical error diagnosis”. V: *Proceedings of the 6th Workshop on Natural Language Processing Techniques for Educational Applications*. 2020, str. 57–66.
- [31] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer in Veselin Stoyanov. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. 2019. arXiv: 1907.11692 [cs.CL].
- [32] Minh-Thang Luong, Hieu Pham in Christopher D Manning. “Effective approaches to attention-based neural machine translation”. V: *arXiv preprint arXiv:1508.04025* (2015).
- [33] Maxime. *What is a Transformer?* <https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04>. Obiskano: Julij 2023. 2019.
- [34] Chris McCormick. *BERT Word Embeddings Tutorial*. <https://mccormickml.com/2019/05/14/BERT-word-embeddings-tutorial/>. Obiskano: Julij 2023. 2019.
- [35] Prakhar Mishra. *Understanding T5 Model : Text to Text Transfer Transformer Model*. <https://towardsdatascience.com/understanding-t5-model-text-to-text-transfer-transformer-model-69ce4c165023>. Obiskano: Julij 2023. 2020.
- [36] Rok Mokorar. *Obvladovanje slovničnih napak v šolskih pisnih izdelkih z metodami za obdelavo naravnega jezika*. Diplomsko delo. Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, 2023.

- [37] Ethan Nam. *Understanding the Levenshtein Distance Equation for Beginners*. <https://medium.com/@ethannam/understanding-the-levenshtein-distance-equation-for-beginners-c4285a5604f0>. Obiskano: Avgust 2023. 2019.
- [38] Michael A Nielsen. *Neural networks and deep learning*. Zv. 25. Determination press, San Francisco, CA, USA, 2015.
- [39] Chris Olah in Shan Carter. “Attention and augmented recurrent neural networks”. V: *Distill* 1.9 (2016), e1.
- [40] Andrej Pančur, Tomaž Erjavec, Mihael Ojsteršek, Mojca Šorn in Neja Blaj Hribar. *Slovenian parliamentary corpus (1990-2018) siParl 2.0*. Slovenian language resource repository CLARIN.SI. 2020. URL: <http://hdl.handle.net/11356/1300>.
- [41] Timotej Petrič. *Predlogi jezikovnih popravkov v slovenščini z modelom SloBERTa*. Diplomsko delo. Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, 2022.
- [42] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu in sod. “Exploring the limits of transfer learning with a unified text-to-text transformer.” V: *J. Mach. Learn. Res.* 21.140 (2020), str. 1–67.
- [43] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li in Peter J. Liu. “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”. V: *J. Mach. Learn. Res.* 21.1 (jan. 2020). ISSN: 1532-4435.
- [44] Subham Sahu, Yogesh Kumar Vishwakarma, Jeevanlal Kori in Jitendra Singh Thakur. “Evaluating performance of different grammar checking tools”. V: *International Journal* 9.2 (2020).
- [45] Mike Schuster in Kaisuke Nakajima. “Japanese and Korean voice search”. V: *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2012, str. 5149–5152. DOI: 10.1109/ICASSP.2012.6289079.

-
- [46] *Spell and Grammar Checker. LanguageTool*. <https://languagetool.org>. Obiskano: September 2023.
- [47] Sik-Ho Tsang. *Review — Pre-LN Transformer: On Layer Normalization in the Transformer Architecture*. <https://sh-tsang.medium.com/review-pre-ln-transformer-on-layer-normalization-in-the-transformer-architecture-b6c91a89e9ab>. Obiskano: Julij 2023. 2022.
- [48] Matej Ulčar in Marko Robnik-Šikonja. “Sequence-to-sequence pretraining for a less-resourced Slovenian language”. V: *Frontiers in Artificial Intelligence* 6 (2023), str. 932519. DOI: 10.3389/frai.2023.932519. URL: <https://www.frontiersin.org/articles/10.3389/frai.2023.932519>.
- [49] Matej Ulčar in Marko Robnik-Šikonja. *Slovenian RoBERTa contextual embeddings model: SloBERTa 2.0*. Slovenian language resource repository CLARIN.SI. 2021. URL: <http://hdl.handle.net/11356/1397>.
- [50] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser in Illia Polosukhin. “Attention is all you need”. V: *Advances in neural information processing systems*. 2017, str. 5998–6008.
- [51] Sean Welleck, Ilia Kulikov, Stephen Roller, Emily Dinan, Kyunghyun Cho in Jason Weston. “Neural Text Generation With Unlikelihood Training”. V: *International Conference on Learning Representations*. 2020. URL: <https://openreview.net/forum?id=SJeYe0NtvH>.
- [52] *Write your best with Grammarly. Grammarly*. <https://grammarly.com/>. Obiskano: September 2023.