Transformer neuronske mreže za obradu prirodnog jezika

Petrović, Darko

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University North / Sveučilište Sjever**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:122:471182

SVEUČILIŠTE SJEVER

SVEUČILIŠNI CENTAR VARAŽDIN

DIPLOMSKI RAD br. 076-MMD-2022

Transformer neuronske mreže za obradu prirodnog jezika

Darko Petrović

Varaždin, rujan 2022. godine

SVEUČILIŠTE SJEVER

SVEUČILIŠNI CENTAR VARAŽDIN

Diplomski sveučilišni studij Multimedija

DIPLOMSKI RAD br. 076-MMD-2022

Transformer neuronske mreže za obradu prirodnog jezika

Student:

Darko Petrović, mat.br. 2229/336

Mentor:

izv. prof. dr. sc. Emil Dumić

Varaždin, rujan 2022. godine

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR–42000 Varaždin

Prijava diplomskog rada

Definiranje teme diplomskog rada i povjerenstva

ODJEL Odjel za multimediju

STUDIJ diplomski sveučilišni studij Multimedija

PRISTUPNIK Petrović Darko | JMBAG 0336020838

DATUM 14.9.2022. | KOLEGIJ Multimedijska videotehnologija

NASLOV RADA Transformer neuronske mreže za obradu prirodnog jezika

NASLOV RADA NA ENGL. JEZIKU Transformer neural networks for natural language processing

MENTOR Emil Dumić | ZVANJE izv.prof.dr.sc.

ČLANOVI POVJERENSTVA

1. doc. art. dr. sc. Mario Periša - predsjednik
2. doc. art. dr. sc. Robert Geček - član
3. izv. prof. dr. sc. Emil Dumić - mentor
4. doc. dr. sc. Andrija Bernik - zamjenski član
5.

Zadatak diplomskog rada

BROJ 076-MMD-2022

OPIS

U ovom radu bit će ispitano nekoliko tipova transformer neuronskih mreža (Neural Networks, NN) za obradu prirodnog jezika.
U radu će biti opisano područje obrade prirodnog jezika (Natural Language Processing, NLP) koja predstavlja vezu između računala i ljudskih jezika. Bit će objašnjeni i povezani pojmovi, shvaćanje prirodnog jezika (Natural Language Understanding) i stvaranje prirodnog jezika (Natural Language Generation, NLG). Najčešće se za obradu prirodnog jezika koriste različite metode strojnog učenja. Posebno će se opisati različiti tipovi dubokih neuronskih mreža koji mogu biti iskorišteni i za obradu prirodnog jezika: RNN (Recurrent Neural Network), ograđena RNN (gated RNN), LSTM (Long Short Term Memory) te transformeri. Zatim će se opisati modeli neuronskog strojnog prevođenja bazirani na pozornosti (attention-based neural machine translation). Potom će se detaljnije opisati transformer tip neuronske mreže, također baziran na pozornosti, te nekoliko tipova takvih istreniranih neuronskih mreža: BERT (Bidirectional Encoder Representations from Transformers), RoBERTa (Robustly optimized BERT pretraining approach), ALBERT (A Lite BERT) i XLNet. Usporedbu različitih metoda za strojni prijevod jednog jezika u drugi može se napraviti koristeći BLEU ocjenu (bilingual evaluation understudy).
U praktičnom dijelu rada bit će prikazano nekoliko tipova transformera i njihovi rezultati za područje obrade prirodnog jezika. Za usporedbu će biti korištena neka od postojećih baza poput WMT (Workshop on Statistical Machine Translation) baze za prijevod, koristeći neki par jezika. Usporedba različitih tipova neuronskih mreža će biti dana koristeći npr. BLEU ocjenu.

ZADATAK URUČEN 14.9.2022. | POTPIS MENTORA Emil Dumić

SVEUČILIŠTE SJEVER

Sažetak

U ovom radu bit će ispitano nekoliko tipova transformer neuronskih mreža (Neural Networks, NN) za obradu prirodnog jezika. U radu će biti opisano područje obrade prirodnog jezika (Natural Language Processing, NLP) koja predstavlja vezu između računala i ljudskih jezika. Bit će objašnjeni i povezani pojmovi, shvaćanje prirodnog jezika (Natural Language Understanding) i stvaranje prirodnog jezika (Natural Language Generation, NLG). Najčešće se za obradu prirodnog jezika koriste različite metode strojnog učenja. Posebno će se opisati različiti tipovi dubokih neuronskih mreža koji mogu biti iskorišteni i za obradu prirodnog jezika: RNN (Recurrent Neural Network), ograđena RNN (gated RNN), LSTM (Long Short Term Memory) te transformeri. Zatim će se opisati modeli neuronskog strojnog prevođenja bazirani na pozornosti (attention-based neural machine translation). Potom će se detaljnije opisati transformer tip neuronske mreže, također baziran na pozornosti, te nekoliko tipova takvih istreniranih neuronskih mreža: BERT (Bidirectional Encoder Representations from Transformers), RoBERTa (Robustly optimized BERT pretraining approach), ALBERT (A Lite BERT) i XLNet. Usporedbu različitih metoda za strojni prijevod jednog jezika u drugi možemo napraviti koristeći BLEU ocjenu (Bilingual evaluation understudy). U praktičnom dijelu rada bit će prikazano nekoliko tipova transformera i njihovi rezultati za područje obrade prirodnog jezika. Za usporedbu će biti korištena neka od postojećih baza poput skup podataka knjiga Opus (eng. Opus books dataset) za prijevod, koristeći engleski i francuski jezik. Usporedba različitih tipova neuronskih mreža će biti dana koristeći BLEU ocjenu.

Ključne riječi u završnom radu: **Neuronska Mreža, Duboko Učenje, Transformer Neuronska Mreža, BERT, BLEU**

Abstract

In this paper, several types of transformer neural networks (NN) for natural language processing will be examined. The paper will describe the field of natural language processing (NLP), which represents the connection between computers and human languages. Related concepts, Natural Language Understanding and Natural Language Generation (NLG) will be explained. Different machine learning methods are most often used for natural language processing. Different types of deep neural networks that can be used for natural language processing will be described in particular: RNN (Recurrent Neural Network), fenced RNN (gated RNN), LSTM (Long Short Term Memory) and transformers. Then attention-based neural machine translation models will be described. Then the transformer type of neural network, also based on attention, and several types of such trained neural networks will be described in more detail: BERT (Bidirectional Encoder Representations from Transformers), RoBERTa (Robustly optimized BERT pretraining approach), ALBERT (A Lite BERT) and XLNet. A comparison of different methods for machine translation from one language to another can be made using the BLEU evaluation (Bilingual evaluation understudy). In the practical part of the work, several types of transformers and their results for the field of natural language processing will be presented. For comparison, some of the existing databases, such as the Opus books dataset, will be used for translation, using English and French. A comparison of different types of neural networks will be given using the BLEU score.

Keywords: **Neural Network, Deep Learning, Transformer Neural Network, BERT, BLEU.**

Popis korištenih kratica

AI Umjetna inteligencija (eng. Artificial intelligence)
NLP Obrada prirodnog jezika (eng. Natural Language Processing)
NLU Shvaćanje prirodnog jezika (eng. Natural Language Understanding)
NLG Stvaranje prirodnog jezika (eng. Natural Language Generation)
DNN Duboke neuronske mreže (eng. Deep neural networks)
RNN Povratna neuronska mreža (eng. Recurrent Neural Network)
LSTM Dugotrajna kratka memorija (eng. Long Short Term Memory)
FFN Mreža za prosljeđivanje (eng. Feedforward network)
Post-LN Normalizacija nakon sloja (eng. Post-layer normalization)
BERT Dvosmjerni prikaz kodera od transformera (eng. Bidirectional Encoder Representations from Transformers)
MLM Maskirani jezični modeli (eng. Masked language models)
NSP Predviđanje sljedeće rečenice (eng. Next sentence prediction)
BPE Kodiranje para bajtova (eng. Byte-Pair Encoding)
BLEU Dvojezična evaluacija strojnih prijevoda (eng. Bilingual evaluation understudy)
GLEU Opće razumijevanje jezika (eng. General Language Understanding)
MCC Matthewsov koeficijent korelacije (eng. Matthews correlation coefficient)
CUDA Računanje opće namjene na grafičkim procesnim jedinicama, razvijen od tvrtke Nvidia (eng. Compute Unified Device Architecture)

Sadržaj

1. Uvod

Jezik je bit ljudske komunikacije. Većina komunikacije se danas odvija preko interneta i s time nastaju novi problemi za računala kao što su: prepoznavanje govora, govor u tekst, tekst u govor, prepoznavanje emocija u tekstu, prepoznavanje značenja riječi u kontekstu, itd. Grana računalne znanosti pod nazivom obrada prirodnog jezika je nastala kako bi se ti problemi riješili. Danas se svaki dan susrećemo sa obradom prirodnog jezik u sljedećim oblicima: web tražilicama, email-ovima, web stranicama, društvenim mrežama, online prevoditeljima i mnogim drugim svakodnevnim funkcijama.

U prosincu 2017., objavljen je članak *Attention Is All You Need*, koji su napisali članovi Google Brain-a i Google Research-a. U ovom radu je prvi puta opisana transformer neuronska mreža. Transformer je nadmašio sve postojeće najsuvremenije NLP modele. Transformer se trenirao brže nego prijašnje arhitekture zahvaljujući njegovoj sposobnosti da se dijelovi računanja izvode paralelno. Transformer je bio bolji u brzini i kvaliteti.

Ubrzo nakon objavljivanja članka *Attention Is All You Need* su se počeli pojavljivati razni modeli temeljeni na transformeru. Cilj tih modela je bio koristiti samo dijelove transformera ili dalje graditi na njemu.

2. Obrada prirodnog jezika

Obrada prirodnog jezika (NLP - eng. Natural Language Processing) je grana računalne znanosti, još specifičnije je grana umjetne inteligencije (AI – eng. Artificial intelligence). Bavi se sposobnošću računala da razumije pročitani tekst ili izgovorene riječi na isti način kao i čovjek.

NLP se može podijeliti u dvije podskupine: shvaćanje prirodnog jezika (NLU – eng. Natural Language Understanding) i stvaranje prirodnog jezika (NLG – eng. Natural Language Generation).

2.1. Što je jezik?

Jezik je sustav konvencionalnih govornih, ručnih (potpisanih) ili pisanih simbola pomoću kojih se izražavaju ljudska bića, kao članovi društvene skupine i sudionici njezine kulture. Funkcije jezika uključuju komunikaciju, izražavanje identiteta, igru, maštovito izražavanje i emocionalno oslobađanje [1].

Osoba koja se susretne s novom riječi može u rječniku lako pronaći značenje te riječi. Zbog toga se nove riječi stalno dodaju u rječnike.

2.2. Prirodni jezik i računalo

Računalo je stroj koji je ograničen na matematička pravila. Nema mogućnost kompleksne interpretacije i razumijevanja poput ljudi, ali može brzo izvršavati kompleksne matematičke izračune. Za sada računala su jako uspješna u klasifikaciji (klasifikacija teksta u neku kategoriju) i prevođenju teksta (prevođenje teksta iz jednog u drugi jezik) [2].

Kako bi stroj mogao baratati prirodnim jezikom, prvo ga mora pretvoriti u matematički radni okvir. Najčešće tehnike za postizanje tog prevođenja:

- Tokenizacija, Izvor riječi, Lematizacija (eng. tokenization, stemming, lemmatization)
- N-Grams
- TF-IDF
- One-Hot Encodings
- Duboke neuronske mreže (DNN - eng. Deep neural networks)

Duboke neuronske mreže će detaljnije biti objašnjene u ovom radu.

2.3. Shvaćanje prirodnog jezika (NLU)

Slično kao i NLP, NLU koristi algoritme za pretvorbu ljudskog jezika u strukturiranu ontologiju. Tada algoritmi umjetne inteligencije otkrivaju stvari kao što su namjera, vrijeme, lokacije i osjećaji [3].

NLU je podskup NLP-a, koji koristi sintaktičku i semantičku analizu teksta i govora za određivanje značenja rečenice. Sintaksa se odnosi na gramatičku strukturu rečenice, dok semantika aludira na njezino namjeravano značenje. NLU također uspostavlja strukturu podataka koja specificira odnose između riječi i izraza. Dok ljudi to prirodno čine u razgovoru, kombinacija ovih analiza potrebna je da bi stroj razumio namjeravano značenje različitih tekstova.

Glavna razlika od NLP-a je u tome što NLU ne samo da razumije riječi, nego i pokušava protumačiti značenje baveći se uobičajenim ljudskim pogreškama poput pogrešnog izgovora.

2.4. Stvaranje prirodnog jezika (NLG)

NLG još je jedan podskup NLP-a. Dok se razumijevanje prirodnog jezika usredotočuje na računalno razumijevanje čitanja, generiranje prirodnog jezika omogućuje računalima pisanje prirodnog jezika. NLG je proces stvaranja tekstualnog odgovora na temelju nekog unosa podataka. Ovaj se tekst također može pretvoriti u govorni format putem usluga pretvaranja teksta u govor. NLG također obuhvaća mogućnosti sažimanja teksta koje generiraju sažetke iz ulaznih dokumenata uz zadržavanje integriteta informacija.

Ustanovilo se kako stroj može komunicirati ideje iz podataka u izvanrednoj mjeri i točnosti. To može učiniti na posebno artikuliran način. Kada stroj automatizira rutinske zadatke analize i komunikacije, povećava se produktivnost i zaposlenici se mogu usredotočiti na aktivnosti veće vrijednosti [4].

NLG aplikacije moraju uzeti u obzir jezična pravila temeljena na morfologiji, leksikonima, sintaksi i semantici kako bi donijeli izbor o tome kako pravilno formulirati tekst/odgovore.

2.5. NLP zadaci (eng. NLP tasks)

Ljudski jezik sadrži puno nejasnoća koje otežavaju izradu programa koji može točno odrediti značenje tekstualnih ili glasovnih podataka. Homonimi, homofoni, sarkazam, idiomi, metafore, gramatičke iznimke, uporabne iznimke – ovo su samo neke od nepravilnosti ljudskog jezika za koje su ljudima potrebne godine da nauče. Ovisno o potrebi se određene iznimke trebaju uzeti u obzir pri izradi aplikacije za prepoznavanje i razumijevanje jezika.

Neki od NLP zadataka razrađuju tekstualne i glasovne podatke kako bi olakšali računalu da shvati sa čime raspolaže. Neki od ovih zadataka su sljedeći:

- Prepoznavanje govora (eng. Speech recognition) – je zadatak pouzdanog pretvaranja glasovnih podataka u tekstualne podatke. Ono što prepoznavanje govora čini posebno izazovnim je način na koji ljudi razgovaraju - brzo, nerazgovjetno, koriste različite naglaske i intonacije i često koriste netočnu gramatiku. Prepoznavanje govora prepisuje usmene podatke u tok riječi. Neuronske mreže i skriveni Markovi modeli (eng. Hidden Markov models) koriste se za smanjenje stope pogreške prepoznavanja govora. Glavni izazov je nedostatak segmentacije u usmenim dokumentima [5].
- Označavanje dijela govora (eng. Part of speech tagging) – još se naziva gramatičkim označavanjem, proces je određivanja dijela govora određene riječi ili dijela teksta na temelju njegove upotrebe i konteksta.
- Višeznačnost smisla riječi (eng. Word sense disambiguation) – je odabir značenja riječi (u slučaju da riječ ima više značenja) kroz proces semantičke analize. Određuje se značenje riječi koje ima najviše smisla u danom kontekstu.
- Prepoznavanje imenovanog entiteta (eng. Named entity recognition) – je entitetsko vađenje, identifikacija i kategorizacija [5]. Identificiraju se riječi ili fraze te ih se grupira po određenim kategorijama.
- Rezolucija koreferenci (eng. Co-reference resolution) – cilj je identificirati odnose li se i kada dvije riječi na isti entitet. Najčešći primjer je određivanje osobe ili predmeta na koji se odnosi određena zamjenica, ali može uključivati i identifikaciju metafore ili idioma u tekstu.
- Analiza osjećaja (eng. Sentiment analysis) – pokušava izvući subjektivne osobine (stavove, emocije, sarkazam, zbunjenost, sumnju) iz teksta.
- Rezimeacija (eng. Summarization) – NLP algoritmi mogu se koristiti za stvaranje skraćene verzije članka, dokumenta, broja unosa itd., s uključenim glavnim točkama i ključnim idejama. Postoje dva opća pristupa: apstraktivna i ekstraktivna rezimeacija. U prvom slučaju, NLP model stvara potpuno novi sažetak u smislu fraza i rečenica koje se koriste u analiziranom tekstu. U drugom slučaju, model izdvaja fraze i rečenice iz postojećeg teksta i grupira ih u sažetak [5].

