SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 153

ODREĐIVANJE PORAVNANJA PAROVA PROTEINSKIH SLJEDOVA KORIŠTENJEM MODELA DUBOKOG UČENJA

Ivan Furač

Zagreb, lipanj 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 153

ODREĐIVANJE PORAVNANJA PAROVA PROTEINSKIH SLJEDOVA KORIŠTENJEM MODELA DUBOKOG UČENJA

Ivan Furač

Zagreb, lipanj 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 10. ožujka 2023.

DIPLOMSKI ZADATAK br. 153

Pristupnik:	**Ivan Furač (0036513492)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentorica:	izv. prof. dr. sc. Mirjana Domazet-Lošo

Zadatak: **Određivanje poravnanja parova proteinskih sljedova korištenjem modela dubokog učenja**

Opis zadatka:

U okviru ovoga diplomskog rada potrebno je proučiti način reprezentacije proteinskih sljedova koji bi bio pogodan za analizu modelima dubokog učenja. Potrebno je analizirati arhitekturu transformer te ju je potrebno prilagoditi za analizu odabrane reprezentacije proteinskih sljedova. Zatim je potrebno izgraditi model nad skupom odabranih proteinskih sljedova koji bi omogućio određivanje poravnanja parova sljedova. Rješenje je potrebno implementirati u programskom jeziku Python, a dobivene rezultate potrebno je usporediti s rezultatima postojećih rješenja za poravnanje parova proteinskih sljedova.

Rok za predaju rada: 23. lipnja 2023.

Zahvaljujem mentorici izv. prof. dr. sc. Mirjani Domazet-Lošo na pomoći u izradi diplomskog rada i na protekle tri godine mentoriranja. Zahvaljujem obitelji što mi je omogućila školovanje te kolegicama i kolegama bez kojih studiranje ne bi bilo ni približno ovako zanimljivo.

Sadržaj

Uvod

Područje bioinformatike obuhvaća istraživanje, razvoj i primjenu računalnih alata i pristupa koji omogućuju prikupljanje, pohranu, organizaciju, analizu i vizualizaciju bioloških i medicinskih podataka [1]. Jedan od temeljnih postupaka u bioinformatici je poravnanje sljedova koje se može definirati kao određivanje korespondencija između pojedinih elemenata (engl. *residue*) sljedova [2]. Elementi mogu biti ili nukleotidi ili aminokiseline, dok korespondencije između njih mogu biti podudaranje (engl. *match*), nepodudaranje (engl. *mismatch*) te umetanje ili brisanje znakova. Svrha poravnavanja dvaju ili više sljedova je procjena njihove sličnosti na temelju čega se može odrediti jesu li sljedovi homologni, odnosno jesu li evolucijski srodni [1]. Utvrđivanjem homologije s poznatim sljedovima mogu se predvidjeti svojstva nepoznatih sljedova jer homologni sljedovi imaju sličnu strukturu, a često i sličnu funkciju [3]. Razvijeni su brojni alati koji omogućuju poravnanje sljedova. Alati BLAST i FASTA pronalaze zajedničke podnizove i na temelju njih grade cijelo poravnanje [4], [5]. Alat MUMmer koristi strukture podataka poput sufiksnih stabala i FM-indeksa [6]. MAFFT je alat koji koristi Fourierovu transformaciju za izračun korelacije između dva slijeda [7]. Postoje i brojni alati koji za određivanje poravnanja koriste skrivene Markovljeve modele [8]. U ovom se radu istražuje mogućnost određivanja poravnanja sljedova korištenjem modela dubokog učenja koji se temelje na arhitekturi *transformer.* Arhitektura *transformer* predstavljena je 2017. godine i omogućila je iznimno velik napredak u području obrade prirodnog jezika [9]. S obzirom na to da postoje sličnosti između prirodnog jezika i bioloških sljedova, *transformeri* imaju sve više primjena i u području bioinformatike pri čemu takvi modeli pokazuju bolju preciznost, efikasnost i interpretabilnost u odnosu na ostala rješenja [10]. Jedan od tih modela je i *BetaAlign* koji omogućuje višestruko poravnanje sljedova nukleotida ili aminokiselina te postiže rezultate koji su usporedivi s najpopularnijim algoritmima za višestruko poravnanje [11]. U nastavku rada će se detaljnije opisati svojstva proteinskih sljedova te svojstva arhitekture *transformer*. Predstavit će se vlastita implementacija modela za poravnanje parova (engl. *pairwise alignment*) proteinskih sljedova koji se temelji na arhitekturi *transformer* te će se dobiveni rezultati usporediti s rezultatima postojećih algoritama.

1. Poravnanje proteinskih sljedova

Proteinski slijed je niz aminokiselina pri čemu je svaka aminokiselina predstavljena jednoslovnim simbolom. Trojka nukleotida (A, C, T ili G) u DNA lancu kodira za pojedine aminokiseline prilikom procesa sinteze proteina pri čemu za istu aminokiselinu postoji više mogućih trojki koje ju kodiraju. Postoji ukupno 20 aminokiselina koje su kodirane takvim trojkama i koje su prikazane u tablici ispod zajedno sa svojim simbolima.

