Arhitekture dubokih neuronskih mreža u problemu obrade prirodnog jezika

Štefanić, Tin

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Tin Štefanić

ARHITEKTURE DUBOKIH NEURONSKIH MREŽA U PROBLEMU OBRADE PRIRODNOG JEZIKA

Diplomski rad

Voditelj rada:
prof. dr. sc. Luka Grubišić

Zagreb, rujan, 2021.

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

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

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Zahvaljujem obitelji na podršci tijekom studiranja, te mentoru prof. dr. sc. Luki Grubišiću na pomoći i savjetima za izradu diplomskog rada.

Sadržaj

Uvod

Umjetna inteligencija je područje koje se počelo jako brzo razvijati kroz zadnjih desetak godina. Unutar umjetne inteligencije jedno od najznačajnijih područja je strojno učenje, gdje se model trenira da iz ulaznih podataka, tj. na parovima ulaza i odgovarajućih opaženih (izmjerenih) izlaza, nauči vraćati željene izlaze. Takvo treniranje zahtjeva mnogo podataka, no razvoj tehnologije, posebno interneta, rezultirao je time da u današnje doba pogodnih skupova podataka ima jako puno, najviše skupova koji sadržavaju slike i tekst, pa su prirodno mreže koje obrađuju te podatke od velikog interesa. U računalnom vidu došlo je do velike promjene kada su se počele koristiti konvolucijske neuronske mreže (CNN) koje su ubrzo postale najuspješniji model i revolucionirale područje računalnog vida. One su imale dvije jako značajne osobine, arhitektura mreže je bila takva da je omogućavala korištenje algoritama koji su bili vrlo pogodni za paralelizaciju. Ovo je omogućilo vrlo brzo treniranje mreža, te se mreža trenirana na jednom problemu mogla uz malo "dodatnog" treniranja koristiti vrlo uspješno na nekom drugom problemu. Ovakav pristup danas zovemo prijenosno učenje (eng. *transfer learning*).

No u obradi prirodnog jezika (eng. NLP) dugo vrijeme slična arhitektura nije postajala. Ovdje su najuspješniji modeli bili rekurzivne neuronske mreže (RNN), te mreže bazirane na njima. Ovako konstruirane mreže su imale dva bitna nedostatka, nisu bile pogodne za paralelizaciju te nisu bile pogodne za prijenosno učenje. CNN modeli su bili korišteni i u NLP-u, ali iako uspješni tu nisu našli uspjeh koji su imali u računalnom vidu. No s vremenom se razvila Transformer arhitektura, koja je donijela veliku promjenu u zadacima NLPa. Ona je, poput CNN-a, bila vrlo pogodna za paralelizaciju, davala je najbolje rezultate u mnogim problemima, te su se iz Transformera počeli razvijati napredniji pred-trenirani modeli. Ti modeli su također bili oblikovani tako da podržavaju prijenosno učenje, te je jedan od prvih modela među njima model BERT.

U ovom radu ćemo prvo obraditi nekoliko osnovnih koncepta u obradi prirodnog jezika, poput rekurzivnih neuronskih mreža, te ćemo pred kraj prvog poglavlja obraditi LSTM, vrlo značajnu nadogradnju običnih rekurzivnih mreža. Zatim ćemo u idućem poglavlju obraditi Transformer arhitekturu. To je arhitektura koja nije imala mane, u smislu fleksibilnosti i skalabilnosti, koje su arhitekture prije nje imale. Ona je služila kao osnova za razvoj prvih više namjenskih pred-treniranih modela u obradi prirodnog jezika, te ćemo

jedan od tih modela, BERT, te njegovu nadogradnju ALBERT, obraditi u trećem poglavlju. Na kraju ćemo u četvrtom poglavlju usporediti modele na SQuAD (The Stanford Question Answering Dataset) skupu podataka.Ovaj skup podataka je dostupan na adresi `https://rajpurkar.github.io/SQuAD-explorer/`. Na toj stranici također možemo naći i mjere uspješnosti algoritama koji su koristili ovaj skup podataka (vidi referencu [9]).

Poglavlje 1

LSTM

U ovom poglavlju ćemo opisati rekurzivnu neuronsku mrežu (RNN) i njezinu nadogradnju, *long short term memory* (LSTM) mrežu. RNN je bilo koja mreža koja sadrži ciklus unutar svojih mrežnih konekcija. To jest, bilo koja mreža gdje je vrijednost sloja[1] direktno, ili indirektno, ovisna o svom prethodnom izlazu, kao ulazu. Iako moćne, takve mreže je teško razumjeti i trenirati. Ali, u ovoj klasi rekurzivnih mreža postoje arhitekture koje su se pokazale vrlo efektivnim kada su primijenjene na govorni i pisani jezik. U ovom poglavlju promatramo jednostavne RNN-ove pa ćemo kasnije promotriti napredniji pristup, LSTM.

1.1 RNN

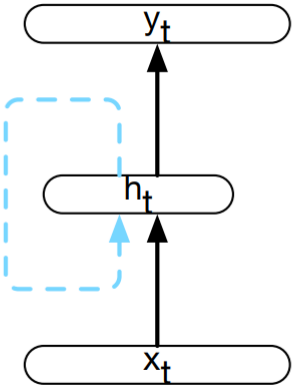
Prvo definiramo neke osnovne pojmove.

Funkciju $g : \mathbb{R} \to \mathbb{R}$ nazivamo aktivacijskom funkcijom ako želimo naglasiti da je koristimo s tenzorima, po komponentama tenzora.

Definicija 1.1.1 (Potpuno povezana neuronska mreža)**.** *Neka su $n, m \in \mathbb{N}$ prirodni brojevi, $W \in \mathbb{R}^{m \times n}$ matrica, $b \in \mathbb{R}^m$ vektor, te $g : \mathbb{R} \to \mathbb{R}$ aktivacijska funkcija. Potpuno povezan sloj je preslikavanje koje vektoru $x \in \mathbb{R}^n$ pridružuje vektor $y = g(Wx + b)$, $y \in \mathbb{R}^m$. Ovdje matricu W zovemo matricom težina. Jedan sloj, ili kompoziciju više ovakvih slojeva, nazivamo potpuno povezana neuronska mreža.*

Definicija 1.1.2 (Skriveni sloj neuronske mreže)**.** *Za sloj neuronske mreže kažemo da je skriven ako se njegov izlaz koristi kao ulaz u neki drugi sloj. Pritom ostale slojeve, čiji izlazi predstavljaju izlaz cijele mreže, nazivamo izlaznim slojevima.*

[1] Pod slojem mreže smatramo bilo koje preslikavanje unutar mreže, pritom to preslikavanje može biti sastavljeno od više manjih preslikavanja.

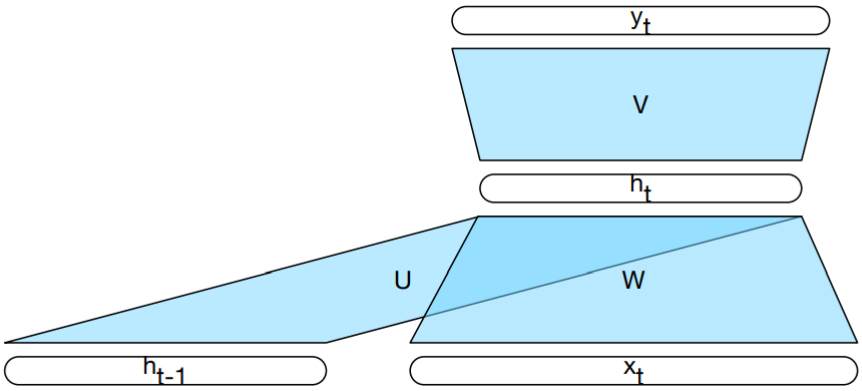


Slika 1.1: Jednostavan RNN. Skriven sloj sadrži rekurzivnu vezu kao dio svog ulaza. To jest, izlazna vrijednost skrivenog sloja ovisi o trenutnom ulazu kao i o prethodnim izlaznim vrijednostima skrivenog sloja iz prošlog vremenskog koraka. Iz [6].

Slika 1.1 ilustrira strukturu RNN-a. Kao kod potpuno povezanih (eng. *fully connected*) neuronskih mreža, ulazni vektor koji predstavlja trenutni ulaz, x_t, je pomnožen s matricom težina i zatim prođe kroz nelinearnu aktivacijsku funkciju da bi se izračunale vrijednosti za skriveni sloj. Taj skriveni sloj se onda koristi za izračunavanje odgovarajućeg izlaza, y_t. Glavna razlika od potpuno povezanih mreža je rekurzivna veza prikazana na slici 1.1 s isprekidanom linijom. Ova veza doda ulazu u računanje na skrivenom sloju vrijednost skrivenog sloja iz prethodnog koraka. Te korake, koji odgovaraju poziciji u ulaznoj sekvenci, nazivamo vremenskim koracima.

Skriveni sloj iz prethodnog vremenskog koraka daje kontekst koji kodira ranije procesiranje i informira odluke koje će se dodati u kasnijim vremenskim trenutcima. Izrazito bitno je to što ovaj pristup ne ograničava prethodni kontekst nekom unaprijed određenom duljinom, nego kontekst sadržan u prijašnjim skrivenim slojevima uključuje informaciju proizvoljno daleku, skroz do početka promatrane sekvence.

Dodavanje ove vremenske dimenzije naizgled čini RNN-ove kompliciranijima od arhitektura koje nisu rekurzivne. Ali u stvarnosti, one nisu toliko različite. Uz dan ulazni vektor i vrijednosti iz skrivenog sloja prethodnog vremenskog koraka, i dalje radimo standardno propagiranje unaprijed. To se lakše može vidjeti iz slike 1.2 koja razjasni prirodu rekurzije i kako se koristi u računanju u skrivenom sloju. Najznačajnija razlika je u novoj matrici težina, U, koja povezuje skriveni sloj iz prethodnog vremenskog koraka s trenutnim skrivenim slojem. Ove težine određuju kako mreža koristi prošli kontekst u računanju izlaza od trenutnog ulaza. Kao kod ostalih težina u mreži, te veze se nauče za vrijeme treniranja.



Slika 1.2: Jednostavan RNN u vremenskom koraku t. Iz [6].

1.2 Zaključivanje unaprijed u RNN-ovima

Zaključivanje unaprijed (eng. *forward inference*) u neuronskoj mreži je mapiranje sekvence ulaza u sekvencu izlaza. U RNN-u zaključivanje unaprijed je skoro identično kao i u FFN-ovima. Da bi izračunali izlaz y_t od ulaza x_t, trebamo izlazne vrijednosti skrivenog sloja h_t. Da bi to izračunali, pomnožimo ulaz x_t s matricom težina W, i izlaz skrivenog sloja iz prošlog vremenskog koraka h_{t-1} pomnožimo s matricom težina U. Zatim zbrojimo dobivene vrijednosti i proslijedimo ih odgovarajućoj aktivacijskoj funkciji, g, da bi stigli do izlazne vrijednosti trenutnog skrivenog sloja, h_t. Jednom kada imamo ove vrijednosti za skriveni sloj, nastavljamo sa standardnim računanjem izlaznog vektora.

$$h_t = g(Uh_{t-1} + Wx_t) \tag{1.1}$$

$$y_t = f(Vh_t) \tag{1.2}$$

Označit ćemo dimenzije ulaznog, skrivenog i izlaznog sloja, redom, d_{in}, d_h i d_{out}. Uz ove oznake, naše tri matrice težina su $W \in \mathbb{R}^{d_h \times d_{in}}$, $U \in \mathbb{R}^{d_h \times d_h}$ i $V \in \mathbb{R}^{d_{out} \times d_h}$.