3. Duboko učenje (eng. Deep learning)

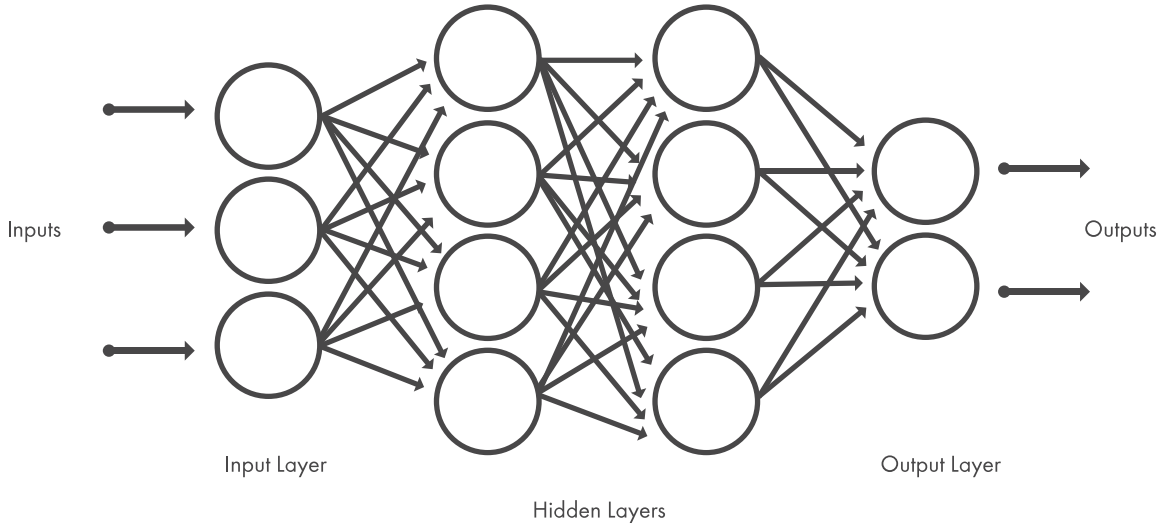
U ovom poglavlju će se oopisat što je duboko učenje i metode dubokog učenja. Specifično, opisat će se sljedeće neuronske mreže: Ponavljajuća neuronska mreža (RNN - eng. Recurrent Neural Network), ograđena RNN (eng. gated RNN) i LSTM (LSTM - eng. Long Short Term Memory).

Jedan od načina treniranja dubokih neuronskih mreža kod NLP-a je nenadzirano pohlepno slojevito predtreniranje (Unsupervised Greedy Layer-Wise Pretraining), koje je korisno kad imamo mnogo neoznačenih riječi, iza kojeg može slijediti nadzirano učenje s npr. mnogo manjim označenim skupom podataka. Takav predtrening uči hijerarhiju značajki. Pohlepni nenadzirani predtrening u slojevima temelji se na obučavanju svakog sloja s algoritmom učenja bez nadzora, uzimajući značajke proizvedene na prethodnoj razini kao ulaz za sljedeću razinu. Tada je jednostavno ekstrahirati značajke ili kao ulaz za standardni prediktor nadziranog strojnog učenja (kao što su strojevi potpornih vektora ili uvjetno nasumično polje) ili kao inicijalizacija za dubinsku nadziranu neuronsku mrežu. Na primjer, svaka iteracija nenadziranog učenja značajki dodaje jedan sloj težine dubokoj neuronskoj mreži. Konačno, skup slojeva s naučenim težinama mogao bi se složiti kako bi se inicijalizirao dubinski nadzirani prediktor, kao što je klasifikator neuronske mreže, ili duboki generativni model, kao što je duboki Boltzmannov stroj (Deep Boltzmann Machine) [6].

Većina metoda dubokog učenja koriste neuronske mreže, iz tog razloga se metode dubokog učenja nazivaju duboke neuronske mreže.

3.1. Duboke neuronske mreže (DNN - eng. Deep neural networks)

Termin „duboko“ najčešće opisuje broj skrivenih slojeva u neuronskoj mreži. Modeli dubokog učenja su trenirani koristeći velike skupove podataka i arhitekture neuronske mreže. Te neuronske mreže uče značajke direktno iz podataka bez potrebe za ručnom ekstrakcijom značajki.



Slika 3.1 Model neuronske mreže s više skrivenih slojeva

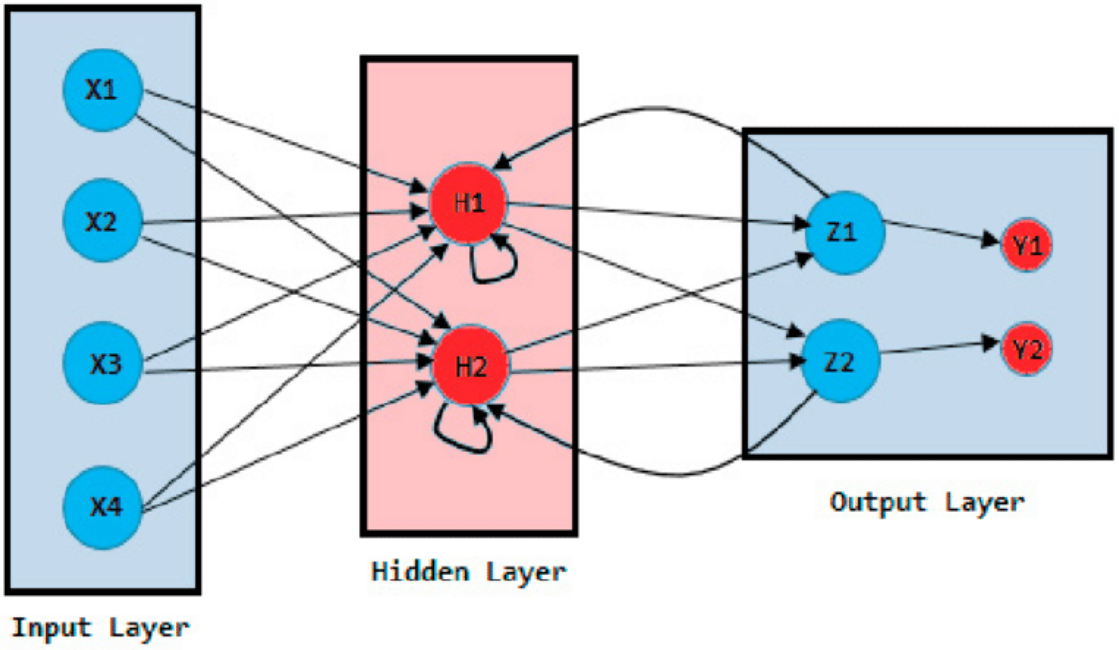
(izvor: https://www.mathworks.com/discovery/deep-learning.html#whyitmatters)

Jedna od najpopularnijih neuronskih mreža za duboko učenje kod obrade slika je konvolucijska neuronska mreža (CNN – eng. Convolutional neural networks).

3.2. Povratna neuronska mreža (RNN)

Povratne neuronske mreže su snažan i robustan tip neuronske mreže i pripadaju najperspektivnijim algoritmima koji se koriste jer jedini imaju internu memoriju. Kao i mnogi drugi algoritmi dubokog učenja, ponavljajuće neuronske mreže relativno su stare. Nastale su osamdesetih godina, ali tek posljednjih godina se je vidio njihov pravi potencijal. Povećanje računalne snage zajedno s ogromnim količinama podataka s kojima se sada može raditi i izumom dugoročnog kratkoročnog pamćenja (LSTM) 1990-ih, doista je dovelo RNN-ove u prvi plan. Zbog svoje interne memorije, RNN-i se mogu sjetiti važnih stvari o unosu koji su primili, što im omogućuje da budu vrlo precizne u predviđanju onoga što slijedi. Zbog toga su preferirani algoritam za sekvencijalne podatke kao što su govor, tekst, financijski podaci, audio, video, vrijeme i još mnogo toga. Ponavljajuće neuronske mreže mogu formirati mnogo dublje razumijevanje slijeda i njegovog konteksta u usporedbi s drugim algoritmima [7].

Kod RNN-a „recurrent“ znači da izlaz u trenutnom vremenskom koraku postaje ulaz u sljedeći vremenski korak. Na svakom elementu slijeda model ne uzima u obzir samo trenutni unos, već i ono što pamti o prethodnim elementima [8]. To se može vidjeti na slici 3.2.



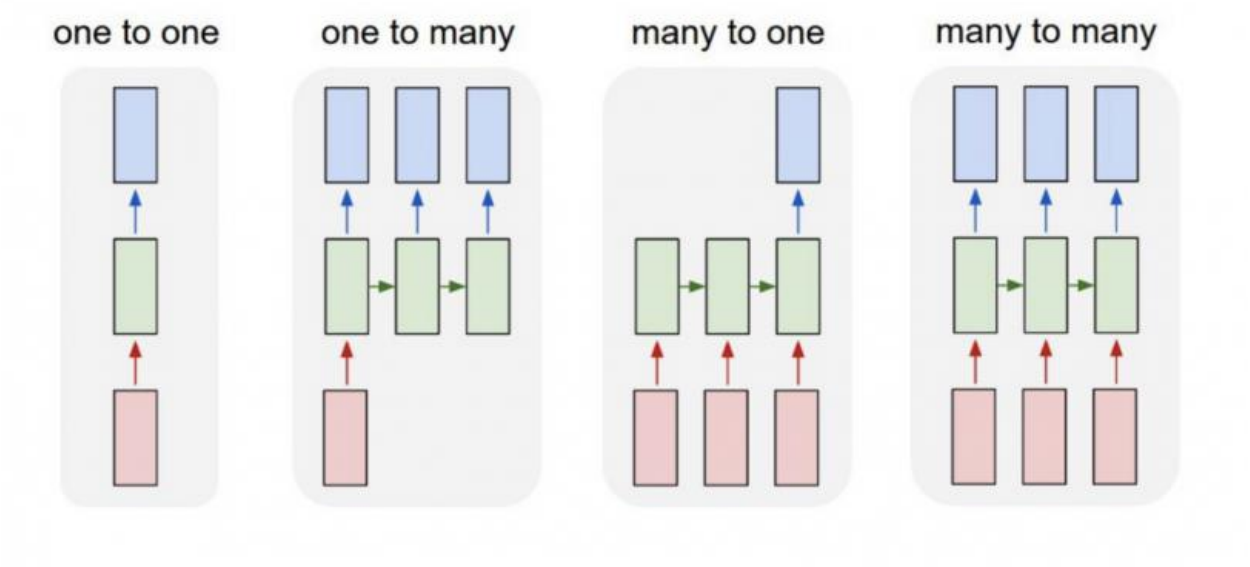
Slika 3.2 RNN arhitektura

(izvor: https://www.researchgate.net/figure/Basic-Architecture-of-Recurrent-Neural-Network-22-23_fig2_325668211)

Ponavljajuće neuronske mreže dodaju neposrednu prošlost sadašnjosti. Stoga RNN ima dva ulaza: sadašnjost i nedavnu prošlost. To je važno jer slijed podataka sadrži ključne informacije o tome što slijedi, zbog čega RNN može raditi stvari koje drugi algoritmi ne mogu. Unaprijedna Neuronska mreža dodjeljuje, kao i svi drugi algoritmi dubokog učenja, težinski faktor svojim ulazima, a zatim proizvodi izlaz. RNN primjenjuju težinski faktor na trenutni i na prethodni unos. RNN će prilagoditi težinski faktor za gradijentni spust i unatražno rasprostiranje kroz vrijeme (BPTT – eng. backpropagation through time) [8].

„Feed-forward“ neuronske mreže mapiraju jedan unos na jedan izlaz, dok RNN-ovi se mogu podijeliti na sljedeće tipove ovisno o omjeru unosa na izlaz:

- Jedan prema jedan (eng. one to one)
- Jedan prema više (eng. one to many)
- Više prema jedan (eng. many to one)
- Više prema više (eng. many to many)



Slika 3.3 Podjela RNN-a ovisno o ulazu i izlazu

(izvor: https://builtin.com/data-science/recurrent-neural-networks-and-lstm)

Jedan koncept kod RNN-a je „backpropagation through time“ (BPTT).

BPTT je primjena algoritma za trening unatražnog rasprostiranja na ponavljajuću neuronsku mrežu primijenjenu na sekvenciranje podataka. Cilj algoritma obuke za pozadinsku-propagaciju je izmijeniti težinu neuronske mreže kako bi se smanjila pogreška mrežnih izlaza u usporedbi s nekim očekivanim izlazom kao odgovor na odgovarajuće ulaze. To je nadzirani algoritam učenja koji omogućuje ispravljanje mreže s obzirom na određene pogreške [9].

BPTT radi tako što odmotava sve ulazne vremenske korake (eng. timesteps). Svaki vremenski korak ima jedan ulaz, jednu kopiju mreže i jedan izlaz. Pogreške se zatim izračunavaju i akumuliraju za svaki vremenski korak. Nakon toga se mreža zamota i težine se obnove. Svaki korak odmotane ponavljajuće neuronske mreže može se promatrati kao dodatni sloj s obzirom na ovisnost o redoslijedu problema, a unutarnje stanje iz prethodnog vremenskog sloja uzima se kao ulaz na naknadnom vremenskom koraku. Ovaj algoritam se može predstaviti u sljedećim koracima [9]:

- Predstavljanje niza vremenskih koraka ulaznih i izlaznih parova na mreži.
- Odmotavanje mreže, a zatim izračunavanje i akumuliranje pogrešaka u svakom vremenskom koraku.
- Zamotavanje mreže i obnavlja težina.
- Ponavljanje ovih koraka.

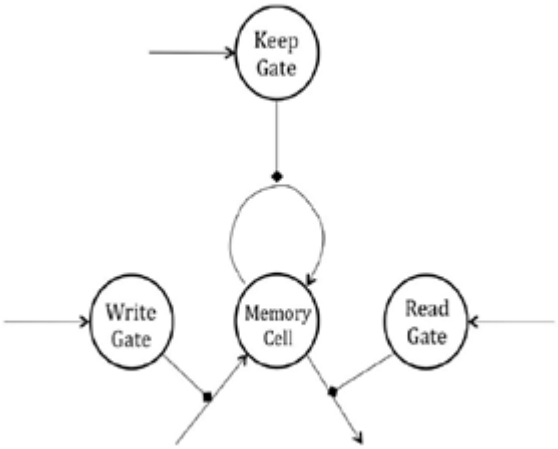
BPTT postaje sve zahtjevniji proces za računalo kako se broj vremenskih koraka povećava. Zbog tog razloga se u praksi koristi takozvana skraćena unatražno rasprostiranje kroz vrijeme (TBPTT. Eng. Truncated Backpropagation Through Time).

TBPTT je modificirana verzija BPTT algoritma treninga za ponavljajuće neuronske mreže gdje se slijed obrađuje jedan po jedan korak i povremeno (k1 vremenskih koraka) BPTT ažuriranje se izvodi natrag za fiksni broj vremenskih raspona (k2 puta). Ovaj algoritam se može predstaviti u sljedećim koracima [9]:

- Prikaz niza k1 (varijabla k1 – broj proslijeđenih vremenskih koraka između ažuriranja koji utječe na to koliko će trening biti spor ili brz, s obzirom na to koliko često se provode ažuriranja težine) vremenskih koraka ulaznih i izlaznih parova na mreži.
- Odmotavanje mreže, a zatim izračunavanje i akumuliranje pogrešaka u k2 (varijabla k2 – broj vremenskih koraka koji se primjenjuju u BPTT, ovaj broj trebao bi biti dovoljno velik da uhvati vremensku strukturu u problemu kako bi mreža naučila) vremenskim koracima.
- Zamotavanje mreže i ažuriranje težina.
- Ponavljanje ovih koraka.

3.3. Ograđena ponavljajuća neuronska mreža

U ograđenim RNN-u uglavnom postoje tri vrata (eng. gates), a to su ulazna/zapisna vrata, vrata za zadržavanje/pamćenje i vrata za izlaz/čitanje. Ova vrata su odgovorna za dopuštanje/onemogućavanje protoka signala iz odgovarajućih stanja. To znači da je vrijednost bilo kojih vrata 0 ili 1 u bilo kojem trenutku, što u biti znači da je signal dopušten ili zaustavljen. Ako je vrijednost ulaznih vrata 0, tada je protok ulaznih informacija u tom trenutku zaustavljen, a kada bi vrijednost bila 1 tada je protok ulaznih informacija u tom trenutku dopušten. Umjesto da se u potpunosti dopusti/zabrani informacija, zbog bolje preciznosti, uvedeno je djelomično propuštanje informacije, što u biti znači da vrata uzimaju vrijednosti između 0 i 1, a ne samo 0 ili 1 [10].