Tablica 1 Aminokiseline i njihovi simboli

Ime	Simbol	Ime	Simbol
Alanin	A	Izoleucin	I
Arginin	R	Leucin	L
Asparagin	N	Lizin	K
Asparaginska kiselina	D	Metionin	M
Cistein	C	Prolin	P
Histidin	H	Serin	S
Fenilalanin	F	Tirozin	Y
Glicin	G	Treonin	T
Glutamin	Q	Triptofan	W
Glutaminska kiselna	E	Valin	V

Postupkom poravnanja dva slijeda svaki simbol iz jednog slijeda uparuje se s odgovarajućim simbolom iz drugog slijeda pri čemu su moguće različite evolucijske promjene, odnosno mutacije. Mutacija može rezultirati supstitucijom jedne aminokiseline drugom pri čemu dolazi do nepodudaranja u poravnanju, te umetanjem ili brisanjem aminokiseline pri čemu dolazi do pojave procjepa (engl.

gap) koji se označava crticom. Primjeri različitih situacija koje se mogu dogoditi u poravnanju sljedova prikazani su u tablici ispod (Tablica 2). Naglasak u ovome radu je na poravnanju parova sljedova (engl. *pairwise alignment*). Složeniji problem je višestruko poravnanje sljedova (engl. *multiple sequence alignment*) gdje je cilj međusobno poravnati veći broj sljedova (tri ili više) odjednom.

Tablica 2 Primjer poravnanja s različitim mutacijama

Vrsta mutacije	**Primjer**
Nema mutacije (podudaranje, engl. *match*)	V **D R** M **R I** R **T W K** - - **V** Q **D R** - **R I** N **T W K** S L **V**
Supstitucija (nepodudaranje, engl. *mismatch*)	**V** D R M R I **R** T W K - - V **Q** D R - R I **N** T W K S L V
Umetanje/brisanje (pojava procjepa, engl. *gap*)	V D R **M** R I R T W K **- -** V Q D R **-** R I N T W K **S L** V

Iz gornje tablice se može zaključiti kako se poravnanje dva slijeda zapravo svodi na umetanje procjepa na odgovarajuća mjesta u oba slijeda čime se konačno dobivaju dva poravnata slijeda koja su jednake duljine. Cilj je postići što je više moguće podudaranja, a što manje supstitucija i procjepa. Što su sljedovi sličniji, odnosno evolucijski srodniji, to će biti moguće ostvariti više podudaranja. Omjer broja podudaranja u poravnatim sljedovima i njihove ukupne duljine može se koristiti kao procjena sličnosti tih sljedova, a time i procjena njihove evolucijske srodnosti.

Poravnavati se mogu i proteinski sljedovi i sljedovi nukleotida, međutim pokazalo se da je korisnije uspoređivati proteinske sljedove iz više razloga [1]. Pojedine mutacije u DNA lancu (koje mogu biti vrlo česte) ne moraju imati nikakav utjecaj na kodirani proteinski lanac, s obzirom na to da istu aminokiselinu može kodirati više različitih tripleta. Neke aminokiseline dijele vrlo slična biofizička svojstva, stoga kod poravnavanja ima više smisla sparivati takve parove aminokiselina, nego druge nepovezane aminokiseline.

2. Strojno i duboko učenje

Algoritmi strojnog učenja koriste skup podataka za treniranje kako bi izgradili model koji može prepoznati skrivene uzorke, pravilnosti i ovisnosti u podatcima i na temelju toga raditi predviđanja na novim podatcima koje nije vidio tijekom treniranja [12]. Vrste strojnog učenja, ovisno o tome kakvog su oblika podatci i što model treba naučiti na temelju tih podataka, prikazane su u tablici ispod.

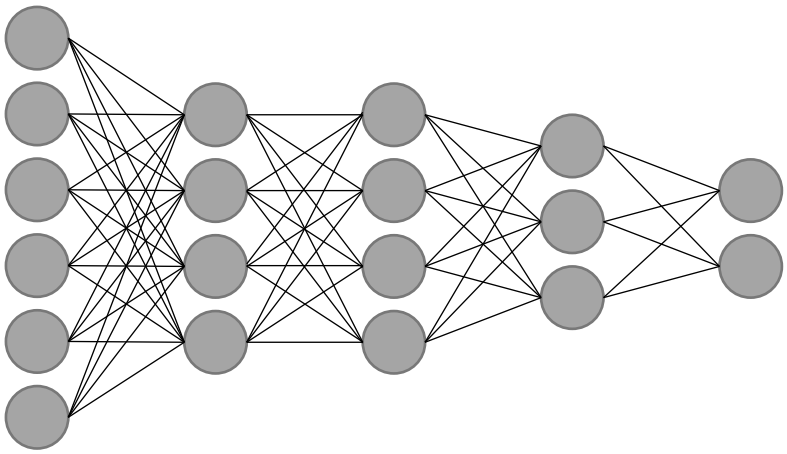
Tablica 3 Vrste strojnog učenja [13]

Nadzirano učenje **(engl. *supervised learning*)**	Model radi s označenim podatcima, odnosno cilj je na temelju ulaznih podataka predvidjeti točne izlazne podatke. Ovisno o tome kakvog su oblika izlazni podatci, nadzirano učenje se dijeli na klasifikaciju (izlazni podatci su diskretne vrijednosti, npr. oznake klase) i regresiju (izlazni podatci su kontinuirane brojčane vrijednosti).
Nenadzirano učenje **(engl. *unsupervised learning*)**	Model radi s neoznačenim podatcima, a cilj je prepoznati određene sličnosti i skrivene uzorke u podatcima. Najčešće se radi o postupcima grupiranja.
Polunadzirano učenje **(engl. *semi-supervised learning*)**	Model koristi djelomično označene podatke. S obzirom na to da često nije moguće pronaći dovoljno označenih podataka za treniranje, označeni i neoznačeni podatci koriste se zajedno kako bi model bio što uspješniji u predikciji.
Samonadzirano učenje **(engl. *self-supervised learning*)**	Ovaj je pristup sličan nenadziranom učenju po tome što model dobiva neoznačene podatke, međutim model sam iz podataka određuje oznake, odnosno izlaze koje uči predvidjeti tijekom treniranja.
Podržano učenje **(engl. *reinforcement learning*)**	Model se ponaša kao agent koji djeluje unutar okoliša ili konteksta. Ponašanje modela se usmjerava nagradama i kaznama. Primjeri korištenja ove vrste učenja mogu se naći u robotici i sustavima autonomne vožnje.