U često susretnutom slučaju blage klasifikacije[2], računanje y_t se sastoji od korištenja softmax-a koji nam da vjerojatnosnu distribuciju nad mogućim klasama.

$$y_t = \text{softmax}(Vh_t) \tag{1.3}$$

[2]Klasificiranje dijelimo na blagu i strogu klasifikaciju, u blagoj klasifikaciju odredimo vjerojatnosti svake klase, pa izaberemo najvjerojatniju, a kod jake klasifikacije direktno biramo klasu, bez da klasama pridružimo vjerojatnosti.

Definicija 1.2.1 (softmax)**.** *Preslikavanje* softmax : $\mathbb{R}^n \to \mathbb{R}^n, n \in \mathbb{N}$, *za* $x \in \mathbb{R}^n$ *definirano sa:*

$$\text{softmax}(x) = \frac{\exp(x)}{\sum_{i=1}^{n} \exp(x_i)} \tag{1.4}$$

nazivamo softmax. Pritom ovdje exp *predstavlja eksponencijalnu funkciju po komponentama, te je dijeljenje također po komponentama.*

Činjenica da računanje u vremenu t zahtjeva vrijednost od skrivenog sloja iz vremena $t-1$ zahtjeva inkrementirajući algoritam zaključivanja koji prolazi od početka sekvence do kraja kao što je ilustrirano u slici 1.3. Sekvencijalna priroda jednostavnih rekurzivnih mreža se također može vidjeti *odmatanjem* mreže kroz vrijeme kao što je prikazano na slici 1.4.

function FORWARDRNN(x, *network*) **returns** output sequence y

$h_0 \leftarrow 0$
for $i \leftarrow 1$ **to** LENGTH(x) **do**
  $h_i \leftarrow g(U\ h_{i-1} + W\ x_i)$
  $y_i \leftarrow f(V\ h_i)$
return y

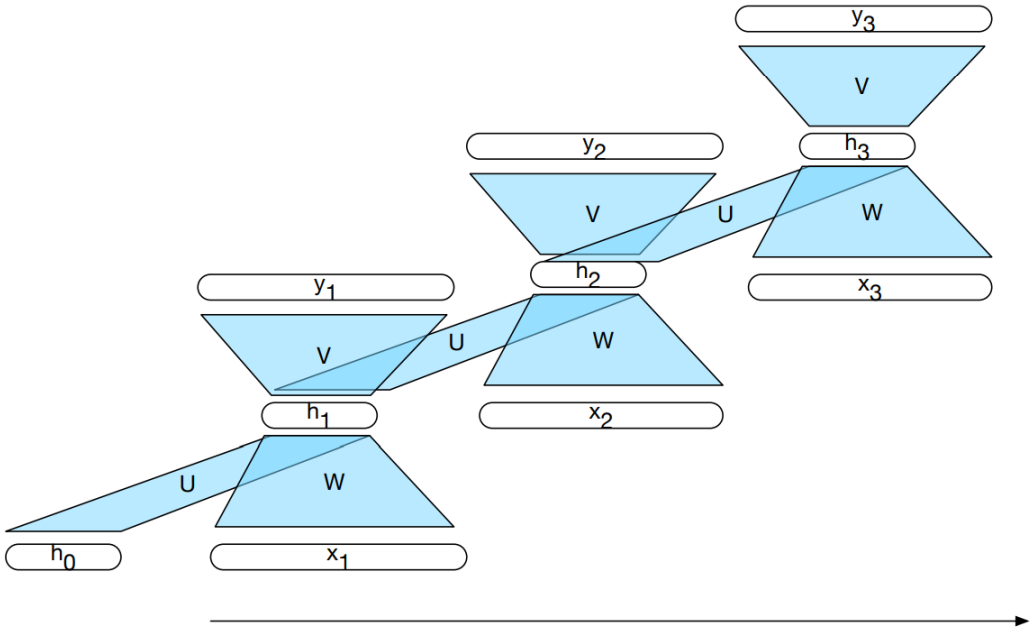
Slika 1.3: Zaključivanje unaprijed u jednostavnoj rekurzivnoj neuronskoj mreži. Matrice U, V i W su dijeljene kroz vremenske korake, dok se nove vrijednosti za h i y računaju kroz svaki vremenski korak. Iz [6].

1.3 LSTM motivacija

U praksi, teško je trenirati RNN-ove za probleme koji zahtijevaju mrežu da koristi informacije udaljene od trenutne pozicije koju obrađujemo. Iako ima pristup cijeloj prethodnoj sekvenci, informacija spremljena u skrivenim slojevima je uglavnom dosta lokalna, ta informacija je relevantnija za nedavne dijelove ulazne sekvence i nedavne odluke. Često je baš ta udaljena informacija kritična u mnogim jezičnim primjenama. Da bi to prikazali, navodimo sljedeći primjer.

Primjer 1.3.1. ***Letove koje je zračna luka otkazala bili su puni.***

Dodjeljivanje velike vjerojatnosti da se riječ ***je*** *odnosi na riječ* ***luka*** *je jednostavno jer su obje riječi u jednini i blizu što daje jaki lokalni kontekst. Ali, dodijeliti odgovarajuću*



Slika 1.4: Jednostavna rekurzivna neuronska mreža prikazana odmotana kroz vrijeme. Slojevi mreže su kopirani za svaki vremenski korak, dok su težine U, V i W dijeljene kroz sve vremenske korake. Iz [6].

vjerojatnost riječi ***su*** *je poprilično teško, ne samo zato što je riječ* ***letove*** *dosta udaljena, nego zato što se u kontekstu između javljaju odnosi riječi u jednini, dok je riječ* ***letove*** *u množini. Idealno, mreža bi trebala sačuvati udaljenu informaciju o riječi* ***letove*** *sve dok joj ne zatreba, dok i dalje točno obrađuje dijelove sekvence koji su između te dvije riječi.*

Jedan od razloga zašto RNN-ovi ne uspijevaju prenijeti naprijed kritičnu informaciju je zato što skrivene slojeve, i težine koje određuju vrijednosti u skrivenim slojevima, tražimo da obave dva zadatka istovremeno: dati informaciju korisnu za trenutni kontekst, i ažurirati i prenijeti naprijed informaciju potrebnu za buduće odluke. Da bi se riješio ovaj problem, dizajnirale su se kompleksnije arhitekture s ciljem da riješe problem čuvanja relevantnog konteksta kroz vrijeme. Preciznije, mreža mora naučiti zaboravljati informaciju koja joj više ne treba i pamtiti informaciju koja će joj trebati za buduće odluke.

1.4 LSTM opis

LSTM mreže (originalno iz [5]) dijele problem upravljanja kontekstom u dva potproblema: brisanje informacije koja više nije potrebna iz konteksta, i dodavanje informacije koja bi mogla biti korisna kasnije. Bitna ideja kako riješiti oba problema je naučiti model upravljati ovim kontekstom. LSTM-ovi ovo postižu tako da prvo dodaju eksplicitni sloj za kontekst u arhitekturu (uz rekurzivne skrivene slojeve), i kroz uporabu sklopova za kontrolu protoka informacija unutar slojeva mreže. Ti slojevi su implementirani uz pomoć dodatnih težina i rade sekvencijalno na ulazu, prethodnom skrivenom sloju, i prethodnom kontekstnom sloju.

Sklopovi u LSTM-u su međusobno slično dizajnirani; svaki se sastoji od sloja propagiranja unaprijed, nakon kojeg slijedi sigmoid[3] aktivacijska funkcija, nakon koje slijedi množenje po komponentama (oznaka $\odot$) sa izlazom sloja na kojeg koristimo sklop. Izbor sigmoida za aktivacijsku funkciju slijedi iz toga što joj je slika interval $(0, 1)$ i što su njezini izlazi često vrlo blizu ili nuli ili jedinici. Kada ovo kombiniramo s množenjem po komponentama dobivamo efekt sličan tome od binarne maske[4]. Vrijednosti u sloju na kojeg djeluje sklop, kojima su pridružene vrijednosti blizu 1 u maski, prolaze skoro ne promijenjeno; dok vrijednosti kojima su pridružene niže vrijednosti u maski su praktički obrisane.

Prvi sklop koji promatramo je sklop zaboravljanja (eng. *forget gate*). Njegova uloga je obrisati informacije iz konteksta koje nam više ne trebaju. Sklop zaboravljanja računa težinsku sumu skrivenog sloja prethodnog stanja i trenutnog ulaza, te zatim rezultat prođe kroz sigmoid. Ova maska je onda pomnožena s vektorom konteksta da se makne informacija iz konteksta koja više nije potrebna.

$$f_t = \sigma(U_f h_{t-1} + W_f x_t) \tag{1.5}$$
$$k_t = c_{t-1} \odot f_i \tag{1.6}$$

Idući zadatak je izračunati pravu informaciju koju trebamo izvući iz prethodnog skrivenog stanja i trenutnih ulaza.

$$g_t = tanh(U_g h_{t-1} + W_g x_t) \tag{1.7}$$

Iduće, generiramo masku za sklop dodavanja (eng. *add gate*) da izaberemo informaciju koju ćemo dodati trenutnom kontekstu.

$$i_t = \sigma(U_i h_{t-1} + W_i x_t) \tag{1.8}$$
$$j_t = g_t \odot i_t \tag{1.9}$$

[3] $\sigma(x) = \frac{1}{1+\exp(-x)}$.

[4] Binarna maska je tenzor čije su vrijednosti ili 0 ili 1.

Iduće, dodajemo ovo modificiranom vektoru konteksta da bi dobili naš novi vektor konteksta.

$$c_t = j_t + k_t \tag{1.10}$$

Zadnji sklop koji ćemo koristiti je izlazni sklop koji se koristi da bi odredili koja informacija je potrebna za trenutno skriveno stanje (za razliku od informacije koju je potrebno sačuvati za kasnije odluke).

$$o_t = \sigma(U_o h_{t-1} + W_o x_t) \tag{1.11}$$

$$h_t = o_t \odot tanh(c_t) \tag{1.12}$$

Slika 1.5 ilustrira kompletno računanje jednog LSTM sloja. Uz dane odgovarajuće težine za razne sklopove, LSTM prihvaća kao ulaz kontekstni sloj, i skriveni sloj iz prethodnog vremenskog koraka, uz trenutni vektor ulaza. Onda generira ažurirani kontekst i skrivene vektore kao izlaze.