Slika 3.4 Ograđena RNN-a

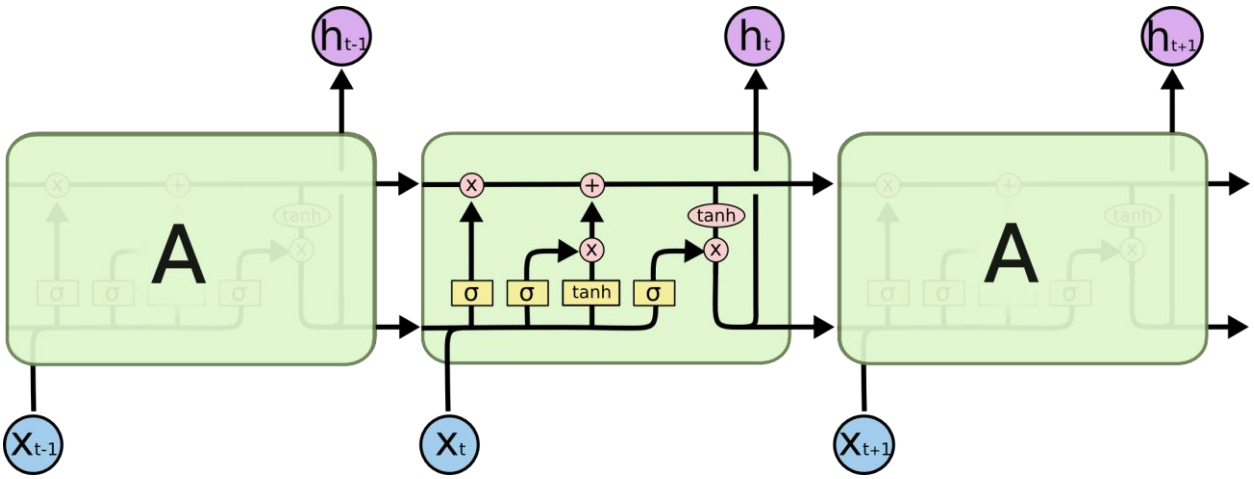
(izvor: https://medium.com/analytics-vidhya/i-strongly-recommend-to-first-know-how-rnn-algorithm-works-to-get-along-with-this-post-of-gated-9b4bf4f0ced2)

Izlazne vrijednost vrata (između 0 i 1) su u biti i izlazne vrijednosti sigmoidne funkcije. Izlazna vrijednost vrata su kontrolirana od strane ulaza što znači da su svaka vrata neuronska mreža sa skrivenim slojem. Dakle, protok informacija kroz vrata kontrolira neuronska mreža koja ima svoj ulaz kao trenutno stanje i stanje memorije u danom trenutku **t**. Na ovaj način se kontrolira protok informacija u ograđenima RNN-ima. Na temelju ove arhitekture su građeni LSTM model i model Ograđene ponavljajuće jedinice (GRU – eng. Gated Recurrent Units) [10].

3.4. Duga Kratkoročna Memorija (LSTM – eng. Long Short Term Memory)

LSTM su proširenje za ponavljajuće neuronske mreže, koje u osnovi proširuje memoriju. Jedinice LSTM-a koriste se kao građevne jedinice za slojeve RNN-a, koji se često nazivaju LSTM mrežom. LSTM-ovi omogućuju RNN-ovima da pamte unose tijekom dugog vremenskog razdoblja. To je zato što LSTM sadrže informacije u memoriji, slično kao memorija računala. LSTM može čitati, pisati i brisati informacije iz svoje memorije. Ova memorija se može promatrati kao ograđena ćelija, što znači da ćelija odlučuje hoće li pohraniti ili izbrisati informaciju, na temelju važnosti koju pripisuje informaciji [7].

Sve RNN imaju oblik lanca ponavljajućih modula neuronske mreže. U standardnim RNN-ovima, ovaj ponavljajući modul imat će vrlo jednostavnu strukturu, kao što je jedan sloj „tanh“. LSTM-ovi također imaju ovu lančanu strukturu, ali modul koji se ponavlja ima drugačiju strukturu. Umjesto da imaju jedan sloj neuronske mreže, postoje četiri [11].



Slika 3.5 Dijagram LSTM mreže

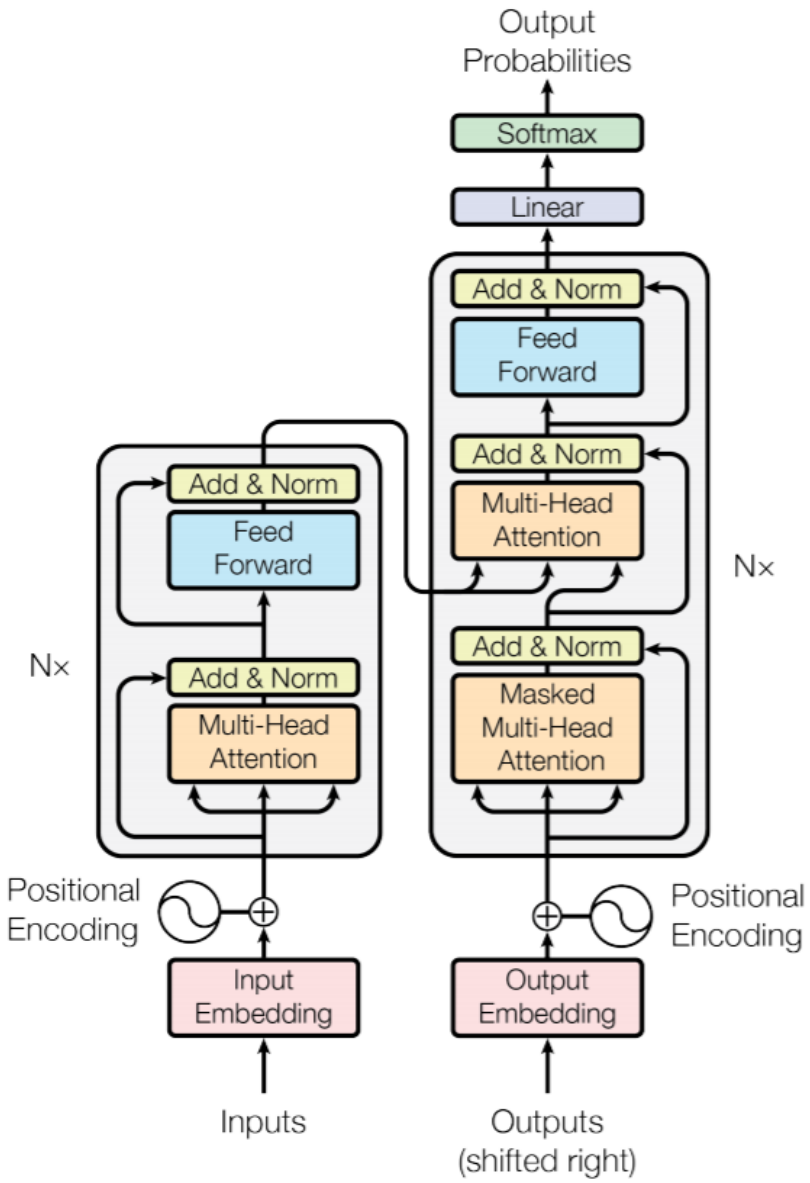
(izvor: https://colah.github.io/posts/2015-08-Understanding-LSTMs)

LSTM arhitektura se sastoji od četiri dijela i svaki dio obavlja pojedinačnu funkciju. Prvi dio se zove ćelija (eng. cell), drugi dio je poznat kao ulazna vrata (eng. Input gate), treći dio je zaboravljajuća vrata (eng. Forget gate), a posljednji je izlazna vrata (eng. Output gate) [12] [13].

Zadatak ćelije je pohranjivanje vrijednosti nakon određenog vremenskog intervala, dok je zadatak ostalih triju vrata regulirati protok podataka do i iz ćelije.

4. Transformer neuronska mreža

Većina postojećih modela prijenosa neuronske sekvence ima strukturu koder-dekoder. Ovdje koder preslikava ulazni slijed prikaza simbola (x1, ..., xn) u niz kontinuiranih prikaza z = (z1, ..., zn). S obzirom na z, dekoder tada generira izlazni niz (y1, ..., ym) simbola jedan po jedan element. U svakom koraku model je autoregresivan, konzumirajući prethodno generirane simbole kao dodatni ulaz prilikom generiranja sljedećeg. Transformer slijedi ovu arhitekturu koristeći složenu samopažnju, te koristi potpuno povezane slojeve i za koder i za dekoder [14].



Slika 4.1 Arhitektura transformer neuronske mreže

(izvor: arXiv:1706.03762v5)

Originalni model transformera je stog od 6 slojeva. Izlaz sloja l je unos sloja l+1 dok se ne postigne konačno predviđanje. Postoji 6-slojni stog kodera s lijeve strane i 6-slojni stog dekodera s desne strane. Pogledajte sliku 4.1.

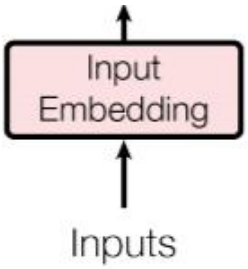
4.1. Koder stog (eng. Encoder stack)

Izvorna struktura sloja kodera ostaje ista za svih N=6 slojeva modela transformera. Svaki sloj sadrži dva glavna podsloja: višeslojni mehanizam pozornosti i potpuno povezana mreža usmjerena naprijed. Postoji zaostala veza koja okružuje svaki glavni podsloj, *sublayer(x),* u modelu transformera. Ove veze prenose neobrađeni ulaz *x* podsloja u funkciju normalizacije sloja. Na ovaj način je osigurano da se ključne informacije kao što je pozicijsko kodiranje neće izgubiti na putu. Na primjer, podsloj za ugrađivanje (eng. embedding) prisutan je samo na donjoj razini stoga. Ostalih pet slojeva ne sadrže sloj za ugradnju, a to jamči da je kodirani ulaz stabilan kroz sve slojeve. Mehanizmi pažnje s više glava (eng multi-head attention) obavljaju iste funkcije od sloja 1 do 6. Međutim, oni ne obavljaju iste zadatke. Svaki sloj uči od prethodnog sloja i istražuje različite načine povezivanja tokena (pojavnica) u nizu [15].

Dizajneri transformera uveli su vrlo učinkovito ograničenje. Izlaz svakog podsloja modela ima konstantnu dimenziju, uključujući sloj za umetanje ulaza i preostale veze. Ova dimenzija je d_{model} i može se postaviti na drugu vrijednost ovisno o željenim ciljevima. U originalnoj arhitekturi transformera, d_{model} =512. Zbog d_{model}, praktički sve ključne operacije su skalarni umnožak. Dimenzije ostaju stabilne, što smanjuje broj operacija za izračunavanje, smanjuje potrošnju stroja i olakšava praćenje informacija dok prolaze kroz model [15].

4.1.1. Umetanje ulaza (eng. Input embedding)

Podsloj za ugrađivanje ulaza pretvara ulazne tokene u vektore dimenzije d_{model} = 512 koristeći već naučene ugradnje u izvornom modelu transformera.



Slika 4.2 Podsloj za umetanje ulaza u transformeru

(izvor: arXiv:1706.03762v5)

Podsloj za ugradnju radi kao i drugi standardni modeli prijenosa. Algoritam za tokenizaciju će transformirati rečenicu u tokene. Svaki tokenizer ima svoje metode, ali rezultati su slični. Tokenizer će normalizirati niz na mala slova i skratiti ga na poddijelove. Tokenizer će općenito osigurati cjelobrojni (eng. integer) prikaz koji će se koristiti za proces ugradnje. Tokenizirani tekst se sada može proslijediti u već naučen podsloj za umetanje. Mnoge metode ugradnje mogu se primijeniti na tokenizirani ulaz. Jedna od tih metoda može biti skip-gram arhitektura. Skip-gram će se usredotočiti na središnju riječ u rasponu riječi i predviđa riječi konteksta. Ciljana riječ je ulaz, dok izlaz sadrži umetnute (eng. embedded) riječi tokeniziranih ulaznih riječi [15].

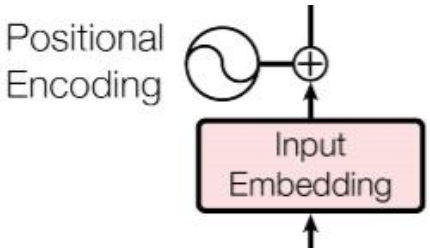
Skip-gram jedna je od tehnika učenja bez nadzora koja se koristi za pronalaženje najsrodnijih riječi za određenu riječ. Koristi se za predviđanje kontekstualne riječi za danu riječ [16].

Budući da je $d_{model} = 512$, dobit ćemo vektor veličine 512 za svaku riječ.

Transformerovi sljedeći slojevi ne počinju proračune bez ijedne informacije, jer su naučili ugradnje riječi koje već pružaju informacije o tome kako se riječi mogu povezati. Međutim, nedostaje veliki dio informacija jer nijedan dodatni vektor ili informacija ne ukazuje na poziciju riječi u nizu [15].

4.1.2. Pozicijsko kodiranje (eng. Positional encoding)

Podaci ulaze u funkciju pozicijskog kodiranja transformera bez informacije o položaju riječi u nizu.



Slika 4.3 Pozicijsko kodiranje u transformeru

(izvor: arXiv:1706.03762v5)

Ne može se stvoriti neovisne vektore za poziciju koji bi imali negativan utjecaj na brzinu treninga transformera i učinili podslojeve pažnje vrlo složenim za rad. Ideja je dodati vrijednost pozicijskog kodiranja na izlaz umetanja ulaza umjesto dodatnih vektora za opisivanje položaja tokena u nizu. Problem je pronaći način da se doda vrijednost ugradnji svake riječi tako da ona ima tu informaciju. Međutim, moramo dodati tu vrijednost na dimenziju $d_{model} = 512$. Za svaki

vektor umetnute riječi, moramo pronaći način da pružimo informacije za *i* u rasponu od 0 do 512 dimenzija vektora umetnutih riječi [15].

Kako bi se ostvario ovaj cilj, ponuđen je način za pozicijskog kodiranja sa sinusnim i kosinusnim vrijednostima, jednadžba 4.1.

$$PE(pos, 2i) = \sin(pos/10000^{2i/dmodel})$$
$$PE(pos, 2i+1) = \sin(pos/10000^{2i/dmodel})$$

Jednadžba 4.1

(izvor: arXiv:1706.03762v5)

Nakon što je generirani vektor za pozicijsko kodiranje *PE* (generiran jednadžba 4.1), potrebno ga je nadodati na vektor za umetnute riječi *y*. Postoje mnoge mogućnosti za povećanje vrijednosti *y* kako bi se osiguralo da se informacije sloja za ugradnju riječi mogu učinkovito koristiti u sljedećim slojevima. [15], jednadžba 4.2.

$$\left(y * \sqrt{dmodel}\right) + PE(pos)$$

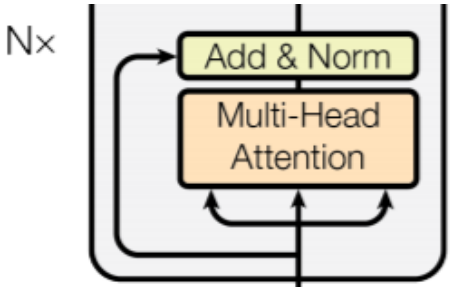
Jednadžba 4.2

(izvor: Rothman, Denis. Transformers for Natural Language Processing: Build innovative deep neural network architectures for NLP with Python, PyTorch, TensorFlow, BERT, RoBERTa, and more. *Packt Publishing Ltd, 2021.)*

Izlaz pozicijskog kodiranja je ulaz za podsloj pažnje s više glava.

4.1.3. Pozornost s više glava (eng. Multi-head attention)

Podsloj pozornosti s više glava sadrži osam glava i slijedi ga normalizacija nakon sloja, koja će dodati preostale veze na izlaz podsloja i normalizirati ga.



Slika 4.4 Podsloj pozornosti s više glava

(izvor: arXiv:1706.03762v5)

Ulaz podsloja višestruke pozornosti prvog sloja stoga kodera je vektor koji sadrži ugradnju i pozicijsko kodiranje svake riječi. Sljedeći slojevi stoga ne započinju ove operacije ispočetka. Svaka se riječ preslikava na sve ostale riječi kako bi se odredilo kako se uklapa u niz. Model će se trenirati kako bi otkrio je li riječ "A" povezana s riječ "B" ili "C". Mogli bismo izvesti ogroman izračun trenirajući model koristeći d_{model} = 512 dimenzija. Međutim, mogli bismo dobiti samo jednu po jednu točku gledišta analizirajući slijed s jednim d_{model} blokom. Bolji način je podijeliti d_{model} = 512 dimenzija svake riječi x_n od x (sve riječi niza) na 8 d_k = 64 dimenzija. Sada bi paralelno radile 8 glava. Jedna glava bi mogla odlučiti da se riječ "A" dobro uklapa uz riječ "B", a druga da riječ "A" dobro pristaje uz riječ "B". Izlaz svake glave je matrica z_i oblika x* d_k. Međutim, Z mora biti spojen tako da izlaz podsloja s više glava ne bude niz dimenzija već jedan redak matrice x_m*d_{model} [15].

Unutar svake glave h_n podsloja pozornosti, svaki vektor riječi ima tri prikaza. Prvi je vektor upita (Q) koji ima dimenziju d_q = 64, koji se aktivira i trenira kada vektor riječi x_n traži sve parove ključ/vrijednost drugih vektora riječi, uključujući sebe u samopozornost. Drugi je vektor ključa (K) koji ima dimenziju d_k = 64, koji će biti uvježban da pruži vrijednost pozornosti. Treći je vektor vrijednosti (V) koji ima dimenziju d_v = 64, koji će biti uvježban da pruži drugu vrijednost pažnje. Pozornost je definirana kao "Skalirana pozornost na skalarni umnožak" [15], jednadžba 4.3.

$$Attention(Q, K, V) = softmax\left(\frac{Q * K^T}{\sqrt{dk}}\right) * V$$

Jednadžba 4.3

(izvor: Rothman, Denis. Transformers for Natural Language Processing: Build innovative deep neural network architectures for NLP with Python, PyTorch, TensorFlow, BERT, RoBERTa, and more. *Packt Publishing Ltd, 2021.)*

Svi vektori imaju istu dimenziju što čini relativno jednostavnom korištenje skaliranog skalarnog umnoška za dobivanje vrijednosti pažnje za svaku glavu, a zatim spajanje izlaza Z od 8 glava. Da bismo dobili Q, K i V, moramo trenirati model s odgovarajućim matricama težine Q_w, K_w i V_w, koje imaju d_k = 64 stupca i d_{model} = 512 redaka. Na primjer, Q se dobiva skalarnim umnoškom između x i Q_w. Q će imati dimenziju d_k = 64 [15].