U ovome se radu za rješavanje problema poravnanja sljedova koristi nadzirano učenje. Ulazni podatci su neporavnati sljedovi. Model na temelju ulaza generira izlaz, odnosno poravnanje dva slijeda koje je dobio na ulazu. Model uči generirati ispravna poravnanja pomoću referentnih izlaza, odnosno optimalnih poravnanja koja su unaprijed generirana korištenjem algoritama dinamičkog programiranja, o čemu će više riječi biti u nastavku rada.

Nedostatak tradicionalnih algoritama strojnog učenja je to što uspješnost modela ovisi o tome kako su podatci predstavljeni modelu, odnosno kojim značajkama (engl. *features*). Značajke su tipično ručno definirane od strane stručnjaka s domenskim znanjem te nije jednostavno odrediti koje su značajke relevantne za model i za problem koji se rješava [12], [14].

Modeli dubokog učenja nadilaze prethodno spomenuta ograničenja zato što mogu raditi sa sirovim, neobrađenim podatcima (engl. *raw data*) i mogu samostalno otkriti odgovarajuće značajke, odnosno pretvoriti neobrađene ulazne podatke u odgovarajuće reprezentacije pogodne za daljnju obradu. Modeli dubokog učenja općenito se sastoje od većeg broja nelinearnih slojeva. Svaki od tih slojeva transformira reprezentaciju jedne razine (reprezentaciju koja je dobivena kao izlaz prethodnog sloja) u reprezentaciju više, apstraktnije razine. Početna razina su upravo neobrađeni ulazni podatci koji se slijedom velikog broja nelinearnih transformacija pretvaraju u vrlo složene reprezentacije. Takve složene reprezentacije olakšavaju detekciju skrivenih uzoraka, pravilnosti i ovisnosti u podatcima koje nisu vidljive u početnim, neobrađenim podatcima [14].



Slika 1 Primjer duboke neuronske mreže s 3 skrivena sloja. Ulazni podatci prolaze kroz 3 sloja nelinearnih transformacija nakon čega se dobiva dvodimenzionalni izlaz modela.

Duboko učenje je područje koje iznimno brzo napreduje. Do danas je razvijen velik broj različitih arhitektura dubokih modela, a primjeri osnovnih arhitektura prikazani su u tablici ispod.

Tablica 4 Primjeri arhitektura dubokog učenja [12], [15], [16]

Višeslojni perceptron (engl. *multilayer perceptron*)	Jednostavna arhitektura koja se sastoji od ulaznog i izlaznog sloja te jednog ili više skrivenih slojeva. Unutar svakog skrivenog sloja ulaz koji je dobiven od prethodnog sloja množi se s težinama te se na njega primjenjuje nelinearna aktivacijska funkcija.
Konvolucijska neuronska mreža (engl. *convolutional NN, CNN*)	Specifični slojevi koje sadrži ova arhitektura su konvolucijski sloj i sloj sažimanja (engl. *pooling layer*). Konvolucijski sloj računa korelaciju između jezgre (engl. *kernel*) i pojedinih dijelova ulaza čime filtrira bitne informacije iz ulaza. Sloj sažimanja obavlja agregaciju lokalno povezanih značajki, npr. računa njihovu prosječnu vrijednost ili odabire maksimalnu vrijednost kao izlaz.
Povratna neuronska mreža (engl. *recurrent NN, RNN*)	Ova arhitektura sadrži povratne veze, odnosno neurone čiji izlaz utječe na njihov vlastiti ulaz. To omogućuje pamćenje stanja, odnosno pamćenje prethodno obrađenih podataka s ulaza o kojima ovisi trenutni izlaz. Koristi se za obradu slijednih vremenski ovisnih podataka koji nisu fiksne duljine, poput prirodnog jezika ili zvučnih zapisa.
Arhitekture sa slojevima pažnje	Ove ahitekture sadrže slojeve pažnje koji omogućuju da se mreža prilikom učenja fokusira na relevantne dijelove ulaza i detektira ovisnosti između pojedinih elemenata ulaza. Primjer je upravo arhitektura *transformer*.
Arhitekture generativnih modela	Autoenkoderi koriste kodiranje kako bi ulaz pretvorili u reprezentaciju smanjene dimenzije u kojoj su pohranjene bitne značajke ulaza, a dekodiranjem te reprezentacije stvaraju što vjerniju repliku ulaza. Ostali primjeri uključuju duboke mreže vjerovanja (engl. *deep belief networks*) te generativne suprotstavljene mreže (engl. *generative adversarial networks, GAN*).

3. Arhitektura *transformer*

Arhitektura transformer predstavljena je 2017. godine u radu *Attention is all you need* [9]. Iako je originalno korištena za problem prevođenja prirodnog jezika, pokazala se uspješnom u rješavanju velikog broja raznolikih zadataka. Tablica 5 prikazuje primjere *transformer* modela iz različitih područja.