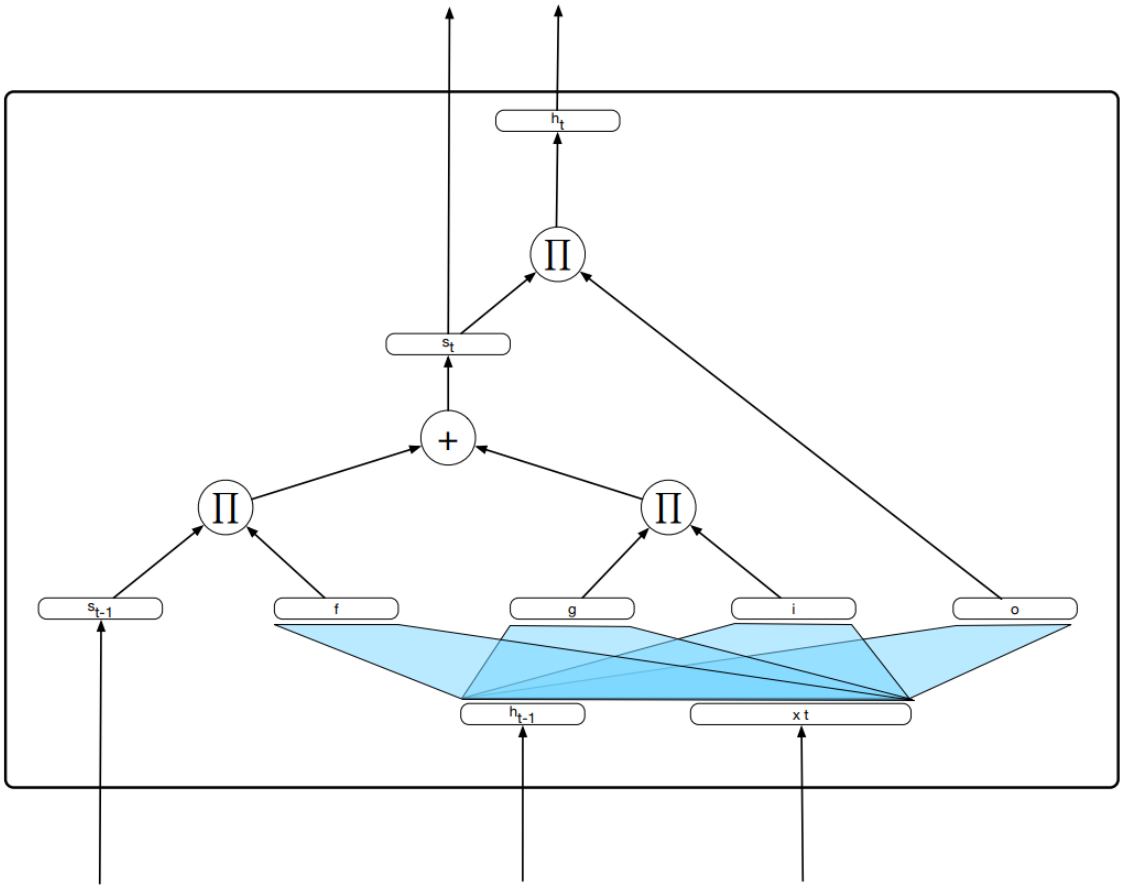
1.5 Ulaganja

Do sada kada smo objašnjavali RNN-ove i LSTM-ove, pretpostavili smo da imamo vektorske reprezentacije riječi, no nismo bili govorili o tome kako dolazimo do tih vektora. Naivan pristup je koristiti tzv. *one hot encoding*-e, ovdje poredamo sve riječi u našem vokabularu duljine V tako da svaka riječ ima svoju jedinstvenu poziciju, te onda i-toj riječi u vokabularu pridružimo vektor duljine V koji ima nule svugdje, osim na poziciji i gdje ima vrijednost 1. Ovaj pristup ima dvije bitne mane, prva je što je veličina vokabulara u primjenama često veća 30,000, što znači da ovaj pristup zahtjeva jako velike vektore. Druga mana je što se u ovakvim vektorima gube odnosi između riječi, jer jedinu informaciju koju ovi vektori nude o originalnoj riječi je njena pozicija u našem vokabularu. Napredniji pristupi pokušavaju stvoriti vektore koji očuvaju odnose između riječi.

Jedan od naprednijih pristupa su ulaganja (eng. *embedding*), to su vektori realnih brojeva gdje je svaki odgovara jednoj riječi iz korištenog vokabulara. Za razliku od prethodno spomenutih vektora, ovi vektori su kratki, najčešće dimenzija između 50-1000. Ovi vektori su gusti, te mogu imati i negativne brojeve, no nažalost ovdje dimenzije vektora nemaju jasnu interpretaciju. Do njih se dolazi treniranjem binarnog klasifikatora[5] za zadatak "Je li vjerojatno da se riječ **w** pojavi u okolini riječi **v**?" Ovdje nas ne zanima klasifikacija, nego naučene težine od klasifikatora jer se one koriste za ulaganja. Ovakva metoda za računanje ulaganja se zove *skip-gram with negative sampling*.

Nećemo u detalje opisati ovaj proces iz dva razloga, u primjenama se gotovo uvijek koriste javno dostupna pred-trenirana ulaganja, poput word2vec-a [8], te moderniji modeli

[5]Binarni klasifikator je klasifikator koji klasificira između točno dvije klase.

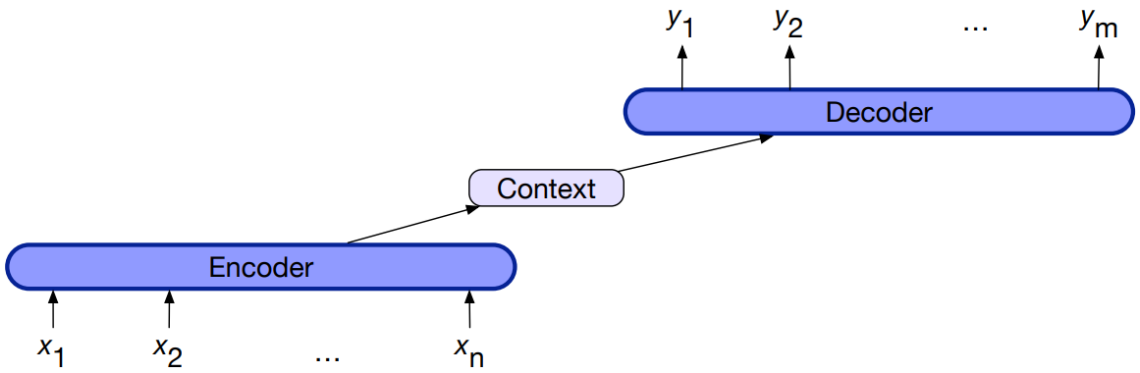


Slika 1.5: Jedan LSTM sloj prikazan kao usmjeren graf. Ulazi u sloj se sastoje od trenutnog ulaza, x_t, i prethodnog skrivenog stanja, h_{t-1}, i prethodnog konteksta, c_{t-1}. Izlazi su novo skriveno stanje, h_t i ažurirani kontekst, c_t. Ostali vrhovi predstavljaju funkcije, usmjereni bridovi u vrh funkcije predstavljaju argumente funkcije, dok usmjereni brid iz vrha predstavlja izlaz funkcije. Iz [6].

se više ne grade nad njima nego nad tzv. kontekstualnim ulaganjima poput BERT-a kojeg obrađujemo u kasnijem poglavlju.

1.6 Enkoder-dekoder arhitektura

U ovom poglavlju smo prošli kroz RNN-ove i kako s njima možemo procesirati sekvencu proizvoljne duljine, ali nam izlaz nije bio proizvoljne duljine, nego uvijek jednak duljini



Slika 1.6: Enkoder-dekoder arhitektura. Kontekst je funkcija skrivenih stanja ulaza, i nju dekoder može upotrijebiti na razne načine. Iz [6].

ulaza. To ograničenje je problematično jer postoje zanimljivi problemi kod kojih bi htjeli da nam izlaz može biti i različite duljine od ulaza, kao npr. strojno prevođenje, gdje najbolji prijevod neće uvijek biti iste duljine kao ulaz. Srećom za taj problem nam ne treba kompliciraniji model, dovoljno je koristiti dva RNN-a, tj. LSTM-a, koje nazivamo enkoder i dekoder. Enkoder prima ulaznu sekvencu i napravi od nje kontekstualnu reprezentaciju, često zvanom kontekst. Tu reprezentaciju onda koristi dekoder koji generira izlaznu sekvencu. Na slici 1.6 je ilustrirana ova arhitektura.

Enkoder-dekoder mreže se sastoje od 3 komponente:

1. Enkodera koji prihvaća ulaznu sekvencu, x^n, i generira odgovarajuću sekvencu kontekstualnih reprezentacija, h^n.

2. Vektor konteksta, c, koji je dobiven iz reprezentacija h^n, i prenosi srž ulaza dekoderu.

3. Dekoder koji prihvaća c kao ulaz i generira proizvoljno dugu sekvencu skrivenih stanja h^m, iz koji se dobiva odgovarajuća sekvenca izlaznih stanja y^m. Dekoder u vokabularu ima posebnu riječ, tj. token, koji označava kraj sekvence, te dekoder završava s generiranjem nakon što generira taj token.

U idućem poglavlju ćemo opisati jednu napredniju arhitekturu koja je sastavljena od enkodera i dekodera, Transformer arhitekturu.

Poglavlje 2

Transformer arhitektura

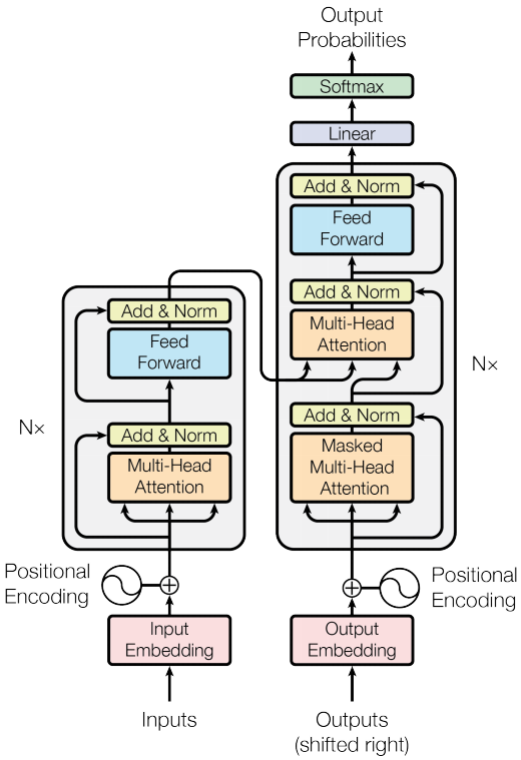
Iako su LSTM-ovi smanjili gubljenje dalekih informacija zbog rekurzije u RNN-ovima, problem je i dalje ostao. Slanje informacije kroz dulji niz rekurzivnih konekcija rezultira u gubitku relevantnih informacija i u otežavanju treniranja. Ovo se javlja kod zadataka poput odgovaranja na pitanja, gdje se odgovor treba naći u kontekstu sastavljenom od više rečenica. Također, sekvencijalna priroda rekurzivnih mreža ne dopušta primjenu paralelnih računalnih resursa. Ova razmatranja dovela su do razvoja Transformera, pristupu obrade sekvenci koji eliminira rekurzivne veze. U ovom poglavlju opisujemo Transformer arhitekturu predstavljenu u ovom radu [10]. Ona je strukturirana od enkodera i dekodera koristeći samopažnju (eng. *self-attention*) i potpuno povezane slojeve, vidljivo na slici 2.1.

2.1 Samopažnja

Slojevi samopažnje mapiraju ulaznu sekvencu oblika $(x_1, \ldots, x_n)$ u izlaznu iste veličine $(y_1, \ldots, y_n)$. Za vrijeme obrade svakog podatka iz ulaza, model ima pristup trenutnom i svim ranijim ulazima, ali nema pristupa informaciji o kasnijim ulazima (ovdje opisujemo slojeve s ovom restrikcijom, no kasnije ćemo koristiti i slojeve samopažnje bez ove restrikcije). Dodatno, računanje napravljeno nad svakim podatkom je nezavisno od svih ostalih računanja. Ovo drugo svojstvo znači da se ovaj dio modela može lagano paralelizirati. Na slici 2.2 se može vidjeti vizualizacija tog sloja.