Svaki pod-sloj pozornosti i svaki podsloj naprijed u transformeru prati normalizacija nakon sloja (Post-LN). Post-LN sadrži funkciju zbrajanja i proces normalizacije sloja. Funkcija dodavanja obrađuje preostale veze (eng. residual connections) koje dolaze s ulaza podsloja. Cilj preostalih veza je osigurati da se kritične informacije ne izgube [15], jednadžba 4.4.

$$LayerNorm(x + Sublayer(x))$$

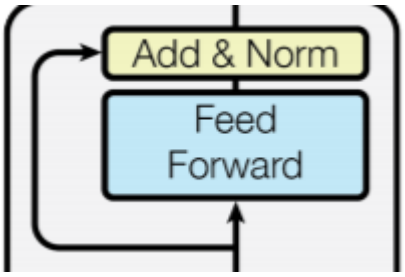
Jednadžba 4.4

(izvor: Rothman, Denis. Transformers for Natural Language Processing: Build innovative deep neural network architectures for NLP with Python, PyTorch, TensorFlow, BERT, RoBERTa, and more. *Packt Publishing Ltd, 2021.)*

Sublayer(x) je sam podsloj. *x* je informacija dostupna na ulaznom koraku *Sublayer(x)*. Ulaz *LayerNorm*-a je vektor *v* koji je rezultat *x* + *Sublayer*(*x*). d_{model} = 512 za svaki ulaz i izlaz transformera, što standardizira sve procese. Sljedeći podsloj sada može obraditi izlaz post-LN ili *LayerNorm(v)*. U ovom slučaju, podsloj je mreža za prosljeđivanje [15].

4.1.4. Mreža za prosljeđivanje (FFN - eng. Feedforward network)

Ulaz FFN-a je d_{model} = 512 izlaz post-LN-a prethodnog podsloja.



Slika 4.5 Podsloj za prosljeđivanje

(izvor: arXiv:1706.03762v5)

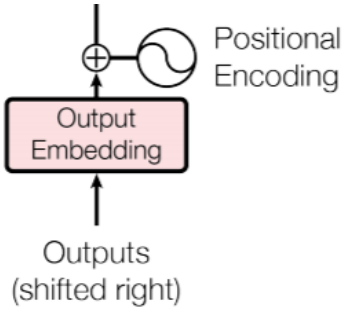
FFN-ovi u koderu i dekoderu su potpuno povezani. FFN je poziciona mreža. Svaka pozicija se obrađuje zasebno i na identičan način. FFN sadrži dva sloja i primjenjuje funkciju ReLU aktivacije. Ulaz i izlaz FFN slojeva je d_{model} = 512, ali je unutarnji sloj veći s d_{ff} = 2048. Izlaz FFN-a ide na Post-LN, nakon toga se izlaz šalje sljedećem sloju u koder stogu i sloju pažnje s više glava u dekoder stogu [15].

4.2. dekoder stog (eng. Decoder stack)

Struktura sloja dekodera ostaje ista kao koder za svih *N* = 6 slojeva modela transformera. Svaki sloj sadrži tri podsloja: maskirani podsloj pozornosti s više glava, podsloj pozornosti s više glava i mreža za prosljeđivanje. U dekoderu, maskirani mehanizam pažnje s više glava je treći glavni podsloj koji se ne javlja u koderu. U ovom izlazu podsloja, na danoj poziciji, sljedeće riječi su

maskirane tako da transformer temelji svoje pretpostavke na svojim zaključcima bez da vidi ostatak niza. Na taj način, u ovom modelu, ne može vidjeti buduće dijelove niza. Izlaz svakog podsloja stog dekodera ima konstantnu dimenziju d_{model} kao u stog kodera, uključujući sloj za ugradnju i izlaz preostalih veza. Struktura svakog podsloja i funkcija dekodera slična je koderu. Ulazni niz u dekoder je prevedeni ulazni niz kodera koji se uči [15].

Podsloj za umetanje izlaza (eng. output embedding) i pozicijsko kodiranje u dekoderu imaju istu funkcionalnost kao i u koderu. Izlazne riječi prolaze kroz sloj za ugradnju riječi, a zatim funkciju pozicijskog kodiranja, kao u dekoder stogu [15].



Slika 4.6 Umetanje izlaza i pozicijsko kodiranje u dekoder stogu transformera (izvor: arXiv:1706.03762v5)

Transformer je auto-regresivni model. Koristi prethodne izlazne sekvence kao dodatni ulaz. Slojevi pozornosti s više glava dekodera koriste isti proces kao i koder. Međutim, maskirani podsloj pozornosti s više glava (*sublayer 1*) dopušta primjenu pozornosti samo na položaje do i uključujući trenutnu poziciju. Buduće riječi su skrivene od transformera, što ga prisiljava da nauči kako predviđati. Proces post-LN slijedi nakon maskiranog podsloja pozornosti s više glava. Podsloj pozornosti s više glava također prati samo položaje do trenutne pozicije koju transformer predviđa kako bi izbjegao uvid u slijed koji mora predvidjeti. Podsloj pozornosti s više glava vadi informacije iz kodera uzimajući u obzir *encoder (K, V)* tijekom operacija pozornosti skalarnog umnoška. Ovaj podsloj također vadi informacije iz maskiranog podsloja pozornosti s više glava (maskirana pozornost) uzimajući u obzir i *sublayer 1 (Q)* tijekom operacija skalarnog umnoška. Dekoder tako koristi uvježbane informacije kodera. Proces post-LN slijedi nakon podsloja pozornosti s više glava. Nakon post-FN-a podaci se šalju u FFN. FFN podsloj ima istu strukturu kao i FFN podsloj u koder stogu. Post-LN FFN-a radi isto kao i post-LN u koder stogu. transformer proizvodi izlaznu sekvencu od samo jednog elementa u isto vrijeme. Linearni sloj će proizvesti sljedeće vjerojatne elemente niza koje će *softmax* funkcija pretvoriti u vjerojatni element. Sloj

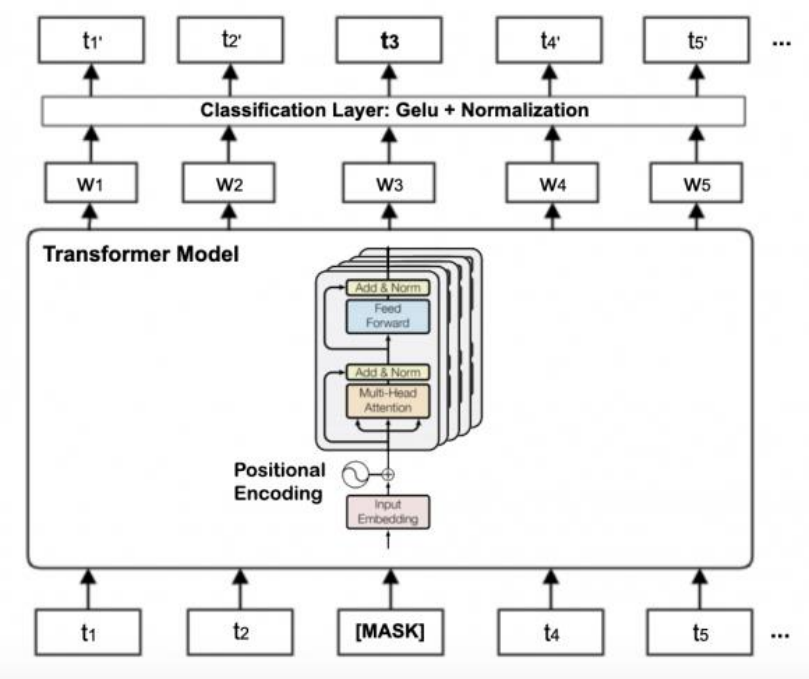
dekodera kao i sloj enkodera će tada ići od sloja *i* do sloja *i+1* sve do gornjeg sloja *N=6*-slojnog stoga transformera [15].

5. Modeli transformer neuronske mreže

Sljedeći modeli transformer neuronske mreže će biti detaljnije objašnjeni: Dvosmjerni prikaz kodera od transformera (BERT – eng. Bidirectional Encoder Representations from Transformers), Robusno optimiziran BERT predtrening pristup (RoBERTa – eng. Robustly Optimized BERT Pre-training Approach), ALBERT (ALBERT - eng. A Lite BERT) i XLNet.

5.1. BERT model

BERT koristi transformer, mehanizam pozornosti koji uči kontekstualne odnose između riječi u tekstu. U svom originalnom obliku, transformer uključuje dva odvojena mehanizma - koder koji čita unos teksta i dekoder koji proizvodi predviđanje zadatka. Budući da je BERT-ov cilj generirati jezični model, potrebna je samo koder stog. Za razliku od usmjerenih modela (eng. directional models), koji uzastopno čitaju unos teksta (s lijeva na desno ili s desna na lijevo), koder transformera čita cijeli niz riječi odjednom. Stoga se smatra dvosmjernim modelom. Ova karakteristika omogućuje modelu da nauči kontekst riječi na temelju cijele okoline (lijevo i desno od riječi). Mnogi modeli predviđaju sljedeću riječ u nizu, ovaj pristup ograničava učenje konteksta. Kako bi prevladao ovaj izazov, BERT koristi dvije strategije treninga: „Maskirani jezični model“ (MLM - eng. Masked language models) i „Predviđanje sljedeće rečenice“ (eng. Next sentence prediction) [17].



Slika 5.1 Arhitektura BERT modela

(izvor: https://journals.openedition.org/ijcol/472)

Za razliku od standardnog kodera stoga, BERT model uvodi novi sloj za klasifikaciju prije umetanja izlaza, slika 5.1.

MLM funkcionira tako da se prije unošenja nizova riječi u BERT, nasumično maskira neki postotak ulaznih tokena, a zatim se predviđaju ti maskirani tokeni [18]. Najčešće je 15% riječi u svakom nizu zamijenjeno tokenom [MASK]. Model zatim pokušava predvidjeti izvornu vrijednost maskiranih riječi, na temelju konteksta koje pružaju druge, nemaskirane, riječi u nizu. Funkcija gubitka (eng. loss function) kod BERT modela uzima u obzir samo predviđanje maskiranih vrijednosti i zanemaruje predviđanje nemaskiranih riječi. Kao posljedica toga, model konvergira sporije od usmjerenih modela, što je karakteristika koja je nadoknađena njegovom povećanom svjesnošću konteksta [17].

Mnogi važni zadaci kao što su odgovaranje na pitanja (QA) i zaključivanje prirodnog jezika (eng. - Natural language inference) temelje se na razumijevanju odnosa između dviju rečenica. Kako bismo uvježbali model koji razumije rečenične odnose, taj model se trenira za binarizirani zadatak predviđanja sljedeće rečenice (NSP, eng. Next sentence prediction) koji se može trivijalno generirati iz bilo kojeg jednojezičnog korpusa [18].

U procesu BERT obuke, model prima parove rečenica kao ulaz i uči predvidjeti je li druga rečenica u paru sljedeća rečenica u izvornom dokumentu. Tijekom treninga 50% ulaza je par u kojem je druga rečenica sljedeća rečenica u izvornom dokumentu, dok se u ostalih 50% kao druga rečenica bira nasumična rečenica iz baze podataka. Pretpostavka je da će nasumična rečenica biti odvojena od prve rečenice. Kako bi model mogao razlikovati dvije rečenice u treningu, unos se obrađuje na sljedeći način prije ulaska u model: Token [CLS] (eng. classification) umeće se na početak prve rečenice, a [SEP] (eng. separator) token se umeće na kraj svake rečenice [17].

Prilikom treniranja BERT modela, „Maskirani jezični model“ i „Predviđanje sljedeće rečenice“ se treniraju zajedno, s ciljem minimiziranja kombinirane funkcije gubitka (eng loss function) dviju strategija.

U *fine-tuning* treniranju BERT modela, većina parametri ostaju isti, no neki se mogu podešavati kako bi se poboljšao rezultat.

5.2. RoBERTa model

RoBERTa model je zapravo fino podešen BERT model. Primijećeno je da treniranje BERT-a na većim skupovima podataka uvelike poboljšava njegov rezultat. Dakle, RoBERTa je treniran na ogromnom skupu nekomprimiranog teksta.

Cilj maskiranog jezika u predtreningu BERT-a u biti je nasumično maskiranje nekoliko tokena iz svake sekvence, a zatim predviđanje tih tokena. Međutim, u izvornoj implementaciji BERT-a,

sekvence su maskirane samo jednom u predprocesiranju. To implicira da se isti uzorak maskiranja koristi za isti slijed u svim koracima treninga. Tu se javlja dinamično maskiranje, pri čemu se uzorak maskiranja generira svaki put kada se sekvenca unese u model. Dinamičko maskiranje ima usporedive ili nešto bolje rezultate od statičkog maskiranja. Stoga je u RoBERTa dinamički pristup maskiranju usvojen za predtrening [19].

BERT kao ulaz uzima konkatenaciju dva segmenta (sekvence tokena), *x1, . . . , xN* i *y1, . . . , yM*. Segmenti se obično sastoje od više prirodnih rečenica. Dva segmenta su predstavljena kao jedan ulazni niz za BERT s posebnim tokenima koji ih razgraničavaju *[CLS], x1, . . . , xN , [SEP], y1, . . . , yM, [EOS]*. *M* i *N* su ograničeni tako da je $M + N < T$, gdje je *T* parametar koji kontrolira maksimalnu duljinu niza tijekom treninga [20].

(Napomena: najčešća korištena vrijednost za T je 512, tako da u ostatku ovoga dijela rada će biti pretpostavljeno da je T = 512.)

Pretpostavlja se da je gubitak kod predvđanja sljedeće rečenice (NSP gubitak) važan čimbenik u treniranju izvornog BERT modela. Međutim, neki noviji radovi doveli su u pitanje nužnost NSP gubitka. Postoje različite vrste ulaznih reprezentacija koje bi se mogle koristiti umjesto NSP-a s BERT-om [20]:

- SEGMENT-PAIR + NSP – Ovo slijedi izvorni ulazni format korišten u BERT-u, s NSP gubitkom. Svaki ulaz ima par segmenata, od kojih svaki može sadržavati više prirodnih rečenica, ali ukupna kombinirana duljina mora biti manja od 512 tokena (što je maksimalna fiksna duljina sekvence za BERT model) [20].
- SENTENCE-PAIR + NSP – Isto kao i SEGMENT-PAIR reprezentacija, samo s parovima rečenica. Međutim, očito je da bi ukupna duljina sekvenci ovdje bila puno manja od 512. Stoga se koristi veća veličina serija tako da je broj obrađenih tokena po koraku treninga sličan onom u SEGMENT-PAIR [20].
- FULL-SENTENCES – Svaki unos ima mnogo cijelih rečenica uzorkovanih uzastopno iz jednog ili više dokumenata, tako da ukupna duljina iznosi najviše 512 tokena. Kada dođemo do kraja jednog dokumenta, počinjemo uzorkovati rečenice iz sljedećeg dokumenta i dodajemo dodatni token za razdvajanje između dokumenata. Ovaj postupak uklanja potrebu za NSP gibutkom[20] .
- DOC-SENTENCES – Unosi su konstruirani slično kao u FULL-SENTENCES, osim što ne smiju prijeći granice dokumenta. Unosi uzorkovani pri kraju dokumenta mogu biti kraći od 512 tokena, tako da dinamički povećavamo veličinu serije u tim slučajevima kako bismo postigli sličan broj ukupnog broja tokena kao u FULL-SENTENCES. Ovaj postupak uklanja potrebu za NSP gubitkom [20].

Transformer i BERT modeli su podložni velikim serijama. Velika serija čini optimizaciju bržom i može poboljšati izvedbu završnog zadatka kada se ispravno podesi (u slučaju ovih modela). S povećanjem veličine serije, prolazi treninga se prilagođavaju, tj. zadani slijed će u konačnici biti optimiziran za isti broj puta. Na primjer, veličina serije od 256 za 1M koraka je ekvivalentna treningu s veličinom serije od 2K za 125K koraka i s veličinom serije od 8K za 31K koraka. [19]

Izvorna implementacija BERT-a koristi kodiranje para bajtova (BPE, eng. Byte-Pair Encoding) riječnik na razini znakova veličine 30K, koji se uči nakon predobrade unosa s heurističkim pravilima tokenizacije. RoBERTa-a model koristi za kodiranje teksta (eng. text encoding) BPE. BPE je hibrid između prikaza na razini znakova i riječi koji omogućuje rukovanje velikim rječnicima uobičajenim u korpusima prirodnog jezika. Umjesto punih riječi, BPE se oslanja na jedinice podriječi (eng. subword), koje se izdvajaju provođenjem statističke analize korpusa treniranja. BPE veličine riječnika obično se kreću od 10K-100K jedinica podriječi. Međutim, *unicode* znakovi mogu činiti znatan dio ovog rječnika pri modeliranju velikih i raznolikih korpusa. Korištenje bajtova omogućuje učenje vokabulara podriječi skromne veličine (50K jedinica) koji još uvijek može kodirati bilo koji ulazni tekst bez uvođenja "nepoznatih" tokena [20].