Tablica 5 Primjeri *transformer* modela

PODRUČJE	IME	OPIS MODELA
Obrada prirodnog jezika (NLP)	BERT	Predtreniran za razumijevanje prirodnog jezika. Može se fino podesiti za različite zadatke NLP-a dodavanjem specifičnih slojeva, npr. za klasifikaciju teksta [17].
	GPT	Predtreniran za generiranje novog teksta na temelju prethodno dobivenog teksta. Koristi se u poznatoj platformi ChatGPT [18].
	T5	Predtreniran za probleme pretvaranja teksta iz jednog oblika na ulazu u drugi oblik na izlazu, npr. može se koristiti za zadatke prevođenja ili sažimanja teksta [19].
Računalni vid	ViT	*Vision Transformer*, koristi se za klasifikaciju slika [20].
	DeiT	*Data-efficient Image Transformer*, također se koristi za klasifikaciju slika, ali zahtijeva manje podataka i resursa za treniranje u odnosu na ViT model [21].
Analiza zvuka	AST	*Audio Spectrogram Transformer*, obavlja klasifikaciju zvučnih zapisa na temelju spektrograma [22].
Bioinformatika	ProteinBERT	Sličan modelu BERT, ali predtreniran na velikom broju proteinskih sljedova [23].
	AlphaFold 2	Koristi se za predviđanje trodimenzionalne strukture proteina [24].
	BetaAlign	Koristi se za poravnanje bioloških sljedova [11].

Osnovni dijelovi koji čine arhitekturu *transformer* su koder i dekoder. Neki se modeli sastoje isključivo od kodera, neki isključivo od dekodera dok neki modeli imaju i koder i dekoder, ovisno o tome za kakav su zadatak namijenjeni. Različiti tipovi arhitekture i zadatci za koje se primjenjuju prikazani su u tablici ispod (Tablica 6).

Tablica 6 Različiti tipovi arhitekture *transformer* [25]

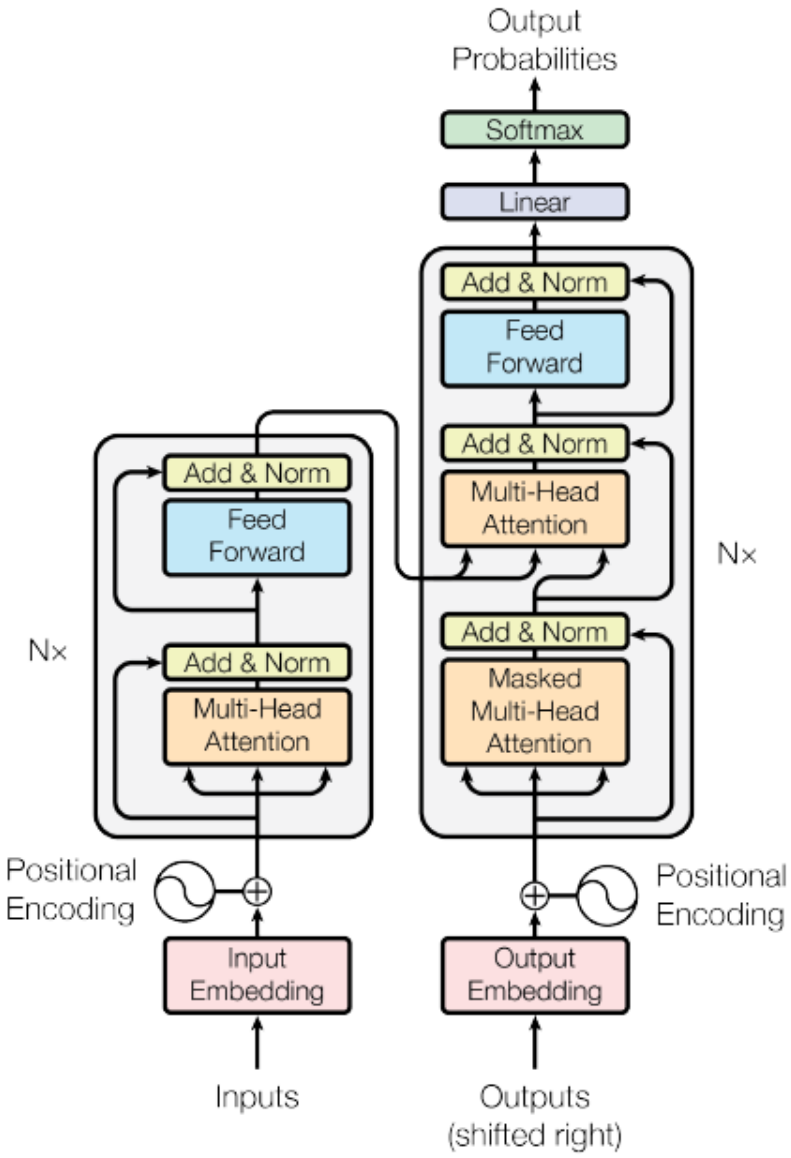
Koder	Zadatci koji zahtijevaju dobro razumijevanje svojstava ulaza, poput klasifikacije teksta ili prepoznavanje entiteta (engl. *named-entity recognition*). Primjer je model BERT.
Dekoder	Generativni zadatci, poput automatskog generiranja novog teksta ili predviđanja sljedeće riječi u rečenici. Primjer je model GPT.
Koder i dekoder	Zadatci transdukcije sljedova, odnosno pretvorbe ulaznog slijeda jednog oblika u izlazni slijed drukčijeg oblika i značenja. To može uključivati prevođenje prirodnog jezika ili sažimanje teksta. Primjer je model T5.

Slika 2 prikazuje arhitekturu *transformer* koja sadrži i koder i dekoder. U nastavku će biti opisani detalji arhitekture *transformer*, počevši od središnje ideje, mehanizma pažnje, pa do pojedinih slojeva koji se nalaze unutar kodera i dekodera. Potpoglavlja u nastavku (3.1, 3.2, 3.3, 3.4, 3.6) prate opise originalne *transformer* arhitekture (tzv. *vanilla transformer*) u literaturi [9], [25], [26] i [27].