Srž pristupa baziranom na pažnji je mogućnost uspoređivanja jednog podatka s kolekcijom drugih podataka na način koji otkriva njihovu važnost u trenutnom kontekstu. U slučaju samopažnje, to su usporedbe s ostalim elementima u istoj sekvenci. Rezultat tih usporedba se onda koristi za računanje izlaza trenutnog ulaza. Najjednostavniji oblik usporedbe između elemenata u sloju samopažnje je skalarni produkt. Rezultate tih usporedbi zovemo ocjene (eng. *score*).

$$score(x_i, x_j) = x_i \cdot x_j \tag{2.1}$$



Slika 2.1: Arhitektura Transformer modela, iz [10].

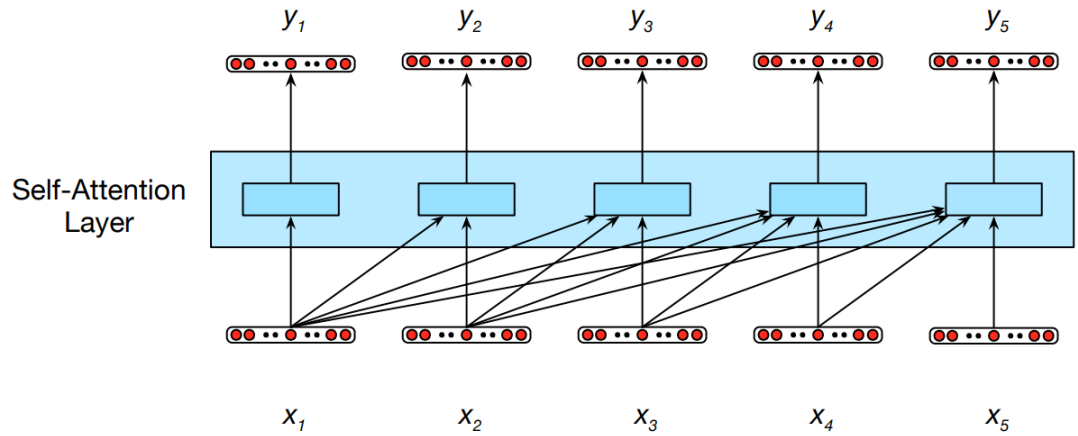
Rezultat skalarnog produkta je vrijednost između $-\infty$ i ∞, gdje veća vrijednost označava da su vektori koje uspoređujemo sličniji. Na primjer ako promatramo sliku 2.2, prvi korak u računanju y_3 je da izračunamo tri ocjene: $x_3 \cdot x_1$, $x_3 \cdot x_2$ i $x_3 \cdot x_3$. Da bi bolje iskoristili ove ocjene, normaliziramo ih softmax-om da bi napravili vektore težina, α_{ij}, koje prikazuju proporcionalnu značajnost svakog ulaznog elementa j naspram odabranog ulaznog elementa i koji je trenutni fokus pažnje.

$$\alpha_{ij} = \text{softmax}(score(x_i, x_j)) \quad \forall j \leq i \tag{2.2}$$

$$= \frac{\exp(score(x_i, x_j))}{\sum_{k=1}^{i} \exp(score(x_i, x_k))} \quad \forall j \leq i \tag{2.3}$$

Uz dane proporcionalne ocjene u α-ma, generiramo izlazne vrijednosti y_i tako da sumiramo sve do sad viđene ulaze s njima odgovarajućim α-ma.

$$y_i = \sum_{j \leq i} \alpha_{ij} x_j \tag{2.4}$$



Slika 2.2: Protok informacija u modelu samopažnje. U obradi svakog podatka iz sekvence model koristi i sve prethodne podatke. No za razliku od RNN-ova, računanje na svakom koraku je nezavisno od svih ostalih koraka pa se zato može izvoditi paralelno. Iz [6].

Koraci u prethodnim jednadžbama predstavljaju srž pristupa baziranom na pažnji: skup usporedbi relevantnih podataka u nekom kontekstu, normalizacija tih ocjena da bi pomoću njih dobili vjerojatnosnu distribuciju pomoću koje provedemo težinsku sumu. Rezultat ovog postupka je izlaz y.

Nažalost, ovakva jednostavna verzija se ne može trenirati, sve ovisi samo o originalnom ulazu x. Posebno, ovako se ne mogu naučiti različiti načini kako riječi mogu pridodati značaju duljih ulaza. Da bi omogućili ovakvo učenje, Transformeri sadrže dodatne parametre u obliku skupa matrica težina koje rade s ulaznim ulaganjima. Da bi motivirali ove nove parametre, predstavljamo uloge koje se odvijaju za vrijeme procesa pažnje.

- ※ Upit (eng. *query*) predstavlja trenutni fokus pažnje dok se uspoređuje sa svim prethodnim ulazima.
- ※ Ključ (eng. *key*) predstavlja vrijednost prethodnog ulaza s kojom se uspoređuje trenutni fokus pažnje.
- ※ Vrijednost (eng. *value*) koja se koristi za računanje izlaza trenutnog fokusa.

Da bi ostvario ove uloge koje ulazni ulaganja igraju u svakom od tih koraka, Transformer uvodi tri skupa težina koje označavamo sa W^Q, W^K i W^V. Te težine će se koristiti za izračunavanje linearnih transformacija svakog ulaza x s rezultirajućim vrijednostima

korištenim u svojim odgovarajućim ulogama (u nadolazećim računanjima).

$$q_i = W^Q x_i; \quad k_i = W^K x_i; \quad v_i = W^V x_i; \tag{2.5}$$

Neka su nam dana ulazna ulaganja veličine d_m, tada su dimenzije tih matrica redom $d_q \times d_m$, $d_k \times d_m$ i $d_v \times d_m$. U Transformer radu, d_m, d_k, d_q i d_v su bili 512.

Koristeći ove projekcije, ocjena između trenutnog fokusa, x_i i elementa iz prethodnog konteksta, x_j je sastavljena od vektorskog produkta između vektora upita q_i i vektora ključa k_j prethodnog elementa. Sada možemo ažurirati prethodni račun usporedbi s ovime na umu.

$$score(x_i, x_j) = q_i \cdot k_j \tag{2.6}$$

Nadolazeće računanje softmax-a koje rezultira u α_{ij} ostaje isto, ali izlaz računanja y_i je sada baziran na težinskoj sumi preko vektora vrijednosti v.

$$y_i = \sum_{j \leq i} \alpha_{ij} v_j \tag{2.7}$$

Slika 2.3 ilustrira taj proces u slučaju računanja trećeg izlaza u sekvenci, y_3.

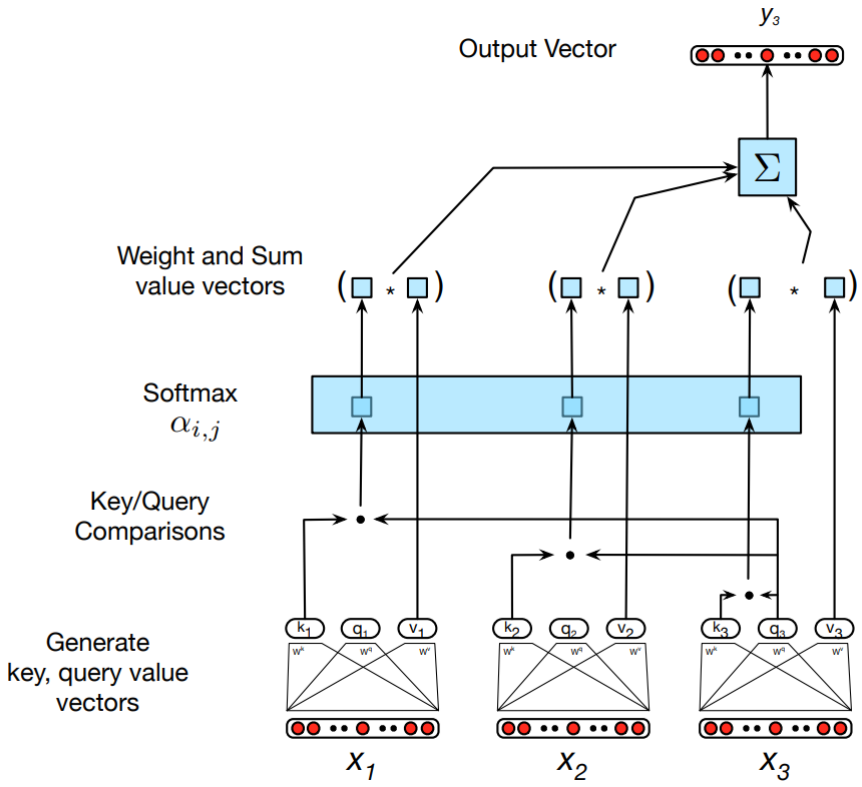
U praktičnoj primjeni se može javiti jedan problem prilikom računanja α_{ij} zbog korištenja skalarnog produkta za uspoređivanje u kombinaciji s eksponenciranjem u softmax-u. Pošto rezultat skalarnog produkta može biti proizvoljno velika (pozitivna ili negativna) vrijednost. Eksponenciranje tako velikih vrijednosti može uzrokovati numeričke probleme u slučaju pozitivnih vrijednosti, tj. u slučaju negativnih vrijednosti eksponenciranjem možemo dobiti nulu, čime će gradient postati nula za vrijeme treniranja. Da bi ovo izbjegli, skalarni produkt treba odgovarajuće skalirati. Skalirani skalarni produkt dijeli rezultat skalarnog produkta s faktorom povezanim s veličinom ulaganja prije nego što upotrijebimo softmax. Standardni pristup je podijeliti rezultat skalarnog produkta s korijenom dimenzije od upit i ključ vektora, i tako ažuriramo našu funkciju ocjene.

$$score(x_i, x_j) = \frac{q_i \cdot k_j}{\sqrt{d_k}} \tag{2.8}$$

Ovaj opis procesa samopažnje je bio iz perspektive računanja jednog izlaza u pojedinom trenutku. Ali, pošto su svi izlazi, y_i, računati nezavisno jedni od drugih ovaj cijeli proces se može paralelizirati koristeći efikasne metode matričnog množenja grupiranjem ulaznih ulaganja u jednu matricu i množeći je s matricom ključeva, upita i vrijednosti kako bi dobili matrice koje sadrže sve vektore ključa, upita i vrijednosti.

$$Q = W^Q X; \quad K = W^K X; \quad V = W^V X; \tag{2.9}$$

Uz gornje matrice možemo izračunati sve upit-ključ usporedbe istovremeno množeći K i Q u jednom matričnom množenju. Nadalje, možemo skalirati ove ocjene, uzeti njihov



Slika 2.3: Računanje vrijednosti trećeg elementa sekvence koristeći jednostavnu samopažnju. Iz [6].

softmax, i onda pomnožiti rezultat s V, i tako svesti cijeli korak pažnje za cijelu sekvencu na iduće računanje.