5.3. ALBERT model

S obzirom da trenutni najmoderniji modeli često imaju stotine milijuna ili čak milijarde parametara, lako je pogoditi ta ograničenja dok se skalira model. Brzina treninga također može biti značajno otežana u distribuiranom treningu, budući da su komunikacijski troškovi izravno proporcionalni broju parametara u modelu. ALBERT uključuje dvije tehnike smanjenja parametara koje uklanjaju glavne prepreke u skaliranju unaprijed istreniranih modela. Prva je faktorizirana parametrizacija ugradnje (eng. factorized embedding parameterization). Razlaganjem velike matrice rječnika na dvije male matrice, odvajamo veličinu skrivenih slojeva od veličine rječnika. Ovo razdvajanje olakšava povećanje skrivene veličine bez značajnog povećanja veličine parametara rječnika. Druga tehnika je dijeljenje parametara na više slojeva. Ova tehnika sprječava rast parametra s dubinom mreže. Obje tehnike značajno smanjuju broj parametara za BERT bez ozbiljnog narušavanja performansi, čime se poboljšava parametarska učinkovitost. ALBERT konfiguracija slična BERT-*large* ima 18*x* manje parametara i može se trenirati oko 1,7*x* brže. Tehnike smanjenja parametara također djeluju kao oblik regularizacije koji stabilizira trening i pomaže u generalizaciji. Kako bi se dodatno poboljšala izvedba ALBERTA, uvodi se samonadzirani gubitak za predviđanje reda rečenica (SOP – eng. Sentence-order prediction). Primarni SOP usredotočuje se na međurečeničnu koherentnost i osmišljen je za

rješavanje neučinkovitosti gubitka predviđanja sljedeće rečenice (NSP) predloženog u izvornom BERT-u [21].

5.4. XLNet model

XLNet je auto-regresivni jezični model koji daje zajedničku vjerojatnost niza tokena na temelju arhitekture transformera s ponavljanjem. Njegov cilj treninga je da izračuna vjerojatnost oznake riječi koja je uvjetovana svim permutacijama znakova riječi u rečenici, za razliku od samo onih lijevo ili samo onih desno od ciljnog tokena [22].

Umjesto korištenja fiksnog redoslijeda faktorizacije naprijed ili natrag kao u konvencionalnim autoregresivnim modelima, XLNet maksimizira očekivanu *log* vjerojatnost niza s obzirom na sve moguće permutacije reda faktorizacije. Zahvaljujući operaciji permutacije, kontekst za svaku poziciju može se sastojati od tokena s lijeve i desne strane. U očekivanju, svaka pozicija uči koristiti kontekstualne informacije sa svih pozicija, tj. hvatanje dvosmjernog konteksta. Kao generalizirani autoregresivni jezični model, XLNet se ne oslanja na oštećenje podataka. Dakle, XLNet nema problema s nepodudarnošću između predtreninga i finog podešavanja, kojeg ima BERT model. U međuvremenu, autoregresivni cilj također pruža prirodan način korištenja pravila množenja za faktorizaciju zajedničke vjerojatnosti predviđenih tokena, eliminirajući pretpostavku neovisnosti napravljenu u BERT-u [23].

Inspiriran najnovijim napretkom u modeliranju autoregresivnog jezika, XLNet integrira mehanizam ponavljanja segmenta i relativnu shemu kodiranja transformer-XL u predtreniranju, što poboljšava performanse, posebno za zadatke koji uključuju duži slijed teksta [23].

Glavne karakteristike XLNet-a:

- Permutacije (eng. Permutations) – S obzirom na sekvencu *x*, autoregresivni model je onaj koji izračunava vjerojatnost $Pr(xi \mid x<i)$. U modeliranju jezika, to je vjerojatnost da se u rečenici pojavi leksema *xi*, uvjetovana oznakama $x<i$ koji joj prethode. Ove uvjetovane riječi nazivaju se kontekstom. Takav model je asimetričan i ne uči iz svih tokenskih odnosa u korpusu. Autoregresivni modeli omogućuju modelu da uči iz odnosa između tokena i onih koji ga slijede. Autoregresivni cilj se u ovom slučaju može vidjeti kao $Pr(xi) = Pr(xi \mid x>i)$. Autoregresivna je u obrnutom slijedu. Moglo bi biti zanimljivih relacija iz kojih bismo mogli naučiti ako pogledamo samo dva najbliža tokena: $Pr(xi) = Pr(xi \mid xi-1, xi+1)$ ili bilo koju kombinaciju tokena $Pr(xi) = Pr(xi \mid xi-1, xi+2, xi-3)$. XLNet predlaže korištenje cilja koji je očekivanje nad svim takvim permutacijama. Uzmimo niz *x* sa $T=4$ tokena. Skup od svih 4! permutacije bio bi $Z = \{[1, 2, 3, 4], [1, 2, 4, 3], \ldots, [4, 3, 2, 1]\}$. XLNet model je auto-regresivan na sve takve

permutacije; može izračunati vjerojatnost tokena *xi* danih prethodnih tokena *x<i* iz bilo kojeg reda *z* od *Z* [22].

- Maska za pozornost (eng. Attention mask) – Originalni transformer ugrađuje pozicijsko kodiranje i na tokene. Ako bi kasnije miješali te tokene onda to pozicijsko kodiranje nije potrebno. Ovaj problem se rješava korištenjem maske za pozornost. Kada model izračunava kontekst koji je ulaz za izračun vjerojatnosti, on to uvijek čini koristeći isti redoslijed tokena i jednostavno maskira one koji nisu u kontekstu koji se razmatra (tj. one koji dolaze naknadno pomiješanim redoslijedom) [22].
- Dvostruka samopozornost (eng. two-stream self-attention) – Svaka pozicija tokena *i* ima dva povezana vektora na svakom sloju samopozornosti *m:* $\boldsymbol{h}_i^m$ i $\mathbf{g}_i^m$. Vektori *h* pripadaju toku sadržaja, dok *g* vektori pripadaju toku upita. Vektori toka sadržaja inicijaliziraju se s ugrađivanjem (eng. embedding) tokena dodanim pozicijskim ugrađivanjem. Vektori toka upita inicijaliziraju se generičkim vektorom za ugradnju *w* koji je dodan pozicijskim ugrađivanjima. *w* je isti bez obzira na token, te se stoga ne može koristiti za razlikovanje između tokena. Svaki vektor sadržaja, $\boldsymbol{h}_i$, ažurira se pomoću onih *h* koji ostaju nemaskirani i samog sebe. Na svakom sloju svaki vektor upita $\boldsymbol{g}_i$ žurira se korištenjem demaskiranih vektora sadržaja i samog sebe. Ažuriranje koristi $\boldsymbol{g}_i$ kao upit, dok koristi $\boldsymbol{h}_j$ kao ključeve i vrijednosti gdje je *j* indeks nemaskiranog tokena u kontekstu *i* [22].

6. Mjere usporedbi algoritama za NLP

Opisat će se što je i čemu služe dvojezična evaluacija strojnih prijevoda (BLEU - eng. Bilingual evaluation understudy) i opće razumijevanje jezika (GLEU – eng. General Language Understanding).

6.1. BLEU

Kvaliteta prevedenog teksta se mjeri pomoću sljedećih karakteristika [24]:

- Adekvatnost - je mjera kojom se zna da li je značenje izraženo s izvornog jezika na ciljni jezik.
- Vjernost - je stupanj u kojem prijevod točno prenosi značenje izvornog teksta.
- Tečnost - mjeri koliko su rečenice gramatički dobro oblikovane uz lakoću tumačenja.

BLEU je način za automatsko mjerenje izvedbe modela strojnog prijevoda. Ukratko, BLEU uspoređuje strojni prijevod – ono što je poznato kao *candidate translation* – s postojećim prijevodima koje je stvorio čovjek, poznatim kao *reference translations*. Kao i svaka metrika koja se temelji na preciznosti, vrijednost BLEU rezultata uvijek je broj između 0 (najgore) i 1 (najbolje) [25].

BLEU je razvijen za procjenu predviđanja sustava za automatsko strojno prevođenje. Nije savršen, ali nudi 5 uvjerljivih prednosti [26]:

- Izračunava se brzo i jeftino;
- Lakoga ga je razumjeti;
- Neovisan je o jeziku;
- U visokoj je korelaciji s ljudskom procjenom;
- Široko je prihvaćen.

BLEU radi tako da broji podudarne n-grame u kandidatskom prijevodu prema n-gramima u referentnom tekstu, gdje bi 1-gram ili unigram bio svaki token, a usporedba bigrama bila bi svaki par riječi. Usporedba se vrši bez obzira na red riječi. Brojanje podudarnih n-grama modificirano je kako bi se osiguralo da uzima u obzir pojavljivanje riječi u referentnom tekstu, ne nagrađujući mogući prijevod koji generira obilje razumnih riječi. To se naziva modificirana preciznost n-grama. Rezultat služi za usporedbu rečenica, ali je također predložena modificirana verzija koja normalizira n-grame prema njihovom pojavljivanju za bolje bodovanje blokova više rečenica. Savršena ocjena u praksi nije moguća jer bi prijevod morao točno odgovarati referenci. To nije

moguće ni ljudskim prevoditeljima. Broj i kvaliteta referenci korištenih za izračun BLEU rezultata znači da usporedba rezultata između skupova podataka može biti problematična [26].

6.2. GLEU

Mjera za procjenu općeg razumijevanja jezika (eng. General Language Understanding Evaluation - GLUE) zbirka je skupova podataka koji se koriste za obuku, procjenu i analizu NLP modela u odnosu jedan na drugi, s ciljem poticanja istraživanja u razvoju općih i robusnih sustava razumijevanja prirodnog jezika. Zbirka se sastoji od devet teških i raznolikih skupova podataka o zadacima osmišljenih za testiranje razumijevanja jezika modela i ključna je za razumijevanje načina na koji se ocjenjuju modeli prijenosa učenja poput BERT-a. Kada netko želi ocijeniti svoj model, trenira i zatim boduje model za svih devet zadataka, a dobivena prosječna ocjena tih devet zadataka je konačna ocjena izvedbe modela. Nije važno točno kako model izgleda ili kako radi sve dok može obraditi ulaze i izlaze predviđanja za sve zadatke [27].

Zadaci su namijenjeni pokrivanju raznolikog i teškog raspona NLP problema i uglavnom su preuzeti iz postojećih skupova podataka [27]:

- Korpus jezične prihvatljivosti - Binarna klasifikacija: pojedinačne rečenice koje su gramatički točne ili netočne;
- Mjerenje pozitivnosti i negativnosti recenzija određenog medija;
- Je li rečenica B parafraza rečenice A;
- Određuje jesu li dvije rečenice semantički ekvivalentne;
- Da li su dva pitanja slična;
- Da li su Rečenica A i B u suprotnosti;
- Da li rečenica B odgovara na pitanje A;
- Povlači li rečenica A rečenicu B;
- Rečenica B zamjenjuje dvosmislenu zamjenicu rečenice A jednom od imenica, i da li je ta imenica točna.

Za svaki zadatak, model mora imati svoju ulaznu i izlaznu reprezentaciju kako bi se prilagodio zadatku. Na primjer, tijekom predtreninga BERT-u su dane dvije rečenice s nekoliko maskiranih riječi kao ulaz. Podaci prolaze kroz BERT model, a zatim idu u klasifikacijski sloj koji daje i binarnu odluku o tome slijede li rečenice jedna za drugom kao i predviđanja o tome što su zapravo maskirane riječi. Srećom, BERT-ov sloj predstavljanja unosa ne treba mijenjati jer se ulazna struktura od dvije rečenice (druga rečenica je zapravo izborna) prilagođava svim GLUE zadacima [27].

6.2.1. Matthewsov koeficijent korelacije (MCC)

Matthewsov koeficijent korelacije razvijen je za mjerenje kvalitete binarnih klasifikacija i može se modificirati tako da bude višerazredni koeficijent korelacije. Klasifikacija u dvije klase može se napraviti s četiri vjerojatnosti za svako predviđanje [28]:

- TP – istinito pozitivan (eng. true positive)
- TN – istinito negativan (eng. true negative)
- FP – lažno pozitivan (eng. false positive)
- FN – lažno negativan (eng. false negative)

Formula za MCC:

$$MCC = \left(\frac{TN * TP - FP * FN}{\sqrt{(TN + FN) * (FP + TP) * (TN + FP) * (FN + TP)}} \right)$$

Jednadžba 6.1

(izvor: https://towardsdatascience.com/matthews-correlation-coefficient-when-to-use-it-and-when-to-avoid-it-310b3c923f7e)

MCC je jedina mjera binarne klasifikacije koja generira visoku ocjenu samo ako je binarni prediktor uspio točno predvidjeti većinu instanci pozitivnih podataka i većinu instanci negativnih podataka. Ona se kreće u intervalu od −1 do +1, s ekstremnim vrijednostima –1 i +1 postignutim u slučaju savršene pogrešne klasifikacije, odnosno savršene klasifikacije, dok je MCC=0 očekivana vrijednost za klasifikator bacanja novčića [29].

7. Praktični dio zadatka

Praktični dio rada će biti podijeljen u dva dijela. U prvo dijelu će se do-trenirati BERT model za klasifikaciju sekvence, dok u drugom dijelu rada će se do-trenirati BERT model za prevođenje sa engleskog na francuski jezik.

Praktični dio rada će biti izvršen u programskom jeziku Python, dok IDE (eng. Source code editor) će biti Jupyter. Jupyter omogućava pisanje i izvršavanje python skripti u web pregledniku.

Glavna python knjižnica korištena u ovom dijelu rada je *Transformers* (izradila tvrtka Hugging Face). Ovo je knjižnica za strojno učenje i temeljena je na PyTorch, TensorFlow i JAX knjižnicama. Sadrži API-je za lako skidanje i treniranje već predtreniranih modela. Neki od modela koje ova knjižnica sadrži: ALBERT, BERT, T5,… Ostale knjižnice će biti ukratko opisane kod uporabe istih.

7.1. Treniranje predtreniranog BERT modela za klasifikaciju sekvenca

U ovom dijelu rada će se dati primjer kako istrenirati BERT model za klasifikaciju teksta. Postoje dvije vrste klasifikacije teksta:

- Binarna – Tekst će biti klasificirani kao klasa 1 ili 0. Vrijednost 1 znači da tekst pripada danoj klasifikaciji, dok vrijednost 0 znači da tekst ne pripada danoj klasifikaciji. U praksi ova vrsta klasifikacije se može vidjeti kod email-ova za provjeru „spam“ pošte.
- Multi – Sadrži više od dvije opcije po kojima se tekst može klasificirati. Te klase su proizvoljno definirane.

BERT model će biti istrenirani za prepoznavanje gramatičke ispravnosti teksta na engleskom jeziku.

```
import torch from torch.utils.data import TensorDataset, DataLoader,
RandomSampler, SequentialSampler
from torch.optim import AdamW
from keras_preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
import transformers
from transformers import BertForSequenceClassification,
get_linear_schedule_with_warmup, BertTokenizer, BertConfig, BertModel
from tqdm import tqdm, trange
import pandas as pd
import io
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.metrics import matthews_corrcoef
```

Kôd 6.1 Učitavanje potrebni knjižnica

Koristit će se grafička kartica za treniranje mreže kako bi se ubrzao taj proces. To se može postići za NVIDIA grafičke kartice tako da se koristi CUDA kao uređaja za „torch“ knjižnicu.

```
l_torch_device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
if torch.cuda.device_count() == 0:
  raise SystemError('torch: GPU device not found.')

if tf.test.gpu_device_name() != '/device:GPU:0':
  raise SystemError('tensorflow: GPU device not found.')
```

Kôd 6.2 Provjera da li je CUDA osposobljena za uporabu

Neke vrijednosti se koriste više puta programu, pa će biti definirane kao konstante. To se može vidjeti na kodu 6.3.

```
MAX_LEN = 256
BATCH_SIZE = 32
EPOCHS = 4
```

Kôd 6.3 Konstante programa

7.1.1. Učitavanje i procesiranje skupa podataka

Corpus of Linguistic Acceptability (CoLA) u punom se obliku sastoji od 10657 rečenica iz 23 lingvističke publikacije, stručno označenih za prihvatljivost (gramatičnost) od strane njihovih izvornih autora. Ovaj skup podatak podijeljen je na 9594 rečenice koje pripadaju skupovima za obuku i razvoj i 1063 rečenice koje pripadaju skupu odloženih testova.