3.1. Mehanizam pažnje

Mehanizam pažnje (engl. *self-attention*) omogućuje *transformeru* da prilikom računanja reprezentacije svakog elementa slijeda (ti se elementi mogu zvati tokeni, ili riječi ako se radi o prirodnom jeziku) uzima u obzir povezanost tog elementa sa ostalim dijelovima slijeda. Na taj način model može detektirati različite ovisnosti između elemenata slijeda.

Izlaz sloja pažnje računa se korištenjem 3 vektora koji su dobiveni iz svakog tokena zasebno. To su upit (engl. *query*), ključ (engl. *key*) i vrijednost (engl. *value*). Neka njihove oznake budu $\mathbf{q}_i$, $\mathbf{k}_i$ i $\mathbf{v}_i$, to su dakle vektori koji predstavljaju token *i*.



Slika 2 Primjer arhitekture *transformer* s koderom i dekoderom [9]

Takvi se vektori dobiju množenjem vektorske reprezentacije svakog tokena s tri matrice, matricom upita, matricom ključeva i matricom vrijednosti. Parametri tih matrica ugađaju se treniranjem modela. Vrijednost pažnje za token *i* računa se kao otežana suma vektora **v** svih tokena. Težina kojom se prilikom izračuna pažnje za token *i* množi svaki od vektora $\mathbf{v}_j$ računa se kao skalarni produkt vektora $\mathbf{q}_i$ te vektora $\mathbf{k}_j$. Svaka se težina zbog stabilnosti gradijenata dodatno dijeli s korijenom dimenzije ključeva, odnosno s $\sqrt{d_{key}}$, nakon čega se primjenjuje *softmax* funkcija kako bi se sve težine sumirale u 1. U praksi se za izračun pažnje koristi matrično množenje. Matrica **X** sadrži vektorske reprezentacije ulaznih tokena, a matrice $\mathbf{W}^{\mathbf{Q}}$, $\mathbf{W}^{\mathbf{K}}$ i $\mathbf{W}^{\mathbf{V}}$

koriste se za izračun matrica **Q**, **K** i **V** koje sadrže upite, ključeve i vrijednosti svih tokena. Izrazi (1), (2) i (3) koriste se za izračun matrica **Q**, **K** i **V**, a izraz (4) pokazuje kako se računa izlaz sloja pažnje za sve tokene.

$$\mathbf{Q} = \mathbf{X} \cdot \mathbf{W}^{\mathbf{Q}} \tag{1}$$

$$\mathbf{K} = \mathbf{X} \cdot \mathbf{W}^{\mathbf{K}} \tag{2}$$

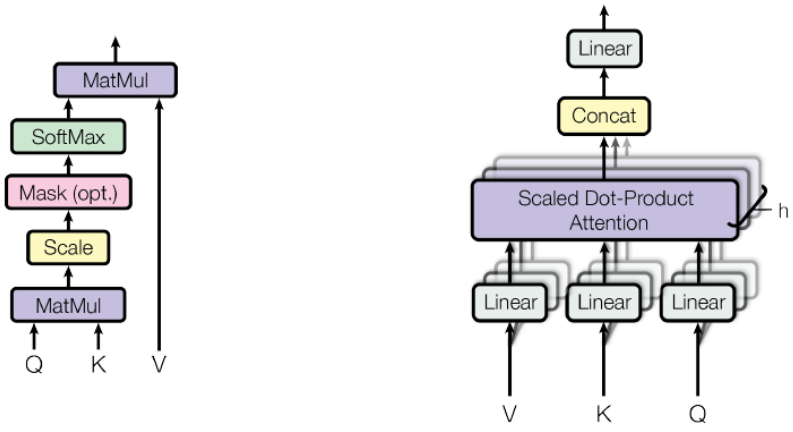
$$\mathbf{V} = \mathbf{X} \cdot \mathbf{W}^{\mathbf{V}} \tag{3}$$

$$Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax\left(\frac{\mathbf{Q} \cdot \mathbf{K}^{\mathrm{T}}}{\sqrt{d_{key}}}\right) \cdot \mathbf{V} \tag{4}$$

Prethodno je opisana situacija u kojoj se koristi jedna glava koja računa jednostruku pažnju. Sloj pažnje općenito može raditi s više glava, pri čemu svaka glava *h* ima svoje matrice $\mathbf{W^Q_h}$, $\mathbf{W^K_h}$ i $\mathbf{W^V_h}$ te računa svoju funkciju pažnje. Ovaj se mehanizam naziva pažnja s više glava (engl. *multi-head attention*). Rezultati glava se zatim konkateniraju u zajedničku matricu koja se množi s matricom $\mathbf{W^0}$, čiji se parametri također ugađaju tijekom treniranja modela. Izraz (5) pokazuje rezultat pažnje jedne glave, dok izraz (6) pokazuje kako se računa ukupna pažnja uzevši u obzir sve glave. Izračun pažnje prikazan je i na slici ispod.

$$Head_h = Attention\left(\mathbf{Q}_{\mathrm{h}} = \mathbf{X} \cdot \mathbf{W}_{\boldsymbol{h}}^{\boldsymbol{Q}}, \mathbf{K}_{\mathrm{h}} = \mathbf{X} \cdot \mathbf{W}_{\boldsymbol{h}}^{\boldsymbol{K}}, \mathbf{V}_{\mathrm{h}} = \mathbf{X} \cdot \mathbf{W}_{\boldsymbol{h}}^{\boldsymbol{V}}\right) \tag{5}$$

$$MultiHeadAttention = Concat(Head_1, \ldots, Head_n) \cdot \mathbf{W^0} \tag{6}$$



Slika 3 Izračun vrijednosti pažnje [9]. Lijevo je prikazan izračun pažnje korištenjem jedne glave. Desno je prikazan izračun pažnje korištenjem više glava čiji se rezultati konkateniraju i zatim pretvaraju u konačan rezultat množenjem s matricom $\mathbf{W^0}$ (linearna projekcija).