$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.10}$$

Nažalost, ovaj proces nekad napravi malo previše jer računanje usporedbi u QK^T rezultira ocjenom za svaki par upita i ključa, računajući one za koje je ključ nakon upita. Ako želimo koristiti samo prethodne riječi, da bi ovo popravili, elemente u gornje-trokutastom dijelu matrice QK^T ispunimo sa $-\infty$, da bi nakon softmax-a postali nule, čime eliminiramo znanje o idućim riječima u sekvenci. Taj postupak nazivamo maskiranje. Sada ćemo formalno definirati pažnju i samopažnju.

Definicija 2.1.1 (Pažnja)**.** *Neka su $n, d_k, d_q \in \mathbb{N}$ prirodni brojevi, te $Q \in \mathbb{R}^{d_q \times n}$, $K, V \in \mathbb{R}^{d_k \times n}$ ulazne matrice. Preslikavanje 2.10 nazivamo pažnja.*

Definicija 2.1.2 (Samopažnja)**.** *Neka su $d_m, d_q, d_k, n \in \mathbb{N}$ prirodni brojevi, $W^Q \in \mathbb{R}^{d_q \times d_m}$ i $W^K, W^V \in \mathbb{R}^{d_k \times d_m}$ matrice, te $X \in \mathbb{R}^{d_m \times n}$ ulazna matrica. Preslikavanje*

$$SelfAttention(X) = Attention(W^Q X, W^K X, W^V X) \tag{2.11}$$

nazivamo samopažnja.

Razlika između pažnje i samopažnje je to što u pažnji matrice upita mogu odgovarati različitoj sekvenci od one kojoj odgovaraju matrice ključa i vrijednosti (tako je bilo u radu gdje je pažnja predložena [2]), dok kod samopažnje sve tri matrice su dobivene iz iste sekvence.

2.2 Višeglavna samopažnja

Različite riječi u rečenici mogu imati više različitih odnosa istovremeno. Bilo bi teško da jedan sloj samopažnje nauči sve različite vrste paralelnih odnosa među svojim ulazima. Transformeri ovaj problem rješavanju sa slojevima višeglavne samopažnje (eng. *multihead self-attention*). To su skupovi slojeva samopažnje, zvani glave, koji se nalaze na paralelnim slojevima na istoj dubini modela, svaki sa svojim skupom parametara. Pošto koriste različite skupove parametara, svaka glava može naučiti različite aspekte odnosa koje postoje među ulazima na istoj razini apstrakcije. Ilustracija višeglavne pažnje se može vidjeti na slici 2.4.

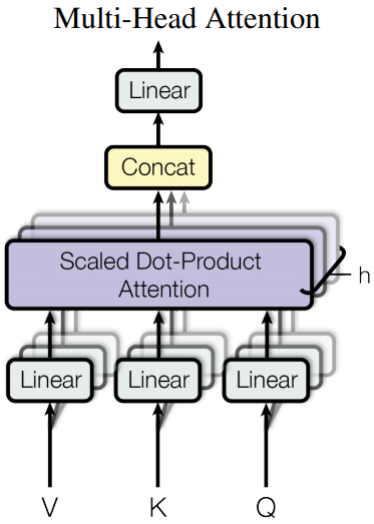
Da bi implementirali gornje, svaka glava, i, u sloju samopažnje ima svoj skup ključ, upit i vrijednost matrica: W_i^K, W_i^Q i W_i^V. One su korištene da bi projekcirali ulaz u sloj, x_i, odvojeno za svaku glavu, gdje ostatak računanja samopažnje ostaje nepromijenjen. Izlaz višeglavnog sloja s h glava sastoji se od h vektora iste duljine, te su u Transformer radu koristili $h = 8$. Da bi iskoristili ove vektore u idućim slojevima, kombiniramo ih i onda ih sažmemo do originalnih ulaznih dimenzija d_m. Ovo se postigne konkatenacijom izlaza iz svake glave, te zatim korištenjem još jedne linearne projekcije da bi taj novi vektor reducirali do originalnih dimenzija izlaza. Linearnu projekciju realiziriamo matricom W^O, dimenzija $d_m \times h\, d_m$.

Definicija 2.2.1 (Višeglavna pažnja)**.** *Neka su $h, n, d_q, d_k \in \mathbb{N}$ prirodni brojevi, gdje h označava broj glava, $W^O \in \mathbb{R}^{d_m \times h\, d_m}$ matrica, te $Q_i \in \mathbb{R}^{d_q \times n}, K_i, V_i \in \mathbb{R}^{d_k \times n}, \forall i \in \{1, \ldots, h\}$ ulazne matrice. Preslikavanje*

$$MultiHeadAttn(Q_1, \ldots, Q_h, K_1, \ldots, K_h, V_1, \ldots, V_h) = W^O(head_1 \oplus \cdots \oplus head_h) \tag{2.12}$$

$$head_i = Attention(Q_i, K_i, V_i) \tag{2.13}$$

nazivamo višeglavna pažnja.



Slika 2.4: Vizualni prikaz višeglavne pažnje, ovdje *scaled dot-product attention* predstavlja mehanizam pažnje koji smo opisali. Iz [10].

Definicija 2.2.2 (Višeglavna samopažnja)**.** *Neka su* $h, n, d_m, d_k \in \mathbb{N}$ *prirodni brojevi, gdje* h *označava broj glava,* $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d_k \times d_m}, \forall i \in \{1, \ldots, h\}$ *matrice, te* $X \in \mathbb{R}^{d_m \times n}$ *ulazna matrica. Preslikavanje*

$$MultiHeadSelfAttn(X) = W^O(head_1 \oplus head_2 \oplus \cdots \oplus head_h) \tag{2.14}$$

$$head_i = Attention(W_i^Q X, W_i^K X, W_i^V X) \tag{2.15}$$

nazivamo višeglavna samopažnja.

2.3 Pozicijska ulaganja

Kod RNN-a informacija o poretku ulaza je dolazila iz toga što RNN radi riječ po riječ. Nažalost isto ne vrijedi za Transformere, kod njih nema ničega što bi dalo modelu informaciju o relativnom ili apsolutnom poretku elemenata iz ulazne sekvence. To se može vidjeti tako što, ako fiksiramo i i promatramo odgovarajući $y_i = \sum_{j \leq i} \alpha_{ij} v_j$, v_j, a ni k_j i q_j, ne ovise o poziciji j-tog elementa (jer su dobiveni kao umnožak matrice i vektora), a $\alpha_{ij} = \text{softmax}(score(x_i, x_j)) \quad \forall j \leq i$ također ne ovisi o poretku jer je zbrajanje komutativno, a $score(x_i, x_j) = \frac{q_i \cdot k_j}{\sqrt{d_k}}$ također ne ovisi o poziciji j-tog vektora (jer je skalarni produkt vektora koji ne ovise o poziciji j-tog vektora). Pa kako $\alpha_{ij} v_j$ ovisi samo o j-tom vektoru, a

ne o poziciji j, to znači da ako promijenimo poredak prethodnih vektora, samo mijenjamo poredak brojeva u sumi, a brojeve koje zbrajamo ostaju isti (do na poredak). Da bi riješili ovaj problem, Transformerove ulaze kombiniramo s pozicijskim ulaganjima specifičnim za svaku poziciju u ulaznoj sekvenci.

Pozicijska ulaganja su vektori (realnih brojeva) pridruženi rednim brojevima, sada se postavlja pitanje kako možemo dobiti pozicijska ulaganja. Jednostavan i efikasan pristup je započeti s nasumično inicijaliziranim ulaganjima koji odgovaraju svakoj mogućoj ulaznoj poziciji do neke najveće duljine. Na primjer, kao što imamo ulaganje za riječ knjiga, imat ćemo ulaganje za poziciju 3. Kao kod riječnih ulaganja, ova pozicijska ulaganja su naučena s drugim parametrima za vrijeme treniranja. Da bi napravili ulazno ulaganje koje sadrži informaciju o poziciji, samo trebamo zbrojiti riječno ulaganje za svaki ulaz odgovarajućem pozicijskom ulaganju. Ovo novo ulaganje služi kao ulaz idućim slojevima modela.

Potencijalan problem s ovim pristupom je što će biti puno primjera u trening skupu za početne pozicije i značajno manje primjera za pozicije blizu najveće duljine. Ta ulaganja pri kraju bi mogla biti značajno slabije istrenirana i možda se uopće ne bi generalizirala za vrijeme testiranja. Alternativni pristup pozicijskim ulaganjima je izabrati statičku funkciju koja mapira ulaz kao cijeli broj u vektor realnih brojeva tako da sačuva odnose između pozicija. To jest, sačuva činjenicu da je pozicija 4 u ulazu uže povezana sa pozicijom 5 nego što je s pozicijom 17. Primjer takvog pristupa se koristi u Transformer radu.

U Transformer radu, koriste se funkcije sinus i kosinus.

$$PE_{pos,2i} = \sin\left(\frac{pos}{10000^{2i/d_m}}\right) \tag{2.16}$$

$$PE_{pos,2i+1} = \cos\left(\frac{pos}{10000^{2i/d_m}}\right) \tag{2.17}$$

ovdje je pos pozicija u sekvenci, a i je pozicija u vektoru. Duljina od PE je ista kao i od ulaganja, d_m.

2.4 Arhitektura modela

Sada prolazimo kroz cijeli Transformer model, koji se može vidjeti na slici 2.1. Model je podijeljen na enkoder i dekoder dio.

Enkoder se sastoji od $N = 6$ identičnih slojeva. Svaki sloj ima 2 podsloja. Prvi podsloj je višeglavni mehanizam samopažnje sličan kao onaj opisan ranije, s glavnom razlikom da ovdje dozvoljavamo svakoj poziciji da pristupi i pozicijama nakon, a ne samo prije, sebe. A drugi podsloj je jednostavna po poziciji potpuno povezana mreža. Također se koriste rezidualne veze oko svaka dva podsloja, nakon kojih se koristi normalizacija sloja (eng. *layer normalization*). To jest, izlaz svakog podsloja je LayerNorm(x+Sublayer(x)), gdje je Sublayer(x) funkcija koja predstavlja izlaz podsloja, prije reziduala i normalizacije sloja.

Da bi mogli koristiti reziduale, svi podslojevi u modelu, te svi slojevi ulaganja, stvaraju izlaze dimenzija $d_{model} = 512$.

Dekoder je također sastavljen od stoga od $N = 6$ identičnih slojeva. Osim što također ima dva podsloja koje imaju enkoder slojevi, dekoder dodaje treći podsloj, koji vrši višeglavnu pažnju preko izlaza od enkoder stoga. Preciznije ovdje koristimo upite iz prethodnog dekoder sloja, a ključ i vrijednosti iz izlaza od enkoder sloja. Slično kao kod enkodera, koriste se rezidualne konekcije oko svakog podsloja, nakon kojih slijedi normalizacija sloja. Ovdje podsloj samopažnje (na početku sloja) gleda samo (isključivo) prethodne pozicije. To se postiže već opisanim maskiranjem i pomakom izlaznih ulaganja za jednu poziciju.

Ranije spomenute rezidualne veze su standardna tehnika kod dubljih neuronskih mreža za poboljšanje rezultata i brzine treniranje mreže, i u praktičnim primjenama su se pokazale jako uspješnima. Rezidualna veza označava jednostavan sloj koji zbraja izlaz prethodnog sloja i ulaz prethodnog sloja. Također smo spomenuli normalizaciju sloja, to je jedna od tehnika za poboljšavanje brzine treniranja, te poboljšavanja generalizacije modela, predložena u ovom radu [1]. Neka promatramo l-ti sloj, čije ćemo izlazne vrijednosti označiti s vektorom a^l, dimenzija H. Sada računamo aritmetičku sredinu i variancu, te uz pomoć njih računamo nove izlaze $\hat{a}^l$ (ovdje je ϵ konstanta radi numeričke stabilnosti).