Link za preuzimanje seta podataka: https://nyu-mll.github.io/CoLA/

Nakon što je set podataka preuzet u ubačen u radnu površinu može se učitati pomoću *read_csv* funkcije iz knjižnice *pandas.* To se može vidjeti u kodu 6.4.

```
l_training_dataset = pd.read_csv(
 "in_domain_train.tsv",
 delimiter='\t',
 header=None,
 names=['sentence_source', 'label', 'label_notes', 'sentence']
)
```

Kôd 6.4 Provjera da li je CUDA osposobljena za uporabu

Svaki uzorak u .tsv datotekama sadrži četiri stupca odvojena tabulatorima: stupac 1 je izvor rečenice (šifra), stupac 2 je oznaka (0=neprihvatljivo, 1=prihvatljivo), stupac 3 je oznaka koju je označio autor, stupac 4 je rečenica koju treba klasificirati.

Nakon što se rečenice učitaju, potrebno je na njih dodati tokene (token [CLS] kako bi se označio početak rečenice i token [SEP] kako bi se označio kraj rečenice). To se može vidjeti na kodu 6.5.

```
l_sentences = l_training_dataset.sentence.values
l_sentences = ["[CLS] " + sentence + " [SEP]" for sentence in l_sentences]
l_labels = l_training_dataset.label.values
```

Kôd 6.5 Tokeniziranje rečenica

Sljedeći korak je tokeniziranje rečenica. Kako bi smo to mogli napraviti moramo učitati već istrenirani BertTokenizer. Koristeći funkciju *tokenize*, svaka riječ u rečenici će biti pretvorena u svoj korijen. Nakon toga se primjenjuje funkcija *convert_tokens_to_ids* kako bi se svi dijelovi rečenice pretvorili u njihove indeksne brojeve u BERT vokabularu. Koristi se konstanta *MAX_LEN* kako bi se standardizirala dužina sekvence. Kod 6.6.

```
l_tokenizer = BertTokenizer.from_pretrained(
  'bert-base-uncased', do_lower_case=True
)

l_tokenized_texts = [l_tokenizer.tokenize(sentence) for sentence in
l_sentences]
l_input_ids = [l_tokenizer.convert_tokens_to_ids(x) for x in
l_tokenized_texts]

l_input_ids = pad_sequences(
  l_input_ids, maxlen=MAX_LEN, dtype="long", truncating="post", padding="post"
)
```

Kôd 6.6 BERT Tokenizer

Želimo spriječiti model da obrati pozornost na podstavljene tokene. Ideja je primijeniti masku s vrijednošću 1 za svaki token, koju će pratiti 0 za ispunu. To se može vidjeti u kodu 6.7.

```
l_attention_masks = []

for seq in l_input_ids:
 l_seq_mask = [float(i>0) for i in seq]
 l_attention_masks.append(l_seq_mask)
```

Kôd 6.7 Postavljanje maske za pozornost

Sljedeći korak je odvajanje podataka na skup za trening i evaluaciju.

```
l_train_inputs, l_validation_inputs, l_train_labels, l_validation_labels =
train_test_split(
 l_input_ids, l_labels, random_state=2018, test_size=0.1
)

l_train_masks, l_validation_masks, l_, l_ = train_test_split(
```

```
 l_attention_masks, l_input_ids, random_state=2018, test_size=0.1
)
```

Kôd 6.8 Podjela podataka na skup za trening i evaluaciju

Kako bi BERT model mogao koristiti podatke, moramo ih pretvoriti u *torch tensors* tip podataka. Taj postupak je prikazan na kodu 6.9.

```
l_train_inputs = torch.tensor(l_train_inputs)
l_validation_inputs = torch.tensor(l_validation_inputs)
l_train_labels = torch.tensor(l_train_labels)
l_validation_labels = torch.tensor(l_validation_labels)
l_train_masks = torch.tensor(l_train_masks)
l_validation_masks = torch.tensor(l_validation_masks)
```

Kôd 6.9 Formatiranje

Sada dok su podaci spremni potrebno je odabrati veličinu serije (konstanta *BATCH_SIZE*) i stvoriti *Iterator*. *Iterator* je način izbjegavanja petlje koja bi učitavala sve podatke u memoriju. *Iterator*, zajedno s *torch DataLoaderom*, može grupno trenirati velike skupove podataka bez rušenja memorije stroja. Kod 6.10.

```
l_train_data = TensorDataset(l_train_inputs, l_train_masks, l_train_labels)
l_train_sampler = RandomSampler(l_train_data)

l_train_dataloader = DataLoader(
 l_train_data, sampler=l_train_sampler, batch_size=BATCH_SIZE
)
l_validation_data = TensorDataset(
 l_validation_inputs, l_validation_masks, l_validation_labels
)
l_validation_sampler = SequentialSampler(l_validation_data)
l_validation_dataloader = DataLoader(
 l_validation_data, sampler=l_validation_sampler, batch_size=BATCH_SIZE
)
```

Kôd 6.10 Stvaranje Iteratora sa torch DataLoader-om

7.1.2. Inicijalizacija „bert base uncased“ BERT modela i optimizatora

BERT base model (uncased) je unaprijed istrenirani model na engleskom jeziku koristeći cilj modeliranja maskiranog jezika (MLM). Ovaj model ne prepoznaje razliku između malog i velikog slova.

Neobrađeni model može se koristiti ili za modeliranje maskiranog jezika ili za predviđanje sljedeće rečenice, ali uglavnom je namijenjen za fino podešavanje. Ovaj model prvenstveno je usmjeren na fino podešavanje zadataka koji koriste cijelu rečenicu (potencijalno maskiranu) za donošenje odluka, kao što je klasifikacija slijeda, klasifikacija tokena ili odgovaranje na pitanja.

```
l_bert_config = BertConfig()
l_model = BertModel(l_bert_config)
l_model = BertForSequenceClassification.from_pretrained(
  "bert-base-uncased", num_labels=2
)
l_model.cuda()
```

Kôd 6.11 Inicijalizacija BERT modela

Sljedeći korak je pripremiti parametre modela za petlju treniranja. Fino podešavanje modela počinje inicijalizacijom unaprijed istreniranih vrijednosti parametara modela. Parametri optimizatora uključuju stopu opadanja težine (eng. weight decay rate) kako bi se izbjeglo prekomjerno prilagođavanje, a neki parametri se filtriraju. Kod 6.12.

```
l_param_optimizer = list(l_model.named_parameters())
l_no_decay = ['bias', 'LayerNorm.weight']
l_optimizer_grouped_parameters = [
 {
  'params': [p for n, p in l_param_optimizer if not any(nd in n for nd in
l_no_decay)],
  'weight_decay_rate': 0.1
 },
 {
  'params': [p for n, p in l_param_optimizer if any(nd in n for nd in
l_no_decay)],
  'weight_decay_rate': 0.0
 }
]
```

Kôd 6.12 Parrametri optimizatora

Inicijalizacija optimizatora se postiže zvanjem funkcije *AdamW*. To se može vidjeti u kodu 6.13. Stopa učenja (*lr*) treba biti postavljena na vrlo malu vrijednost rano u fazi optimizacije i postupno povećavati nakon određenog broja ponavljanja. Time se izbjegavaju veliki gradijenti i prekoračenje ciljeva optimizacije.

```
l_optimizer = AdamW(l_optimizer_grouped_parameters, lr = 2e-5, eps = 1e-8)
```

Kôd 6.13 Inicijalizacija optimizatora

Na kraju se treba odrediti broj stopa za treniranje i inicijalizirati planer za stopu učenje. To se može vidjeti u kodu 6.14. U tom kodu se još može vidjeti inicijalizacija varijable *l_train_loss_set* kao niz (eng. array) u koji će se spremati vrijednosti *training loss* za svaki skup.

```
l_total_steps = len(l_train_dataloader) * EPOCHS

l_scheduler = get_linear_schedule_with_warmup(
 l_optimizer, num_warmup_steps = 0, num_training_steps = l_total_steps
)
```

```
l_train_loss_set = []
```

Kôd 6.14 Inicijalizacija planera

7.1.3. Petlja za treniranje

Prije početka treninga, potrebno je definirati funkciju za mjerenje točnosti kod usporedbi predviđanja (*preds*) sa očekivanim rezultatom (*labels*). Kod 6.15

```
def flat_accuracy(preds, labels):
 f_pred_flat = np.argmax(preds, axis=1).flatten()
 f_labels_flat = labels.flatten()
 return np.sum(f_pred_flat == f_labels_flat) / len(f_labels_flat)
```

Kôd 6.15 Funkcija za mjerenje točnosti

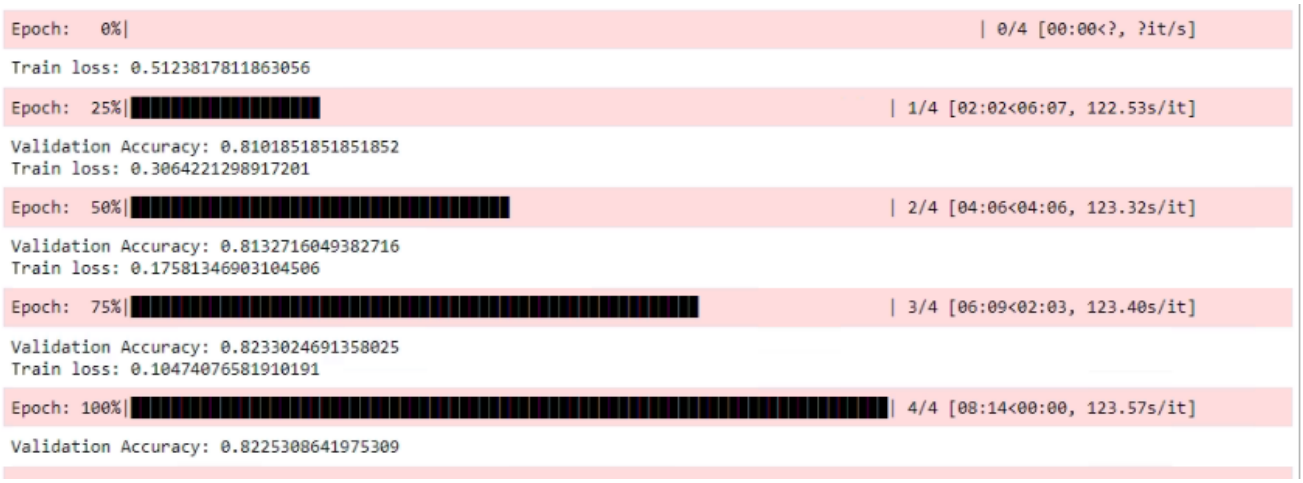
Petlja treninga slijedi standardne procese učenja. Broj epoha postavljen je na 4, a program će pratiti gubitke i točnost. Proces treninga se mjeri i ocjenjuje.

```
for l_ in trange(EPOCHS, desc="Epoch"):
 l_model.train()

 l_tr_loss = 0
 l_nb_tr_examples = 0
 l_nb_tr_steps = 0

 for step, batch in enumerate(l_train_dataloader):

  l_batch = tuple(t.to(l_torch_device) for t in batch)
  b_input_ids, b_input_mask, b_labels = l_batch

  l_optimizer.zero_grad()

  l_outputs = l_model(
   b_input_ids,
   token_type_ids=None,
   attention_mask=b_input_mask,
   labels=b_labels
  )

  l_loss = l_outputs['loss']
  l_train_loss_set.append(l_loss.item())

  l_loss.backward()
  l_optimizer.step()
  l_scheduler.step()

  l_tr_loss += l_loss.item()
  l_nb_tr_examples += b_input_ids.size(0)
  l_nb_tr_steps += 1

 print("Train loss: {}".format(l_tr_loss/l_nb_tr_steps))

 l_model.eval()
```

```
l_eval_loss, l_eval_accuracy = 0, 0
l_nb_eval_steps, l_nb_eval_examples = 0, 0

for batch in l_validation_dataloader:
 l_batch = tuple(t.to(l_torch_device) for t in batch)
 b_input_ids, b_input_mask, b_labels = l_batch

 with torch.no_grad():
  l_logits = l_model(
   b_input_ids, token_type_ids=None, attention_mask=b_input_mask
  )

 l_logits = l_logits['logits'].detach().cpu().numpy()
 l_label_ids = b_labels.to('cpu').numpy()

 l_tmp_eval_accuracy = flat_accuracy(l_logits, l_label_ids)

 l_eval_accuracy += l_tmp_eval_accuracy
 l_nb_eval_steps += 1

print("Validation Accuracy: {}".format(l_eval_accuracy/l_nb_eval_steps))
```

Kôd 6.16 Trening petlja

Na kraju svake epohe će biti ispisana informacija o treningu, slika 6.1.



Slika 6.1 Epohe treninga mreže

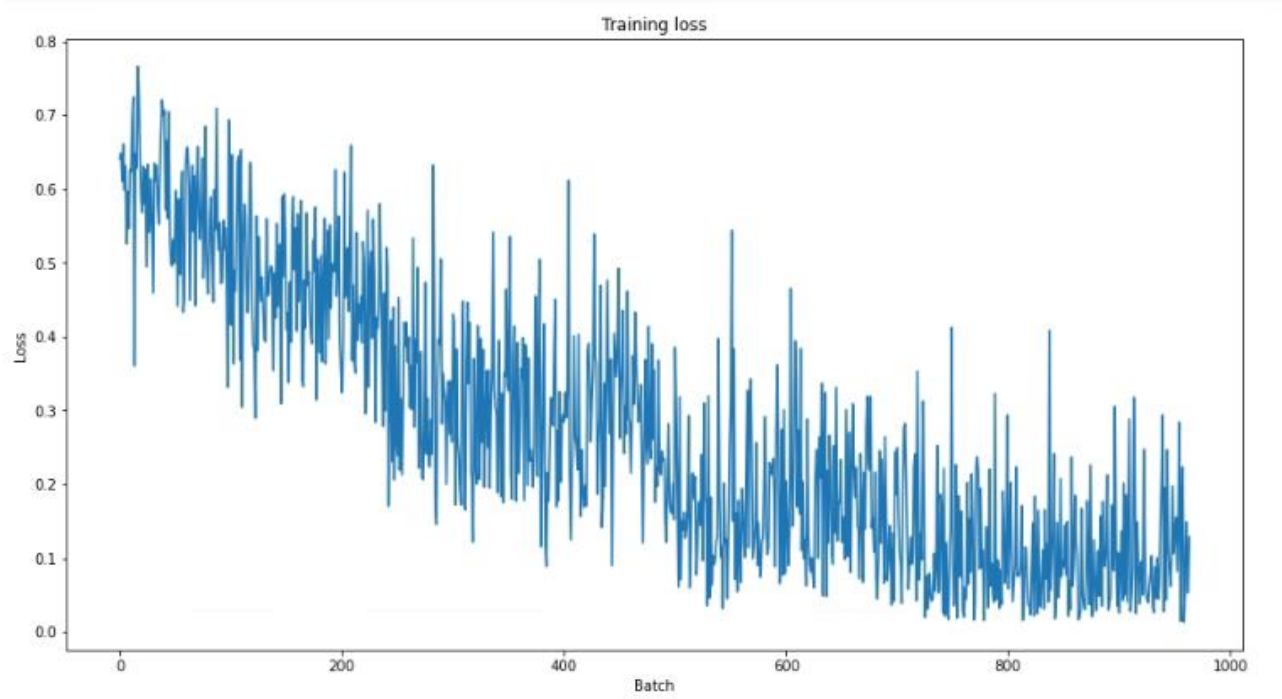
Vrijednosti gubitak (eng. training loss) su bile spremljene u varijablu *l_train_loss_set* i mogu biti iscrtane u graf, kod 6.17.

```
plt.figure(figsize=(15,8))
plt.title("Training loss")

plt.xlabel("Batch")
plt.ylabel("Loss")

plt.plot(l_train_loss_set)
```

```
plt.show()
```

Kôd 6.17 Iscrtavanje grafa za training loss

Izlaz koda 6.17 će biti graf koji pokazuje da je proces treninga prošao dobro.



Slika 6.2 Epohe treninga mreže

Na slici 6.2 se može vidjeti da je mreža uspješno dotrenirana. U prvoj epohi prosječni *loss* je 0.51, a kod zadnje epohe je 0.11. Potrebno je napomenuti da najveći napredak postignut kroz treniranje u prve dvije epohe.