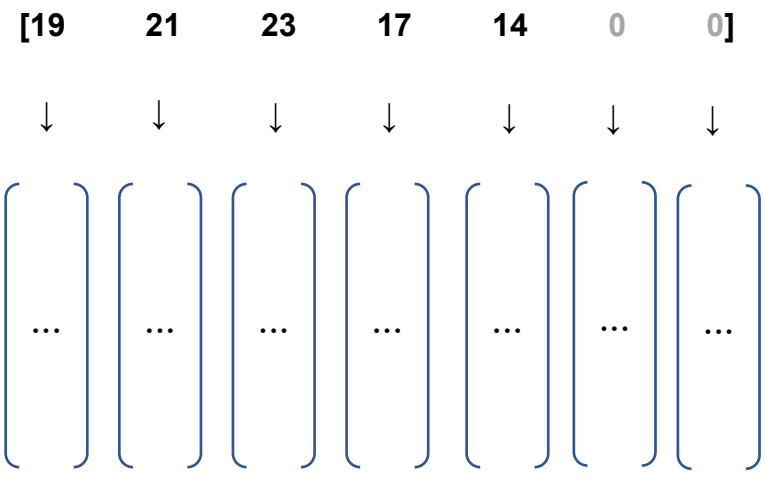
3.2. Ugrađivanje ulaza

Ugrađivanje (engl. *embedding*) je postupak koji pretvara slijed simbola u brojčanu reprezentaciju. Prvi korak je rastavljanje slijeda na tokene i zamjena svakog tokena njegovom jednoznačnom brojčanom oznakom. Može se jednostavno koristiti redni broj tokena u rječniku koji sadrži sve moguće tokene koji se mogu pojaviti u slijedu, ali postoje i druge metode. Slijed se tako pretvara u niz brojeva koji se dopunjuje na unaprijed zadanu duljinu (može se označiti s N_{input}) dodavanjem posebnih tokena (engl. *padding*) ili skraćuje odbacivanjem nekih tokena, s obzirom na to da *transformer* očekuje da ulaz bude fiksne duljine zbog kasnijih matričnih operacija.

Ako pretpostavimo da nam je ulaz kratki proteinski slijed, tokeni su pojedine aminokiseline, a *transformer* očekuje ulaz duljine sedam tokena, postupak pretvorbe ulaza u tokene izgleda ovako:

M S T L Q → [19 21 23 17 14 0 0]

Svaka aminokiselina ima svoju brojčanu oznaku, a posljednje dvije oznake predstavljaju dopunu za koju je određeno da se označava vrijednošću 0. Sljedeći korak je pretvorba svakog tokena, odnosno svake vrijednosti, u vektor unaprijed zadane duljine. Ta se duljina naziva dimenzija modela, d_{model}. Model tijekom treniranja samostalno pronalazi optimalan način pretvorbe tokena u vektore. Pretvorba tokena u vektore može se prikazati ovako:



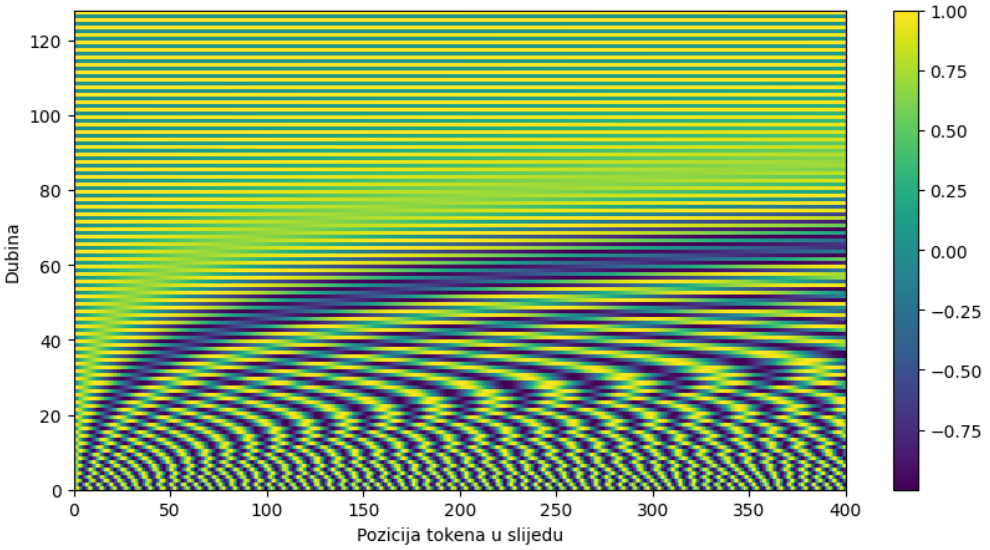
Ovaj način ugrađivanja modelu ne daje informaciju o redoslijedu tokena u ulaznom slijedu. Kada bi se ovako dobiveni vektori predali modelu na daljnju obradu, model bi ulaz shvaćao kao "vreću riječi" (engl. *bag of words*), odnosno ne bi bio sposoban razlikovati sljedove u kojima su potpuno isti tokeni, ali u različitom redoslijedu. Zato

je bitno da se u vektore reprezentacije pojedinih tokena na neki način dodatno ugradi informacija o položaju tokena unutar slijeda. Postoji više metoda kako se može napraviti pozicijsko kodiranje: pronalaskom optimalnog načina treniranjem modela, na isti način na koji se obavlja i prethodno opisano ugrađivanje ulaza, ili unaprijed definiranim algoritmom. U ovom se radu koristi drugi način, odnosno koriste se funkcije sinusa i kosinusa različitih frekvencija kako je opisano u literaturi [9]. Za računanje vektora pozicijskog kodiranja koriste se izrazi (7) i (8), pri čemu *pos* označava položaj tokena u ulaznom slijedu, a *i* određuje položaj u vektoru pozicijskog kodiranja, što se još naziva i dubinom. Vrijednost *i* kreće se u rasponu $\left[0, \frac{d_{model}}{2}\right]$. Izraz (7) određuje vrijednosti na parnim dubinama unutar vektora pozicijskog kodiranja, a izraz (8) određuje vrijednosti na neparnim dubinama.