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \tag{2.18}$$

$$\sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2} \tag{2.19}$$

$$\hat{a}_i^l = \frac{a_i^l - \mu^l}{\sqrt{(\sigma^l)^2 + \epsilon}} \quad \forall i \tag{2.20}$$

Sada sloj normalizacije sloja možemo prikazati kao:

$$\mathrm{LN}_{\gamma,\beta}(a^l) = \gamma\,\hat{a}^l + \beta \tag{2.21}$$

Gdje su $\gamma \in \mathbb{R}$ i $\beta \in \mathbb{R}^H$ parametri koji se nauče za vrijeme treniranja.

Sada ćemo opisati po poziciji potpuno povezanu mrežu koju smo spomenuli ranije, koju primjenjujemo po komponentama ulaza u sloj (ulaz je sekvenca vektora, tj. matrica). Ona je sastavljena od dvije linearne transformacije s ReLU[1] aktivacijskom funkcijom između njih.

$$\mathrm{FFN}(x) = \max(0, W_1 x + b_1)W_2 + b_2 \tag{2.22}$$

Ovdje su dimenzije matrica W_1 i W_2, redom $d_{ff}\times d_{model}$ i $d_{model}\times d_{ff}$, te su dimenzije vektora b_1 i b_2, redom, d_{ff} i d_{model}. Iako su linearne transformacije iste na različitim pozicijama,

[1] ReLU(x) = max(0, x).

one koriste različite parametre iz sloja u sloj. U originalnom Transformer radu d_{model} je 512, a d_{ff} je 2048.

Preostaje nam opisati zadnja dva dijela modela. Ulazni i izlazni tokeni se pretvaraju u d_{model} dimenzionalne vektore pomoću naučenih ulaganja. Također se koriste linearna transformacija i softmax funkcija za konvertiranje izlaza dekodera u vjerojatnosnu distribuciju idućih tokena. Tokeni ovdje predstavljaju dijelove rečenice (najčešće riječi) te je linearna transformacija matrica dimenzija $V \times d_{model}$, gdje je V predstavlja veličinu vokabulara modela, tj. ukupan broj različitih riječi (ili tokena) koje model prepoznaje, koja je kod većih skupova podataka V najčešće oko 35000. Nakon linearne transformacije u dobivenom vektoru dimenzija V, svakoj poziciji odgovara jedan token.

Poglavlje 3

BERT

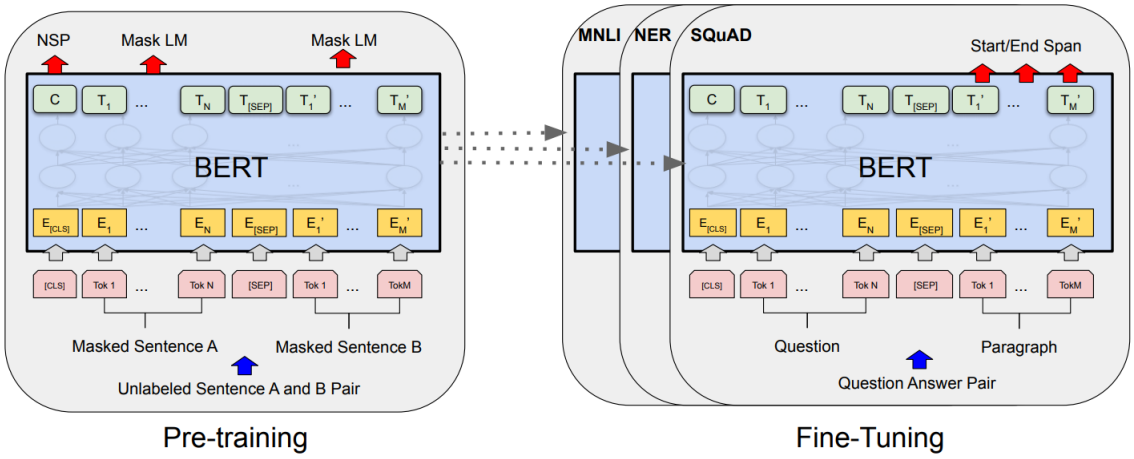
U ovom poglavlju obrađujemo BERT (***B**idirectional **E**ncoder **R**epresentations from **T**ransformers*), model baziran na enkoder dijelu Transformer modela. BERT je dizajniran da pred-trenira duboke dvosmjerne reprezentacije iz neetiketiranog (eng. *unlabeled*) teksta korištenjem lijevog i desnog konteksta u svim slojevima. To znači, da se pred-treniran BERT model može trenirati sa samo još jednim dodatnim izlaznim slojem da bi napravili moćne modele za razne zadatke, bez bitnih promjena u arhitekturi specifičnima za dani zadatak. To treniranje već pred-treniranog modela zovemo dotjerivanje (eng. *finetuning*). U kontekstu BERT-a ćemo promatrati nizvodne (eng. *downstream*) zadatke, tj. zadatke koji se pokušavaju riješiti dotjerivanjem nekog pred-treniranog modela.

3.1 Opis BERT-a

Dva su bitna koraka u BERT modelu-u: pred-treniranje i dotjerivanje. Za vrijeme pred-treniranja, model je treniran na neetiketiranim podacima preko više različitih zadataka. Za dotjerivanje, BERT model je prvo inicijaliziran s pred-treniranim parametrima, i onda su svi parametri dotjerivani koristeći etiketirane podatke iz nizvodnog zadatka. Svaki nizvodni zadatak ima različiti dotjeran model, iako su inicijalizirani s istim pred-treniranim parametrima. Zadatak odgovaranja na pitanja iz slike 3.1 nam služi kao primjer.

Bitno svojstvo BERT-a je njegova ujedinjena arhitektura kroz različite zadatke. Razlika između pred-trenirane arhitekture i nizvodne arhitekture je minimalna.

Prateći oznake iz prošlog poglavlja, gdje N označava broj slojeva, d_{model} duljinu izlaza podslojeva, d_{ff} unutarnju dimenziju FFN sloja, a h broj glavi pažnje, navodimo dva modela u BERT radu: $\mathbf{BERT_{BASE}}$ ($N = 12$, $d_{model} = 768$, $d_{ff} = 3072$, $h = 12$, ukupno parametara= 110M) i $\mathbf{BERT_{LARGE}}$ ($N = 24$, $d_{model} = 1024$, $d_{ff} = 4096$, $h = 16$, ukupno parametara= 340M).



Slika 3.1: Sveukupna procedura pred-treniranja i dotjerivanja BERT-a. Osim izlaznih slojeva, ista arhitektura je korištena u pred-treniranju i dotjerivanju. Isti pred-trenirani parametri se koriste za inicijalizaciju modela za različite nizvodne zadatke. Za vrijeme dotjerivanja, svi parametri se dotjerivaju. [CLS] je specijalan token dodan na početak svakog ulaza, a [SEP] je specijalni separacijski token (npr. on odvaja pitanje od konteksta uz pomoć kojeg se treba odgovoriti). Iz [3].

Također BERT ne koristi ReLU aktivacijsku funkciju nego GELU (***G**aussian **E**rror **L**inear **U**nit*) [4]. GELU se pokazala kao bolja alternativa za ReLU funkciju te se koristi u većini Transformer modela. Ona je definirana pomoću funkcije distribucije normalne (Gaussove) razdiobe, tj. koristeći slučajnu varijablu $X \sim \mathcal{N}(0,1)$:

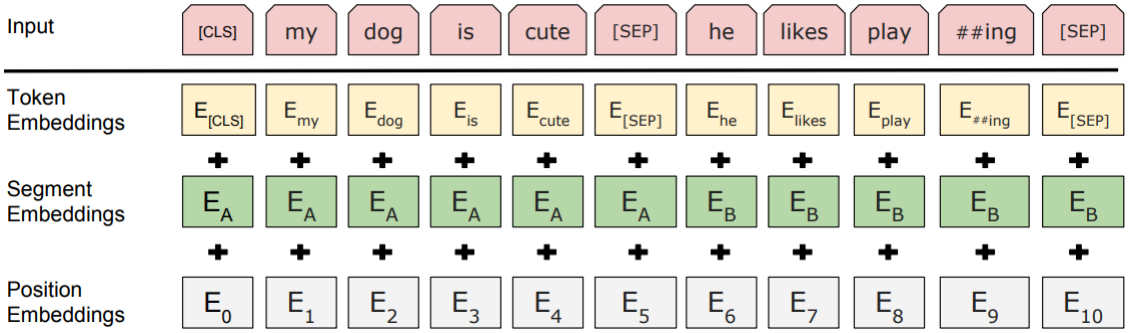
$$\mathrm{GELU}(x) := x\,P(X \leq x) = x\,\Phi(x) \tag{3.1}$$

Da bi BERT mogao rukovati raznim nizvodnim zadacima, on koristi ulaznu reprezentaciju koja može reprezentirati i jednu rečenicu i par rečenica u jednoj sekvenci tokena. Zato nadalje ”rečenica” može biti i tekst od više rečenica u kontekstu BERT-a. ”Sekvenca” predstavlja sekvencu BERT-ovih ulaznih tokena, i ona može biti ili jedna rečenica ili dvije rečenice zapakirane skupa.

BERT koristi WordPiece ulaganja ([12]) koja imaju vokabular od 30,000 tokena. Prvi token svake sekvence je uvijek specijalni token za klasificiranje ([CLS]). Zadnje skriveno stanje koje odgovara ovom tokenu se koristi kao združena sekvencna reprezentacija kod zadatka klasificiranja. Parovi rečenica su grupirani skupa u jednu sekvencu. Rečenice se razlikuju na dva načina. Prvo ih se odvoji s posebnim tokenom ([SEP]). Drugo, dodaju se naučena ulaganja svakom tokenu koji označavaju pripada li token rečenici A ili rečenici B. Kao što je prikazano na slici 3.1, ulazna ulaganja su označena s E, a zadnji skriveni vektor

specijalnog [CLS] tokena je označen sa $C \in \mathbb{R}^{d_{model}}$, te zadnji skriveni vektor za i-ti ulazni token je označen sa $T_i \in \mathbb{R}^{d_{model}}$.

Za dani token, njegova ulazna reprezentacija se konstruira sumirajući odgovarajuća token, segmenta i pozicijska ulaganja. Vizualizacija se može vidjeti na slici 3.2.



Slika 3.2: Reprezentacija BERT-ovog ulaza. Ulazna ulaganja su suma token, segment i pozicijskih ulaganja. Iz [3].

3.2 BERT pred-treniranje i dotjerivanje

Za pred-treniranje BERT koristi BooksCorpus (800M riječi) [13] i englesku Wikipediju (2,500M riječi). Kod Wikipedije se izvlače samo odlomci teksta i ignoriraju se liste, tablice i naslovi.

MLM

Ovdje se BERT koristi slično kao jezični model, tj. kao model koji za ulazne sekvence predviđa koji je idući token u sekvenci. No BERT neće predviđati iduću riječ, nego maskiramo neki postotak ulaznih tokena nasumično, i zatim BERT predviđa te maskirane tokene. Ovu proceduru autori BERT-a nazivaju MLM (eng. ***m**asked **l**anguage **m**odel*). U ovom slučaju završni skriveni vektori koji odgovaraju maskiranim tokenima su proslijeđeni izlaznom softmax-u.