7.1.4. Petlja za evaluaciju

Prije nego što se model može evaluirati, mora se učitati skup podataka za testiranje i obraditi se na isti način kao i u poglavlju 7.1.1.

```
l_prediction_and_evaluation_dataset = pd.read_csv(
 "out_of_domain_dev.tsv",
 delimiter='\t',
 header=None,
 names=['sentence_source', 'label', 'label_notes', 'sentence']
)

l_prediction_and_evaluation_sentences                                    =
l_prediction_and_evaluation_dataset.sentence.values
l_prediction_and_evaluation_sentences = [
```

```
 "[CLS] " + i + " [SEP]" for i in l_prediction_and_evaluation_sentences
]
l_prediction_and_evaluation_labels                                    =
l_prediction_and_evaluation_dataset.label.values
l_prediction_and_evaluation_tokenized_texts = [
 l_tokenizer.tokenize(i) for i in l_prediction_and_evaluation_sentences
]

l_prediction_and_evaluation_input_ids = [
 l_tokenizer.convert_tokens_to_ids(x)          for          x          in
l_prediction_and_evaluation_tokenized_texts
]

l_prediction_and_evaluation_input_ids = pad_sequences(
 l_prediction_and_evaluation_input_ids,
 maxlen=MAX_LEN,
 dtype="long",
 truncating="post",
 padding="post"
)

l_attention_masks = []
for seq in l_prediction_and_evaluation_input_ids:
 l_seq_mask = [float(i>0) for i in seq]
 l_attention_masks.append(l_seq_mask)

 l_prediction_inputs = torch.tensor(l_prediction_and_evaluation_input_ids)
 l_prediction_masks = torch.tensor(l_attention_masks)
 l_prediction_labels = torch.tensor(l_prediction_and_evaluation_labels)

l_prediction_data = TensorDataset(
 l_prediction_inputs, l_prediction_masks, l_prediction_labels
)
l_prediction_sampler = SequentialSampler(l_prediction_data)
l_prediction_dataloader = DataLoader(
 l_prediction_data, sampler=l_prediction_sampler, batch_size=BATCH_SIZE
)
```

Kôd 6.18 Priprema podataka za petlju evaluacije

Prije ulazaka u petlju model se stavlja u evaluacijski način rada zvanjem funkcije *eval()*. Inicijaliziraju se varijable *l_predictions* i *l_true_labels* kako bi se u njih mogle spremiti vrijednosti predviđanja i očekivanog rezultata. Program zatim pokreće skupna predviđanja pomoću *dataloader*-a, kod 6.19.

```
l_model.eval()

l_predictions = []
l_true_labels = []

for batch in l_prediction_dataloader:
 l_batch = tuple(t.to(l_torch_device) for t in batch)
 b_input_ids, b_input_mask, b_labels = l_batch

 with torch.no_grad():
  l_logits = l_model(
   b_input_ids, token_type_ids=None,attention_mask=b_input_mask
```

```
  )

 l_logits = l_logits['logits'].detach().cpu().numpy()
 l_label_ids = b_labels.to('cpu').numpy()
 l_predictions.append(l_logits)
 l_true_labels.append(l_label_ids)
```

Kôd 6.19 Petlja za evaluaciju

Sada kada imamo predviđanja i očekivane rezultate, možemo izračunati Matthewsov koeficijent korelacije nad cijelim skupom podataka ili pojedinačnim skupovima. Kod 6.20 pokazuje kako izračunati Matthewsov koeficijent korelacije nad pojedinačnim skupovima.

```
l_matthews_set = []

for i in range(len(l_true_labels)):
 l_matthews = matthews_corrcoef(
  l_true_labels[i],
  np.argmax(l_predictions[i], axis=1).flatten()
 )
 l_matthews_set.append(l_matthews)

print(l_matthews_set)
```

Kôd 6.20 Evaluacija pojedinačnih setova s Matthewsovim koeficijentom korelacije

Sada kada smo ocijenili pojedinačne skupove, možemo izračunati Matthewsov koeficijent korelacije nad cijelim skupom podataka, kod 6.21.

```
l_flat_predictions = [item for sublist in l_predictions for item in sublist]
l_flat_predictions = np.argmax(l_flat_predictions, axis=1).flatten()
l_flat_true_labels = [item for sublist in l_true_labels for item in sublist]
l_eval = matthews_corrcoef(l_flat_true_labels, l_flat_predictions)

print("="*50)
print('Evaluation: ', l_eval)
```

Kôd 6.21 Evaluacija cijelog skupa podataka s Matthewsovim koeficijentom korelacije

Izlaz (0, 550166) potvrđuje da je Matthewsov koeficijent korelacije pozitivan, što pokazuje da postoji korelacija za ovaj model i skup podataka.

Na kraju treniranja BERT modela, potrebno ga je spremiti lokalno kako bi se mogao dalje koristiti, kod 6.22.

```
l_model_path = "bert_Sequence"
l_model.save_pretrained(l_model_path)
l_tokenizer.save_pretrained(l_model_path)
```

Kôd 6.22 Spremanje istreniranog modela i tokenizatora

7.1.5. Korištenje istreniranog modela

Prije nego što možemo bilo što raditi s istreniranim modelom, moramo ga učitati i postaviti da računa s grafičkom karticom, kod 6.23.

```
import torch
from transformers import BertForSequenceClassification, BertTokenizer

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
if torch.cuda.device_count() == 0:
 raise SystemError('torch: GPU device not found.')

model_path = "bert_Sequence"

tokenizer = BertTokenizer.from_pretrained(
 model_path, do_lower_case=True, local_files_only=True
)
model = BertForSequenceClassification.from_pretrained(
 model_path, num_labels=2, local_files_only=True
)
model.to(device)
```

Kôd 6.23 Učitavanje istreniranog modela

Sljedeći korak je definirati funkciju koja će primati vrijednost (tipa *string*) i provest je kroz istrenirani model. Istrenirani model će tada odlučiti da li je dani tekst gramatički ispravan ili ne. U slučaju da je ispravan dat će vrijednost „*LABEL_1*“. Funkcija je složena tako da u slučaju da model odredi da je tekst ispravan, vraća vrijednost *True*, a u slučaju da mode odredi da je tekst neispravan vraća vrijednost *False*. Kod 6.24.

```
def examine_text(text):
 inputs = tokenizer(text, return_tensors="pt").to(device)

 with torch.no_grad():
  logits = model(**inputs).logits

 if model.config.id2label[logits.argmax().item()] == "LABEL_1":
  return True

 return False
```

Kôd 6.24 Funkcija za ispitivanje gramatičke točnosti

Potrebno je napomenuti da će model neispravno klasificirati tekst koji se sastoji od nasumično generiranih slova (npr. „*fg atjg sfs ttca*.“) ili rečenicu od tri ili manje riječi.

U prvom skupu primjera ću prvo dati gramatički ispravnu rečenicu i zatim tu istu rečenicu modificiranu da bude gramatički neispravna. Vrijednosti u komentaru su očekivani rezultat (*1 = True, 0 = False*). Pogledajte kod 6.25.

```
print("="*50)
print(examine_text(
 "They claimed they had settled on something, but it wasn't clear what they
had settled on."
)) #1
print(examine_text(
 "They claimed thay had setled on something, but it wasn't clir what they hed
settled on."
)) #0

print("="*50)
print(examine_text("They should pen the letter quickly.")) #1
print(examine_text("They shuld pen the leter quickly.")) #0

print("="*50)
print(examine_text("So fast did he run that nobody could catch him.")) #1
print(examine_text("So fast diid he run that nobody could katch him.")) #0
```

Kôd 6.25 Prvi skup primjera

Kao što se vidi na slici 6.3 model je ispravno klasificirao rečenice. U kraćim rečenicama je dovoljno jedna ili dvije gramatičke neispravnosti da bi bila klasificirana kao neispravna, dok je kod duljih rečenica potrebno više neispravnosti.

```
==================================================
True
False
==================================================
True
False
==================================================
True
False
```

Slika 6.3 Ispis rezultata prvog skupa primjera

U drugom setu primjera su dane gramatički točne i gramatički netočne rečenice, kod 6.26.

```
print("="*50)
print(examine_text(
 "The book that crime was declining surprised many people."
)) #0

print("="*50)
print(examine_text("She knew French for Tom."))  #0

print("="*50)
print(examine_text("Ted was bitten by the spider.")) #1

print("="*50)
print(examine_text("No you had book.")) #0
```

```
print("="*50)
print(examine_text("What did you leave before they did?")) #0

print("="*50)
print(examine_text("So fast did he run that nobody could catch him.")) #1

print("="*50)
print(examine_text("The boat seen down the river sank.")) #1
```

Kôd 6.26 Drugi skup primjera

Na slici 6.4 se može vidjeti da je model dobro klasificirao šest od sedam primjera. Petu rečenicu je klasificirao kao ispravnu, dok je u stvarnosti neispravna.

```
==================================================
False
==================================================
False
==================================================
True
==================================================
False
==================================================
True
==================================================
True
==================================================
True
```

Slika 6.4 Ispis rezultata drugog skupa primjera

7.2. Treniranje predtreniranog BERT modela za prevođenje jezika

U ovom dijelu rada će se dati primjer kako istrenirati BERT model za prevođenje jezika. Model će biti trenirani za prevođenje sa engleskog na francuski jezik. Kao predtrenirani model koristit će se *BERT t5-small*.

BERT t5-small je model kodera-dekodera unaprijed istrenirani za više zadaćnu mješavinu nenadziranih i nadziranih zadataka i za koji se svaki zadatak pretvara u format teksta u tekst. *T5* dobro radi na raznim zadacima tako što ispred ulaza koji odgovara svakom zadatku dodaje drugačiji prefiks.

T5 je trenirani na setovima podataka: „c4“ [30], „CoLA Warstadt i dr., 2018“ [31], „SST-2 Socher i sur., 2013“ [32], i mnogo drugih.

7.2.1. Treniranje BERT modela sa trenerom

Prije treniranja je potrebno učitati sve knjižnice koje će se koristiti. Zatim se treba provjeriti sposobnost korištenja grafičke kartice za treniranje i definirati neke konstante kao što je prefiks (kako bi model mogao znači što raditi, u ovom slučaju dati mu do znanje da se radi o prijevodu sa engleskog na francuski jezik). Nakon toga se učita skup podataka i inicijalizira model, kod 6.27.

```
import os
import wandb
import torch
from datasets import *
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM,
DataCollatorForSeq2Seq, Seq2SeqTrainingArguments, Seq2SeqTrainer

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
if torch.cuda.device_count() == 0:
 raise SystemError('torch: GPU device not found.')

model_path = "bert-Translation"
source_lang = "en"
target_lang = "fr"
prefix = "translate English to French: "

dataset = load_dataset("opus_books", "en-fr")
dataset = dataset["train"].train_test_split(test_size=0.18, shuffle=False)

tokenizer = AutoTokenizer.from_pretrained("t5-small")
model = AutoModelForSeq2SeqLM.from_pretrained("t5-small").to(device)
data_collator = DataCollatorForSeq2Seq(tokenizer=tokenizer, model=model)
```

Kôd 6.27 Inicijalizacija varijabli i BERT modela

Sljedeći korak je definirati funkciju koja će tokenizirati skup podataka. Pogledajte kod 6.28.

```
def preprocess_function(examples):
 inputs = [
  prefix + example[source_lang] for example in examples["translation"]
 ]
 targets = [example[target_lang] for example in examples["translation"]]
 model_inputs = tokenizer(inputs, max_length=512, truncation=True)

 with tokenizer.as_target_tokenizer():
 labels = tokenizer(targets, max_length=512, truncation=True)

 model_inputs["labels"] = labels["input_ids"]
 return model_inputs


tokenized_dataset = dataset.map(preprocess_function, batched=True)
```

Kôd 6.28 Procesiranje skupa podataka

Kod inicijalizacije trenera se definiraju postavke kao što je broj treniranih epoha, direktoriji za spremanje modela, stopa učenja, veličina serije za treniranje i evaluaciju (*per_device_train_batch_size* i *per_device_eval_batch_size* – što je veća vrijednost ovih parametara, to će se mreža brže trenirati, ali će i korištenje memorije grafičke kartice biti veće), kod 6.29.

```
training_args = Seq2SeqTrainingArguments(
 output_dir=model_path,
 evaluation_strategy="steps",
 overwrite_output_dir=True,
 do_train=True,
 do_eval=True,
 learning_rate=2e-5,
 weight_decay=0.01,
 per_device_train_batch_size=12,
 per_device_eval_batch_size=12,
 num_train_epochs=4,
 eval_steps=3000,
 save_steps=3000,
 fp16=True,
 report_to="wandb"
)

trainer = Seq2SeqTrainer(
 model=model,
 args=training_args,
 train_dataset=tokenized_dataset["train"],
 eval_dataset=tokenized_dataset["test"],
 tokenizer=tokenizer,
 data_collator=data_collator
)
```

Kôd 6.29 Inicijalizacija trenera

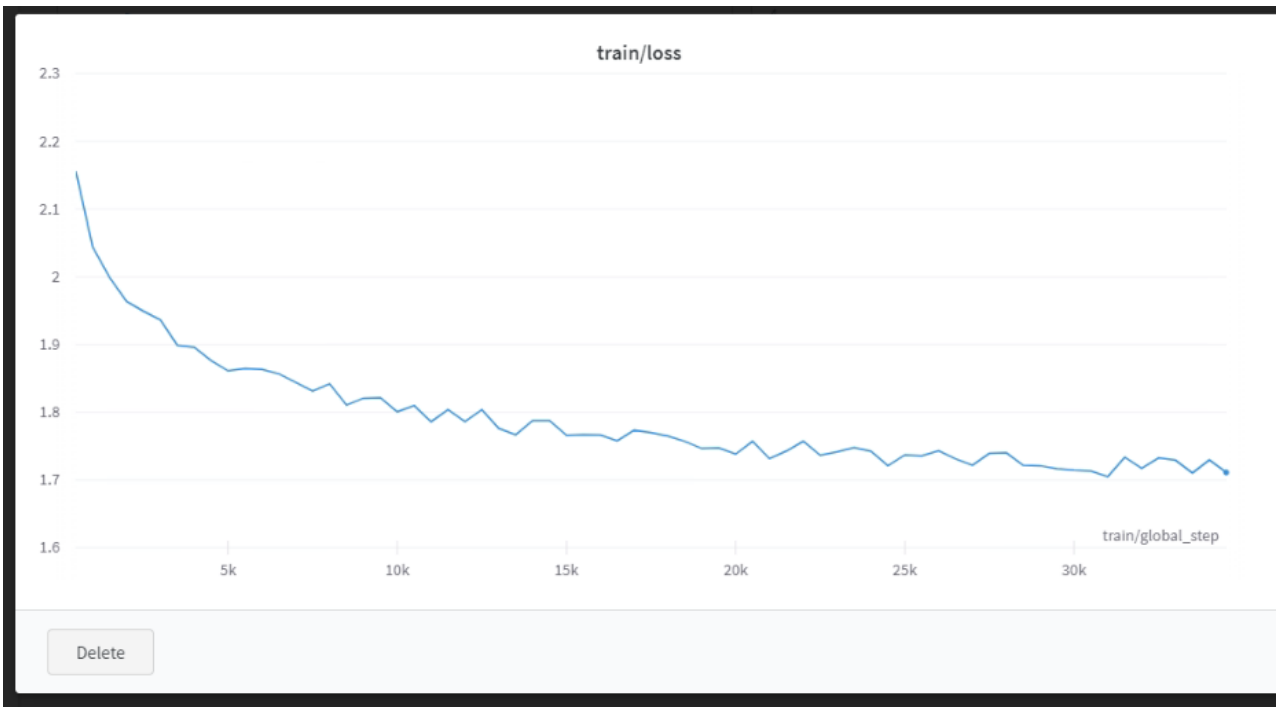
Sljedeći korak je samo pokrenuti trening zvanjem funkcije *train()*, kod 6.30.

```
print(trainer.evaluate())
trainer.train()
print(trainer.evaluate())
```

Kôd 6.30 Treniranje modela

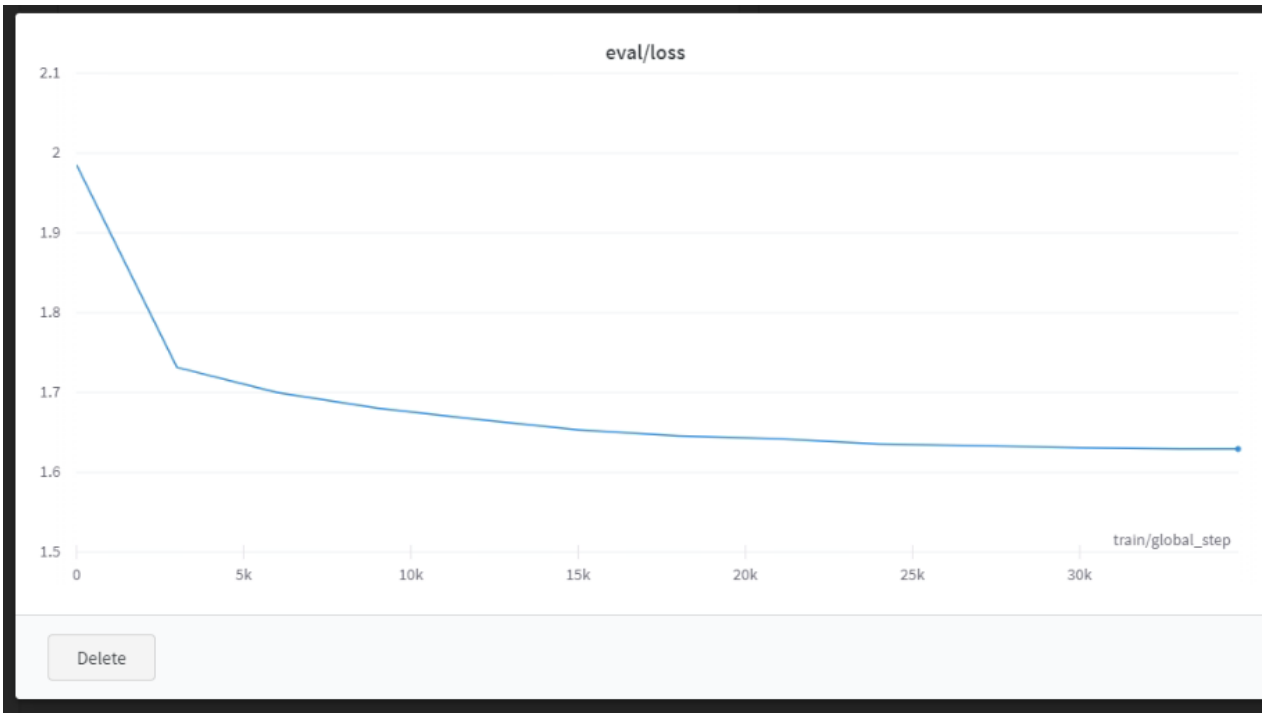
Evaluacija prije treninga je pokazala da je mjera gubitka testnog skupa (evaluation loss) 1,984848.