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}}) \tag{7}$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}}) \tag{8}$$

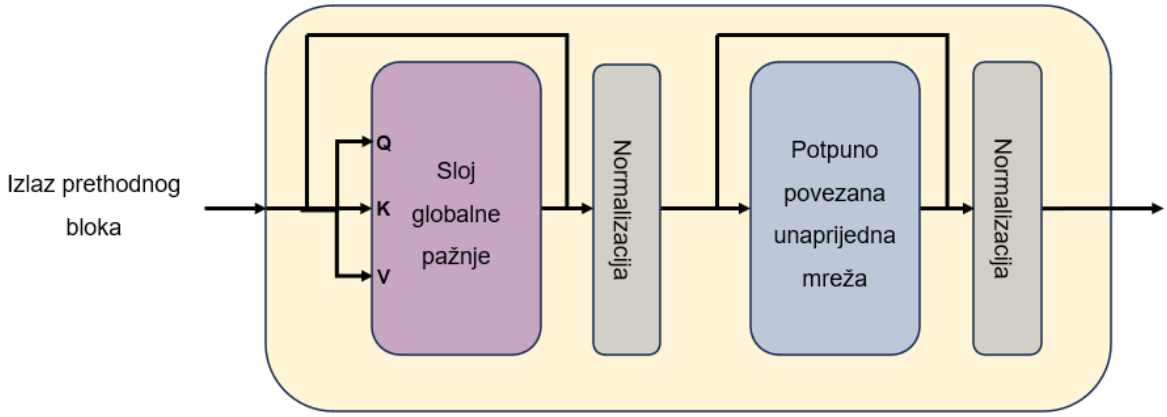
Na ovaj se način za svaki token računa vektor pozicijskog kodiranja koji je iste dimenzije kao i prethodno izračunati vektor reprezentacije tokena, odnosno dimenzije d_{model}. Vektori reprezentacije tokena dodatno se množe s faktorom $\sqrt{d_{key}}$ nakon čega se jednostavno zbrajaju s vektorima pozicijskog kodiranja čime se dobiva konačna vektorska reprezentacija ulaza te je ugrađivanje ulaza završeno.



Slika 4 Primjer pozicijskog kodiranja. Duljina ulaznog slijeda je 400 tokena, a dimenzija modela (maksimalna dubina) je 128. Različite boje prikazuju vrijednosti funkcija sinusa i kosinusa na različitim pozicijama i različitim dubinama.

3.3. Koder

Koder se sastoji od blokova koji imaju identičnu arhitekturu pri čemu je izlaz svakog bloka ulaz u sljedeći blok. Ulaz prvog bloka dobiven je postupkom ugrađivanja ulaznog slijeda koji je opisan u prethodnom potpoglavlju (3.2). Izlaz posljednjeg bloka predstavlja konačan izlaz kodera. Na slici ispod još jednom je prikazan blok kodera zajedno sa svim slojevima.



Slika 5 Prikaz jednog bloka kodera. Sastoji se od sloja globalne pažnje te jednostavne potpuno povezane unaprijedne mreže. Nakon svakog sloja primjenjuje se normalizacija uz rezidualnu vezu.

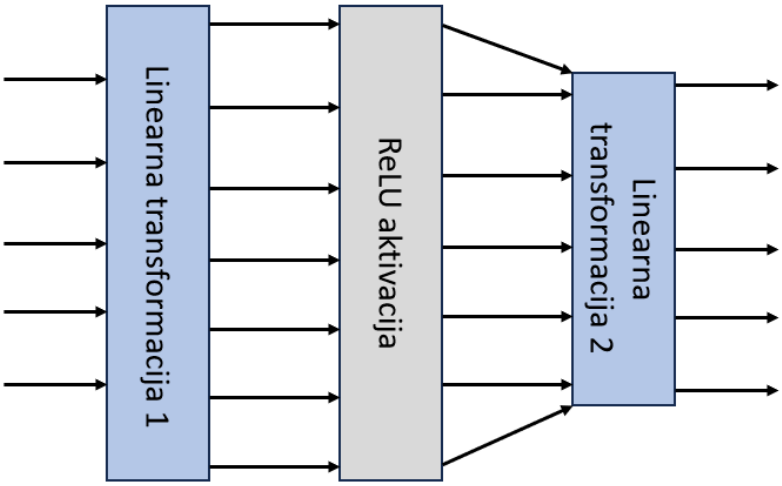
Prvi sloj u svakom bloku kodera je globalni sloj pažnje koji se može sastojati od više glava. Svi se ključevi, vrijednosti i upiti tog sloja dobivaju iz ulaza koji predstavlja izlaz prethodnog bloka. Ovaj sloj kodera omogućuje detekciju ovisnosti između pojedinih tokena ulaznog slijeda. Prilikom izračuna pažnje svaki token ima pristup svim ostalim pozicijama u slijedu, zbog čega se koristi naziv globalna pažnja.

Nakon sloja pažnje slijedi rezidualna veza koja se primjenjuje nakon svakog sloja u *transformeru*. Izlaz sloja zbraja se s njegovim ulazom. Ovdje je ključno to što su ulazi i izlazi svih slojeva u *transformeru* istih dimenzija, odnosno dimenzije modela d_{model} što pojednostavljuje njihovo zbrajanje. Rezidualna veza olakšava računanje gradijenata tijekom treniranja modela.