Iako nam ovo omogućava da dobimo dvosmjerni pred-trenirani model, mana je da se stvara velika neusklađenost između pred-treniranja i dotjerivanja, jer se [MASK] token ne pojavljuje za vrijeme dotjerivanja. Da bi se ovo ublažilo, u BERT-u se ne zamjene uvijek "maskirane" riječi sa [MASK] tokenom. Generator trenirajućeg skupa podataka izabire 15% pozicija tokena nasumično za pogađanje. Ako je i-ti token izabran, zamijenimo i-ti token s:

1. [MASK] tokenom u 80% slučajeva.
2. Nasumičnim tokenom u 10% slučajeva.
3. Ostavimo originalni *i*-ti token u 10% slučajeva.

Onda, T_i se koristi za predviđanje originalnog tokena sa *cross entropy loss*-om.

NSP

Mnogo bitnih nizvodnih zadataka, poput odgovaranja na pitanja, su bazirani na razumijevanju odnosa između rečenica. Da bi BERT model razumio odnose rečenica, pred-treniran je na binarnom zadatku predviđanja iduće rečenice. Preciznije, biraju se rečenice A i B za svaki primjer u pred-treniranju, u 50% slučajeva B je stvarno rečenica koja slijedi A (etiketirana kao *IsNext*), i u drugih 50% slučajeva B je nasumična rečenica iz korpusa (etiketirana kao *NotNext*). Konačni model BERT-a ostvaruje točnost od 97-98% na NSP-u. Kao što je vidljivo na slici 3.1, *C* je korištena za *next sentence prediction* (NSP). No kasnija istraživanja nakon objave BERT-a su pokazala da NSP nema pouzdanog efekta.

Dotjerivanje

Dotjerivanje je jednostavno pošto mehanizam samopažnje u Transformeru BERT-u omogućava da modelira razne nizvodne zadatke, neovisno o tome koriste li jedan tekst ili par tekstova, zamjenjujući prikladne ulaze i izlaze.

Za svaki nizvodni zadatak, samo trebamo koristiti zadatak-specifične ulaze i izlaze s BERT-om i dotjerati sve parametre. Kod izlaza, reprezentacije tokena se proslijede izlaznom sloju za zadatke na razini tokena, kao npr. odgovaranje na pitanja, a reprezentacija [CLS] tokena se proslijedi izlaznom sloju za klasifikaciju.

3.3 ALBERT

BERT je bio jako značajan model te nakon što je postao javno dostupan bilo je od velikog interesa kako ga unaprijediti. U ovoj sekciji ćemo obraditi jedno od tih unaprjeđenja, ALBERT (***A Lite BERT***) [7]. Glavnu stvar koju ALBERT želi poboljšati je veličinu modela, tj. efikasnije korištenje parametara. Razlog je što iako povećavanje modela poput BERT-a dovodi do boljih rezultata, ograničeno je koliko se ti modeli mogu povećati jer GPU-evi i TPU-evi imaju ograničenu memoriju. ALBERT smanjuje veličinu modela uz pomoć dvije tehnike redukcije parametara.

Faktorizirana parametrizacija ulaganja

U BERT-u veličina WordPiece ulaganja, d_m, je vezana uz veličinu skrivenog sloja, tj. $d_m = d_{model}$. Ta odluka se pokazala suboptimalnom i za modeliranje i za praktičnu primjenu.

Iz perspektive modeliranja, WordPiece ulaganja trebaju naučiti reprezentacije nezavisne o kontekstu, dok za razliku ulaganja u skrivenom sloju trebaju naučiti reprezentacije ovisne o kontekstu. Snaga BERT-ovih reprezentacija je u tome što koriste kontekst da bi naučile reprezentacije ovisne o kontekstu. Zato, micanje veze veličine WordPiece ulaganja d_m i veličine skrivenih slojeva d_{model} nam omogućava efikasnije korištenje modelovih parametara, jer za potrebe modeliranja korisnije je imati $d_{model} \gg d_m$.

Iz praktične perspektive, obrada prirodnog jezika često zahtjeva da veličina vokabulara V bude velika. Ako je $d_m = d_{model}$, tada povećanje d_{model} uzrokuje povećavanje matrice ulaganja, čija je veličina $V \times d_m$. Ovo lagano može rezultirati u modelu s milijardama parametara, gdje se većina njih rijetko ažurira za vrijeme treniranja.

Zato, ALBERT koristi faktorizaciju parametara ulaganja, dekomponirajući ih u dvije manje matrice. Umjesto projeciranja *one-hot* vektora izravno u skriven prostor dimenzija d_{model}, prvo ih projeciramo u manje dimenzionalan prostor ulaganja veličine d_m, pa ih onda projeciramo u skriveni prostor. Koristeći ovu dekompoziciju, broj parametara ulaganja se smanji s $O(V \times d_{model})$ na $O(V \times d_m + d_m \times d_{model})$. Ova redukcija je značajna kada je $d_{model} \gg d_m$.

Dijeljenje parametara kroz slojeve

Kao drugi način poboljšanja efikasnosti parametara ALBERT koristi dijeljenje parametara kroz slojeve. Ima više načina za dijeliti parametre, npr. samo dijeliti FFN parametre kroz slojeve ili samo dijeliti *attention* parametre. ALBERT dijeli sve parametre kroz slojeve.

SOP

BERT koristi NSP da bi se naučili odnosi među rečenicama, no taj pristup se kasnije pokazao lošim. Pretpostavlja se da je razlog NSP-ove neefikasnosti to što NSP nije težak zadatak, u usporedbi s MLM-om. Kako je formuliran, NSP miješa predviđanje tema i predviđanje povezanosti u jednom zadatku. No, predviđanje teme je lakše za naučiti u usporedbi s predviđanjem povezanosti, te se također više preklapa s onime što je naučeno koristeći MLM.

Autori ALBERT-a se slažu da je modeliranje odnosa između rečenica bitan aspekt razumijevanja jezika, zato koriste drugu tehniku umjesto NSP-a. ALBERT koristi binarni zadatak *sentence-order prediction* (SOP), gdje za dane rečenice A i B treba odrediti je li B slijedi nakon A, ili A slijedi nakon B. SOP izbjegava predviđanje teme i fokusira se na modeliranje odnosa između rečenica. Kao pozitivne primjere SOP koristi istu tehniku kao

i BERT (dva uzastopna segmenta iz istog dokumenta), a kao negative primjere ista dva uzastopna segmenta, ali u obrnutom poretku. Ovo tjera modela da nauči finije veze između rečenica.

Poglavlje 4

Usporedba na SQuAD podacima

4.1 Promatrani skup podataka

U ovom poglavlju koristiti ćemo SQuAD v2.0 (***S**tanford **Q**uestion **A**nswering **D**ataset*) [9]. Ovaj skup podataka se sastoji od preko 150,000 parova pitanja i paragrafa konteksta, na koje se treba odgovoriti citirajući kontekst (tj. podstringom konteksta). Ovdje kontekst predstavlja paragraf s Wikipedije. Istaknimo da nije nužno da se sva pitanja mogu odgovoriti. Štoviše, oko 50,000 pitanja su napravljena da izgledaju slično pitanjima koji se mogu odgovoriti, ali nema odgovora na njih unutar konteksta. Za ovakva pitanja očekuje se od modela da odgovori da nema odgovora.

Upravo tih 50,000 pitanja bez odgovora bitno otežava ovaj skup podataka. U prijašnjoj verziji, SQuAD v1.1, koji nema tih pitanja bez odgovora, modeli koji su proizvodili vrlo dobre rezultate su imali jedan nedostatak, za bilo koji primjer pitanja i konteksta bi pogađali odgovor, čak i kad očito nema odgovora. To pogađanje odgovora ne predstavlja znanje odgovaranja na pitanja kakvo je originalno bilo zamišljeno od modela te je SQuAD v2.0 skup bio napravljen da popravi ovu manu originalnog SQuAD-a.

Točnost modela na SQuAD-u ocjenjujemo s dvije metrike, EM i F1. EM (***E**xact **M**atch* predstavlja postotak pitanja na koje je odgovoreno s odgovorom identičnim jednim od ponuđenih odgovora. Za razliku EM-a, koja uvjek vrati ili 0 ili 1, F1 metrika vraća broj u segmentu $[0, 1]$. Preciznije definirana je kao

$$\mathrm{F1} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} \tag{4.1}$$

Gdje *precision* predstavlja koliki postotak predviđenog odgovora predstavljaju tokeni koji se nalaze u oba odgovora, a *recall* (ili osjetljivost) koliki postotak točnog odgovora predstavljaju tokeni koji se nalaze u oba odgovora. To jest, ako sa *num_same* označimo broj zajedničkih tokena, sa *num_pred* broj tokena u predviđenom odgovoru, te sa *num_gold* broj

tokena u točnom odgovoru, imamo:

$$\text{precision} = \frac{\text{num\_same}}{\text{num\_pred}} \tag{4.2}$$

$$\text{recall} = \frac{\text{num\_same}}{\text{num\_gold}} \tag{4.3}$$

No ova formula nema smisla ako je jedan od odgovora prazan, tj. ako je odgovor "nema odgovora". U tom slučaju kažemo da je F1 = 1, ako su oba odgovora ista, tj. 0 ako su odgovori različiti.

Također, kao *benchmark*, i ljudi su bili rješavali testni skup podataka, u tablici 4.1 se mogu usporediti rezultati ljudi, trenutnog najboljeg modela, te BERT-a i ALBERT-a kada su izašli.

	EM	F1
ljudi	86.831	89.452
trenutni najbolji model (4.6.2021)	90.939	93.214
BERT (single model) (8.11.2018)	80.005	83.061
ALBERT (single model) (16.9.2019)	88.107	90.902
ALBERT (ensemble model) (18.9.2019)	89.731	92.215

Tablica 4.1: Izabrani rezultati na SQuAD v2.0 skupu podataka, u zagradi je datum kada je model bio dodan na rang listu. *Single model* predstavlja da je korišten jedan model, dok *ensemble* predstavlja da je trenirano više modela, te da je za konačan odgovor uzet najčešći odgovor od tih modela (u slučaju da je neodlučeno između više odgovora, uzima se najkraći odgovor). Zanimljivo je da su sposobnosti NLP modela narasle do razine gdje ostvaruju bolje rezultate od ljudi na ovom skupu podataka. Rezultati najboljeg modela su uzeti s rang liste 27.8.2021.

4.2 Primjena

U ovoj sekciji primjenjujemo BERT i ALBERT *base* modele na SQuAD skupu podataka. Iako su BERT i ALBERT dostupni izravno od njihovih autora[1,2], koristit ćemo python biblioteku zvanu transformers [11]. Ova biblioteka je kompatibilna sa pytorch-om (kojeg koristimo) i tensorflow-om, i daje pristup velikom broju različitih pred-treniranih Transformer modela, te odgovarajućih funkcija za pretvorbu teksta u tokene. Također koristimo

[1] https://github.com/google-research/bert

[2] https://github.com/google-research/ALBERT

ovaj repozitorij[3] korišten na Stanford-ovom predmetu cs224n[4], on sadrži mnoge pomoćne alate za treniranje modela na SQuAD skupu podataka.