Na slici 6.5 se može vidjeti da je mreža uspješno istrenirana. Mreža je počela sa nekih 2,2 gubitka skupa za trening i unaprijedila ga je na 1,715.



Slika 6.5 Graf za training loss

Na slici 6.6 se može vidjeti da je mreža trenirana na dobar način zato što je *evaluation loss* konstantno imao manju vrijednost od *training loss*.



Slika 6.6 Graf za evaluation loss

7.2.2. Evaluacija BERT modela za prevođenje jezika koristeći BLEU ocjenu

Prije treniranja je potrebno uvesti sve knjižnice koje će se koristiti. Zatim će se osposobiti *numpy* ponašanje i provjeriti sposobnost korištenja grafičke kartice za evaluaciju. Nakon toga se učita skup podataka. Za evaluaciju će se odvojiti 8400 primjera iz skup, kod 6.31.

```
import os
import wandb
import numpy as np
from tensorflow.python.ops.numpy_ops import np_config
import torch
from datasets import *
from     transformers     import     AutoTokenizer,     AutoModelForSeq2SeqLM,
DataCollatorForSeq2Seq, Seq2SeqTrainingArguments, Seq2SeqTrainer
import evaluate

np_config.enable_numpy_behavior()

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
if torch.cuda.device_count() == 0:
 raise SystemError('torch: GPU device not found.')

model_path = "bert-Translation"
model_checkpoint = os.path.join(model_path, "checkpoint-33000")
source_lang = "en"
target_lang = "fr"
prefix = "translate English to French: "

dataset = load_dataset("opus_books", "en-fr")
dataset = dataset["train"].train_test_split(test_size=0.18, shuffle=False)
dataset = dataset["test"].shuffle().select(range(8400))

metric = evaluate.load("sacrebleu")
tokenizer = AutoTokenizer.from_pretrained("t5-small")
model = None
data_collator = None
```

Kôd 6.31 Inicijalizacija skripte za evaluaciju

Za tokenizaciju podataka koristit će se ista funkcija kao u kodu 6.28.

Sljedeći korak je definirati funkciju koja će napraviti BLEU evaluaciju na trenutno učitanim BERT modelu. Funkcija će vratiti BLEU vrijednost, kod 6.32.

Potrebno je napomenuti da zbog nekompatibilnosti *Huggingface* knjižnice sa verzijom *Numpy* knjižnice nije bilo moguće tokenizirani skup podataka poslati u model. Iz tog razloga se rečenica opet tokenizira kao *torch*.

```
def calc_bleu():
 all_preds = []
 all_labels = []

 batch_size = 4
 tf_dataset = tokenized_dataset.to_tf_dataset(
```

```
  columns=["input_ids", "attention_mask", "labels"],
  collate_fn=data_collator, shuffle=False, batch_size=4
 )

 i = 0

 for batch in tf_dataset:
  for x in range(batch['labels'].shape[0]):
   sentence = prefix + tokenized_dataset[(i*batch_size) +
x]["translation"][source_lang]

   l_input = tokenizer(sentence, return_tensors="pt", max_length=512,
padding=True).to(device)

   predictions = model.generate(
    input_ids=l_input["input_ids"],
    attention_mask=l_input["attention_mask"], max_length = 512,
   )

   decoded_preds = tokenizer.batch_decode(predictions,
skip_special_tokens=True)
   decoded_preds = [pred.strip() for pred in decoded_preds]
   all_preds.extend(decoded_preds)

   labels = batch["labels"].numpy()
   labels = np.where(labels != -100, labels, tokenizer.pad_token_id)

   decoded_labels = tokenizer.batch_decode(labels, skip_special_tokens=True)
   decoded_labels = [[label.strip()] for label in decoded_labels]

   for label in decoded_labels:
    all_labels.append(label[0])

   i = i + 1

   print('Step: %d' % i, end='\r')

 result = metric.compute(predictions=all_preds, references=all_labels)

 return result["score"]
```

Kôd 6.32 Funkcija za izračunavanje BLEU ocjene BERT modela

Prvo se učitava „*t5-small*“ model i evaluira se. Rezultat te evaluacije je bio 15,89 (BLEU: 0,1589). Zatim se učitava model istrenirani u prošlom poglavlju (stopa 33000) i evaluira se. Rezultat te evaluacije je bio 19,83 (BLEU: 0,1983). Pogledajte kod 6.33. Potrebno je napomenuti da su ocjene relativno niske zato što korišteni set podataka koristi arhaične riječi i u nekim slučajevima nestandardnu strukturu rečenica.

```
model = AutoModelForSeq2SeqLM.from_pretrained("t5-small").to(device)
data_collator = DataCollatorForSeq2Seq(tokenizer=tokenizer, model=model)

print("Base: ", calc_bleu())
#izlaz: 15.8904656906288

model = AutoModelForSeq2SeqLM.from_pretrained(
```

```
 model_checkpoint, local_files_only=True
).to(device)
data_collator = DataCollatorForSeq2Seq(tokenizer=tokenizer, model=model)

print("Trained: ", calc_bleu())
#izlaz: 19.825461462334935
```

Kôd 6.33 Ealuacija originalnog i istreniranog modela

Na rezultatu evaluacije se može vidjeti da je treniranje unaprijedilo sposobnost mreže za prevođenje sa engleskog na francuski jezik.

7.2.3. Korištenje BERT modela za prevođenje jezika

Kako bi se lako mogao koristiti istrenirani model, definira se funkcija koja prima (tipa *string*) i provodi se kroz model. Funkcija će vratiti unesenu rečenicu prevedenu na francuski jezik. Pogledajte kod 6.34.

```
import os
import torch
import evaluate
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
if torch.cuda.device_count() == 0:
 raise SystemError('torch: GPU device not found.')

model_checkpoint = os.path.join("bert-Translation", "checkpoint-33000")
prefix = "translate English to French: "
metric = evaluate.load("sacrebleu")
tokenizer = AutoTokenizer.from_pretrained("t5-small")
model        =        AutoModelForSeq2SeqLM.from_pretrained(model_checkpoint,
local_files_only=True).to(device)

def translate(text):
 l_input = tokenizer(
  prefix + text, return_tensors="pt", max_length=512, truncation=True
 ).to(device)

 outputs = model.generate(
  input_ids=l_input["input_ids"],
  attention_mask=l_input["attention_mask"],
  max_length = 512
 )

 return tokenizer.batch_decode(outputs, skip_special_tokens=True)[0]
```

Kôd 6.34 Funkcija za prevođenje s engleskog na francuski jezik

Sljedeći korak je definirati funkciju koja će primiti tekst koji će biti preveden i referencu koja će biti očekivani rezultat prijevoda. Funkcija će tada ispisati prijevod modela i ispisati BLEU ocjenu. Pogledajte kod 6.35.

```
def bleu_eval(pred="", ref=""):
 result = metric.compute(predictions=[pred], references=[ref])

 return result["score"]


def translate_and_bleu(text, ref):
 l_pred = translate(text)
 print("Translation: ", l_pred)
 print("BLEU: ", bleu_eval(pred=l_pred, ref=ref))
 print("="*125)
```

Kôd 6.35 Funkcija za prevođenje i BLEU ocjenjivanje prijevoda

Kao prvi skup primjera odlučio sam uzeti par rečenica iz skupa podataka za treniranje.

```
translate_and_bleu(
 "My father, whom I used to call M. Seurel as did other pupils, was head of
the Middle School and also of the Higher Elementary classes where pupils worked
for the preliminary teacher's examination.",
 "Mon père, que j’appelais M. Seurel, comme les autres élèves, y dirigeait à
la fois le Cours Supérieur, où l’on préparait le brevet d’instituteur, et le
Cours Moyen."
)

translate_and_bleu(
 "Thus to-day I picture our arrival.",
 "C’est ainsi, du moins, que j’imagine aujourd’hui notre arrivée."
)

translate_and_bleu(
 "Yet we had already been ten years in that district when Meaulnes arrived.",
 "Nous étions pourtant depuis dix ans dans ce pays lorsque Meaulnes arriva."
)

translate_and_bleu(
 "No one came to let in the unknown visitor.",
 "Personne ne venait ouvrir à la visiteuse inconnue."
)
```

Kôd 6.36 Prvi skup primjera

```
Translation:  Mon père, que j'appelais M. Seurel comme d'autres élèves, était c
hef de l'école moyenne et des classes supérieures élémentaires où les élèves tr
availlaient pour l'examen préliminaire de l'enseignant.
BLEU:  11.90
==================================================
Translation:  Ainsi, aujourd’hui je me fassai notre arrivée.
BLEU:  14.45
==================================================
Translation:  Pourtant, nous avions déjà dix ans dans ce quartier, lorsque Meau
lnes arriva.
BLEU:  32.90
==================================================
Translation:  Personne ne venait laisser l’inconnu visiteur.
BLEU:  19.74
==================================================
```

Slika 6.7 Izlaz prvog skupa primjera

U drugom setu sam odlučio staviti par rečenica koje se često koristi u svakodnevnici.

```
translate_and_bleu(
 "I do not speak French.",
 "Je ne parle pas français."
)

translate_and_bleu(
 "Where is a good restaurant/a good café?",
 "Où est un bon restaurant/un bon café?"
)

translate_and_bleu(
 "Hello. May I have your passport and your ticket, please?",
 "Bonjour. Puis-je avoir votre passeport et votre billet, s'il-vous-plaît?"
)

translate_and_bleu(
 "Excuse me, where is the nearest metro station?",
 "Excuse-moi, où est la station de métro la plus proche?"
)
```

Kôd 6.37 Drugi skup primjera

```
Translation:  Je ne parle pas français.
BLEU:  100.00
================================================
Translation:  Où est un bon restaurant ou un bon café?
BLEU:  65.80
================================================
Translation:  Bonjour, puis-je avoir votre passeport et votre billet, veuillez-
je?
BLEU:  54.91
================================================
Translation:  Excusez-moi, où est la gare de métro la plus proche?
BLEU:  63.40
================================================
```

Slika 6.8 Izlaz drugog skupa primjera

8. Zaključak

Arhitektura transformer neuronske mreže je pristupila problemu modeliranja sekvence tako da umjesto dotadašnjih popularnih pristupa kao što su konvolucijska neuronska mreža ili povratna neuronska mreža uvela simetrični dizajn kod standardizacije dimenzija kodera i dekodera, pozicijsko kodiranje i paralelizaciju slojeva.

Pojava transformera je označila početak nove generacije modela umjetne inteligencije spremnih za korištenje. Na primjer, Hugging Face i Google Brain olakšavaju implementaciju umjetne inteligencije s nekoliko redaka koda.

Originalna transformerova arhitektura pruža osnovu za mnoge druge inovativne varijacije koje sa sobom nose još efikasniji i specijaliziraniji način modeliranja jezika.

Fino podešavanje pred-reniranog modela zahtijeva manje strojnih resursa nego treniranje od nule. Fino podešeni modeli mogu obavljati razne zadatke. BERT dokazuje da možemo unaprijed istrenirati model na dva ili više zadatka, što je samo po sebi izvanredno. Ali izrada višezadaćnog fino podešenog modela temeljenog na predtreniranim parametrima BERT modela je izvanredna.

U Varaždinu, _____ _____

Potpis studenta

Sveučilište Sjever

IZJAVA O AUTORSTVU I SUGLASNOST ZA JAVNU OBJAVU

Završni/diplomski rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, DARKO PETROVIĆ *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom TRANSFORMER NEURONSKE MREŽE ZA OBRADU PRIRODNOG JEZIKA *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)
Darko Petrović
(vlastoručni potpis)

Sukladno Zakonu o znanstvenoj djelatnost i visokom obrazovanju završne/diplomske radove sveučilišta su dužna trajno objaviti na javnoj internetskoj bazi sveučilišne knjižnice u sastavu sveučilišta te kopirati u javnu internetsku bazu završnih/diplomskih radova Nacionalne i sveučilišne knjižnice. Završni radovi istovrsnih umjetničkih studija koji se realiziraju kroz umjetnička ostvarenja objavljuju se na odgovarajući način.

Ja, DARKO PETROVIĆ *(ime i prezime)* neopozivo izjavljujem da sam suglasan/na s javnom objavom završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom TRANSFORMER NEURONSKE MREŽE ZA OBRADU PRIRODNOG JEZIKA *(upisati naslov)* čiji sam autor/ica.

Student/ica:
(upisati ime i prezime)
Darko Petrović
(vlastoručni potpis)

9. Literatura

[1] https://www.britannica.com/topic/language dostupno 7.2.2022.

[2] https://medium.com/analytics-vidhya/natural-language-processing-from-basics-to-using-rnn-and-lstm-ef6779e4ae66 dostupno 7.2.2022.

[3] https://medium.com/sciforce/nlp-vs-nlu-from-understanding-a-language-to-its-processing-1bf1f62453c1 dostupno 9.2.2022.

[4] https://narrativescience.com/resource/blog/what-is-natural-language-generation/ dostupno 9.2.2022.

[5] https://www.avenga.com/magazine/natural-language-processing-application-areas/ dostupno 9.2.2022.

[6] Y. Xie, L. Le, Y. Zhou, and V. V. Raghavan, ‘‘Deep learning for natural language processing,’’ in Handbook of Statistics. Amsterdam, The Netherlands: Elsevier, 2018.

[7] https://builtin.com/data-science/recurrent-neural-networks-and-lstm dostupno 13.2.2022.

[8] https://towardsdatascience.com/recurrent-neural-networks-by-example-in-python-ffd204f99470 dostupno 13.2.2022.

[9] https://machinelearningmastery.com/gentle-introduction-backpropagation-time/ dostupno 15.2.2021.

[10] https://medium.com/analytics-vidhya/i-strongly-recommend-to-first-know-how-rnn-algorithm-works-to-get-along-with-this-post-of-gated-9b4bf4f0ced2 dostupno 25.2.2022.

[11] https://colah.github.io/posts/2015-08-Understanding-LSTMs/ dostupno 26.2.2022.

[12] https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/ dostupno 26.2.2022.

[13] https://en.wikipedia.org/wiki/Long_short-term_memory dostupno 26.2.2022.

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin, "Attention is All you Need", Advances in Neural Information Processing Systems 30 (NIPS 2017)

[15] ROTHMAN, Denis. Transformers for Natural Language Processing: Build innovative deep neural network architectures for NLP with Python, PyTorch, TensorFlow, BERT, RoBERTa, and more. Packt Publishing Ltd, 2021.

[16] https://towardsdatascience.com/skip-gram-nlp-context-words-prediction-algorithm-5bbf34f84e0c dostupno 16.5.2022.

[17] https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270 dostupno 30.5.2022.

[18] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).

[19] https://medium.com/dataseries/roberta-robustly-optimized-bert-pretraining-approach-d033464bd946 dostupno 3.6.2022.

[20] Liu, Yinhan, et al. "Roberta: A robustly optimized bert pretraining approach." arXiv preprint arXiv:1907.11692 (2019).

[21] Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., Soricut, R.: ALBERT: a lite BERT for self-supervised learning of language representations., ICLR 2020

[22] https://www.borealisai.com/en/blog/understanding-xlnet/ dostupno 17.6.2022.

[23] Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R.R., Le, Q.V.: XLNet: generalized autoregressive pretraining for language understanding. In: Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS 2019), pp. 5753–5763 (2019)

[24] https://towardsdatascience.com/bleu-bilingual-evaluation-understudy-2b4eab9bcfd1 dostupno 24.6.2022.

[25] https://towardsdatascience.com/nlp-metrics-made-simple-the-bleu-score-b06b14fbdbc1 dostupno 24.6.2022.

[26] https://machinelearningmastery.com/calculate-bleu-score-for-text-python/ dostupno 14.7.2022.

[27] https://mccormickml.com/2019/11/05/GLUE/ dostupno 20.7.2022.

[28] https://towardsdatascience.com/matthews-correlation-coefficient-when-to-use-it-and-when-to-avoid-it-310b3c923f7e dostupno 14.8.2022.

[29] https://bmcgenomics.biomedcentral.com/articles/10.1186/s12864-019-6413-7 dostupno 14.8.2022.

[30] https://huggingface.co/datasets/allenai/c4 dostupno 1.9.2022.

[31] https://arxiv.org/abs/1805.12471 dostupno 1.9.2022.

[32] https://nlp.stanford.edu/~socherr/EMNLP2013_RNTN.pdf dostupno 1.9.2022.

Popis kôda

Popis slika

Popis Jednadžbi