Pošto test skup od SQuAD-a nije javno dostupan, za testni skup ćemo koristiti dev skup, što je opravdano time što treniramo samo jednu epohu, zbog veličine trening skupa, tj. dev skup ćemo koristiti samo za krajnju evaluaciju modela. Za *learning rate* koristimo $3 \cdot 10^{-5}$, to je jedna od vrijednosti koju autori BERT-a preporučuju za korištenje s BERT-om. Koristit ćemo Adam optimizator sa *batch size*-om 4. Također koristit ćemo Google Colab platformu[5]. Koristimo iste parametre za treniranje oba modela da bi ih mogli direktno usporediti.

Naša *loss* funkcija, tj. funkcija pomoću koje treniramo model tako da je minimiziramo, je *negative log-likelihood* (NLL), preciznije, ako očekivane početne i krajnje pozicije označimo sa i i j, tada je odgovarajući *loss*

$$\text{loss} = -\log(\mathbf{p}_{start}(i)) - \log(\mathbf{p}_{end}(j)) \tag{4.4}$$

Pritom, ako nema odgovora, tada za očekivane početne i krajnje pozicije uzimamo $i = j = 0$. Ovdje nam $\mathbf{p}_{start}(i)$ predstavlja vjerojatnost da je početna pozicija i, ta vjerojatnost je dobivena iz odgovarajućeg izlaza modela, vektora koji predstavlja vjerojatnosnu distribuciju nad svim pozicijama iz ulazne sekvence. Isto vrijedi i za $\mathbf{p}_{end}(j)$ (model vraća dva vektora kao izlaz).

Rezultati evaluiranja na dev skupu su vidljivi u tablici 4.2. Kao što smo očekivati, ALBERT model postiže vidljivo bolje rezultate, no ako usporedimo s razlikom BERT-a i ALBERT-a u tablici 4.1, vidimo da je tamo značajno veća razlika u modelima. Razlog tome je što smo mi koristili BERT i ALBERT modele slične arhitekture, tj. iako ALBERT model ima značajno manje parametara, i dalje se radi jednako računanja kao i u BERT modelu, zato je i treniranje oba modela trajalo jednako dugo. No autori ALBERT-a na SQuAD nisu koristili arhitekturu koja odgovara $\text{BERT}_{\text{LARGE}}$ modelu nego značajno veću, $\text{ALBERT}_{\text{XXLARGE}}$, razlike tih modela su vidljive na slici 4.1. Korištenje Tensorboard-a nam omogućava da promatramo ponašanje NLL-a za vrijeme treniranja, kao što je vidljivo na slici 4.2 za BERT model, tj. na slici 4.3 za ALBERT model.

[3] https://github.com/chrischute/squad

[4] https://web.stanford.edu/class/cs224n/

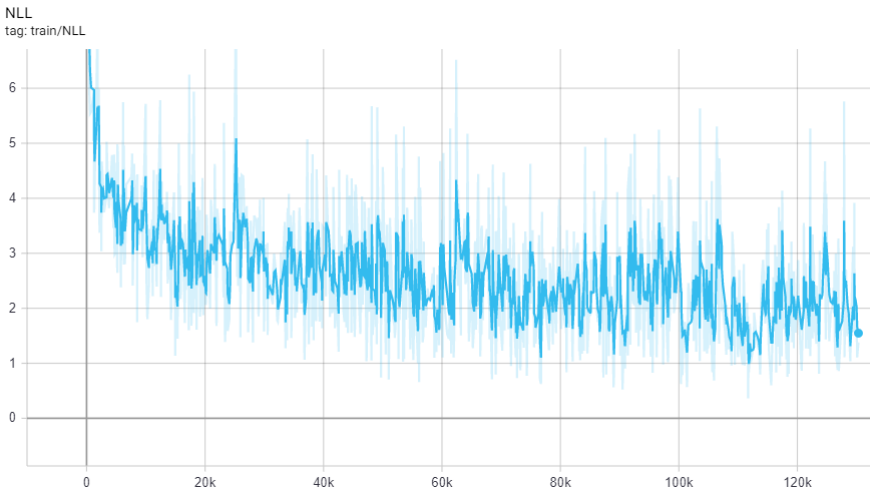
[5] https://colab.research.google.com

model	NLL	EM	F1	AvNA
BERT	2.43	64.86	68.22	74.47
ALBERT	2.13	69.13	72.46	78.61

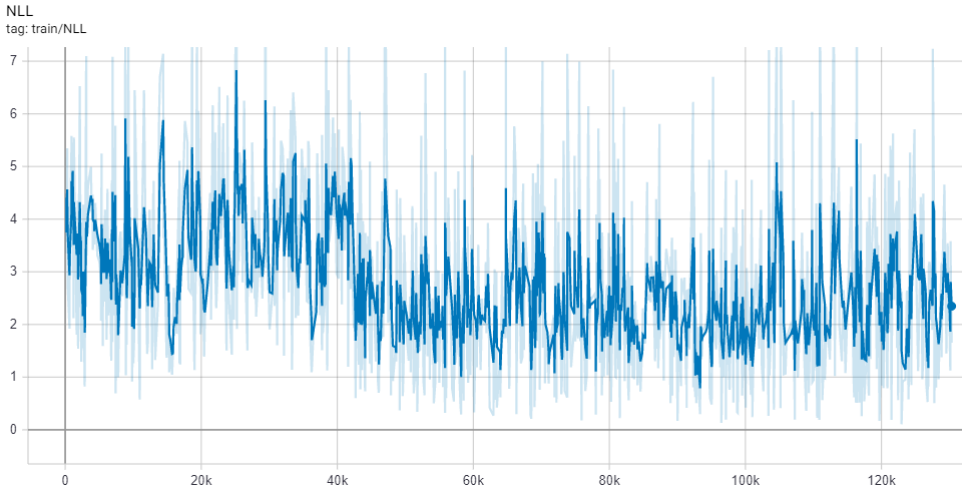
Tablica 4.2: Rezultati naših modela evaluiranih na dev skupu. Ovdje je AvNA kratica od *answer vs no answer*, tj. metrika koja mjeri koliki postotak primjera je model točno odredio da imaju odgovor, tj. da nemaju odgovor.

Model		Parameters	Layers	Hidden	Embedding	Parameter-sharing
BERT	base	108M	12	768	768	False
	large	334M	24	1024	1024	False
ALBERT	base	12M	12	768	128	True
	large	18M	24	1024	128	True
	xlarge	60M	24	2048	128	True
	xxlarge	235M	12	4096	128	True

Slika 4.1: Konfiguracije BERT i ALBERT modela. Iz [7].



Slika 4.2: NLL za vrijeme treniranja BERT-a. Apscisa predstavlja broj testnih primjera, a ordinata NLL.



Slika 4.3: NLL za vrijeme treniranja ALBERT-a. Apscisa predstavlja broj testnih primjera, a ordinata NLL.

Pitanje:

Where is the Congress Hall located?

Kontekst:

Thanks to numerous musical venues, including the Teatr Wielki, the Polish National Opera, the Chamber Opera, the National Philharmonic Hall and the National Theatre, as well as the Roma and Buffo music theatres and the Congress Hall in the Palace of Culture and Science, Warsaw hosts many events and festivals. Among the events worth particular attention are: the International Frédéric Chopin Piano Competition, the International Contemporary Music Festival Warsaw Autumn, the Jazz Jamboree, Warsaw Summer Jazz Days, the International Stanisław Moniuszko Vocal Competition, the Mozart Festival, and the Festival of Old Music.

Točan odgovor:

in the Palace of Culture and Science

Odgovor modela:

palace of culture and science , warsaw

Slika 4.4: Primjer pitanja iz dev skupa te BERT-ovog odgovora. Razlog zašto je odgovor modela u malim slovima je zato što je odgovor rekonstruiran iz token reprezentacije ulaza, a (ova varijanta) BERT-a ne koristi tokene koji razlikuju malo i veliko slovo.

Bibliografija

[1] Jimmy Lei Ba, Jamie Ryan Kiros i Geoffrey E. Hinton, *Layer Normalization*, 2016.

[2] Dzmitry Bahdanau, Kyunghyun Cho i Yoshua Bengio, *Neural Machine Translation by Jointly Learning to Align and Translate*, 2016.

[3] Jacob Devlin, Ming Wei Chang, Kenton Lee i Kristina Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019.

[4] Dan Hendrycks i Kevin Gimpel, *Gaussian Error Linear Units (GELUs)*, 2020.

[5] Sepp Hochreiter i Jürgen Schmidhuber, *Long Short-Term Memory*, Neural Computation **9** (1997), br. 8, 1735–1780.

[6] Dan Jurafsky i James H. Martin, *Speech and Language Processing (3rd ed. draft)*, 2020.

[7] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma i Radu Soricut, *ALBERT: A Lite BERT for Self-supervised Learning of Language Representations*, 2020.

[8] Tomas Mikolov, Kai Chen, Greg Corrado i Jeffrey Dean, *Efficient Estimation of Word Representations in Vector Space*, 2013.

[9] Pranav Rajpurkar, Robin Jia i Percy Liang, *Know What You Don't Know: Unanswerable Questions for SQuAD*, 2018.

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

[11] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest i Alexander M. Rush, *Transformers: State-of-the-Art Natural Language Processing*, Proceedings of

the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations (Online), Association for Computational Linguistics, listopad 2020, str. 38–45, https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[12] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes i Jeffrey Dean, *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*, 2016.

[13] Yukun Zhu, Ryan Kiros, Richard Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba i Sanja Fidler, *Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books*, 2015.

Sažetak

U ovom diplomskom radu obrađuju se izabrane arhitekture dubokih neuronskih mreža korištene u problemu obrade prirodnog jezika. U prvom poglavlju cilj je obraditi klasične arhitekture, rekurzivne neuronske mreže i LSTM, te njihove prednosti i mane. Zatim u idućem poglavlju se obrađuje modernija Transformer arhitektura, arhitektura koja sadrži prednosti klasičnih arhitektura, ali nema njihove mane. U trećem poglavlju su obrađene napredne pred-trenirane Transformer arhitekture, BERT i njegova nadogradnja ALBERT. Zatim u zadnjem poglavlju te arhitekture su primjenjene na problem odgovaranja na pitanja, problem na kojem su izražene mane klasičnih arhitektura.

Summary

This thesis deals with selected architectures of deep neural networks used in the problem of natural language processing. In the first chapter, the goal is to deal with classical architectures, recurrent neural networks and LSTM, and their advantages and disadvantages. Then, in the next chapter, we discuss the modern Transformer architecture, an architecture that possesses the advantages of classical architectures, but doesn't share their disadvantages (e.g. it is easier to use parallel algorithms and it allows for transfer learning) . The third chapter deals with the advanced pre-trained Transformer architecture, BERT and its upgrade ALBERT. Then, in the last chapter, these architectures are applied to a question answering problem, a problem on which the flaws of classical architectures are expressed.

Životopis

Rođen sam 13. listopada 1997. u Zagrebu. U Zagrebu sam pohađao osnovnu školu Nikole Tesle te XIII. gimnaziju, prirodoslovno-matematički smjer, gdje sam u 4. razredu na državnom natjecanju iz informatike osvojio drugo mjesto u kategoriji algoritama. U akademskoj godini 2016./2017. upisujem preddiplomski studij Matematika na Prirodoslovno-matematičkom fakultetu Sveučilišta u Zagrebu. Studij sam završio 2019., te u akademskoj godini 2019./2020. upisujem diplomski studij Računarstvo i matematika na istom fakultetu.