Nakon svakog sloja se uz rezidualnu vezu provodi i normalizacija. Normalizacija se općenito provodi kako bi se pojedine značajke svele na istu skalu, odnosno kako bi se spriječilo da jedna značajka magnitudama svojih vrijednosti odudara od ostalih

značajki. Postoji više načina na koje se može primijeniti normalizacija, a u ovoj se arhitekturi primjenjuje *layer normalization* postupak. Specifičnost ovog postupka je što se primjenjuje na sve značajke jednog primjera, neovisno o drugim primjerima. Drugim riječima, sve značajke istog primjera normaliziraju se istim parametrima (srednja vrijednost i standardna devijacija), ali različiti primjeri imaju međusobno različite parametre [28]. Druga mogućnost normalizacije je *batch normalization*.

Sljedeći sloj je unaprijedna potpuno povezana mreža s jednim skrivenim slojem nakon kojeg se primjenjuje ReLU aktivacijska funkcija. Dimenzija skrivenog sloja je jedan od hiperparametara *transformer* modela, a u ovom radu će se zvati latentna dimenzija, d_{latent}. Slika 6 detaljnije prikazuje ovaj sloj.

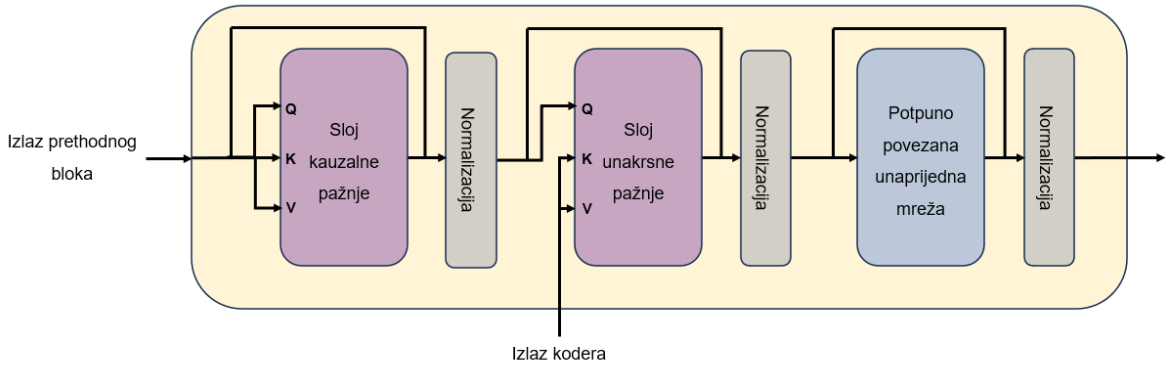


Slika 6 Prikaz unaprijedne mreže unutar *transformera*. Prva linearna transformacija pretvara ulaz iz dimenzije d_{model} u dimenziju d_{latent} nakon čega se primjenjuje ReLU aktivacijska funkcija. Druga linearna transformacija na ulaz dobiva vektor dimenzije d_{latent} te računa izlaz koji je ponovno dimenzije d_{model}.

Nakon unaprijedne potpuno povezane mreže opet se na isti način kao i prije primjenjuju rezidualne veze uz normalizaciju. Nakon toga se dobiva konačan izlaz bloka koji se predaje na ulaz sljedećem bloku. Izlaz posljednjeg bloka predstavlja konačan izlaz kodera, apstraktnu reprezentaciju ulaznog slijeda u kojoj su pohranjene bitne značajke. Ta se reprezentacija može koristiti za različite zadatke klasifikacije ili se može predati dekoderu koji će koristiti izlaz enkodera kao kontekst za generiranje novih sljedova.

3.4. Dekoder

Dekoder je vrlo sličan koderu, ali ima nešto složeniju arhitekturu. Uloga dekodera je generiranje smislenog slijeda tokena. Tokeni se na izlazu generiraju ovisno o kontekstu (izlazu kodera) i svim prethodno generiranim tokenima. Dekoder se, kao i koder, sastoji od više blokova identične arhitekture. Ulaz u prvi blok je slijed svih prethodno generiranih tokena na izlazu dekodera koji su obrađeni postupkom ugrađivanja (3.2). Izlaz posljednjeg bloka dekodera predstavlja sljedeći token u slijedu. Generiranje izlaza odvija se u više koraka, pri čemu se u svakom koraku generira jedan token koji se nastavlja na prethodno generirane tokene. Slika 7 prikazuje jedan blok dekodera zajedno sa svim slojevima.



Slika 7 Prikaz jednog bloka dekodera. Sastoji se od dva sloja pažnje te jednostavne potpuno povezane unaprijedne mreže. Nakon svakog sloja primjenjuje se normalizacija uz rezidualnu vezu.

Razlika između bloka kodera i bloka dekodera je u slojevima pažnje. Blok dekodera ima dva sloja pažnje od kojih se, kao i u koderu, svaki može sastojati od više glava.

Prvi sloj pažnje omogućuje detekciju ovisnosti između različitih pozicija u ulazu, s time da ulaz u dekoder predstavljaju svi prethodno generirani tokeni. Pažnja svakog tokena računa se isključivo pomoću prethodnih pozicija u slijedu, zbog čega se ovaj sloj naziva sloj kauzalne pažnje.

Drugi sloj unakrsne pažnje predstavlja vezu između kodera i dekodera. Upiti za računanje pažnje dobivaju se na temelju izlaza prethodnog sloja dok se ključevi i vrijednosti dobivaju iz izlaza kodera, odnosno konteksta. Ovaj sloj omogućuje detekciju ovisnosti između generiranih tokena i različitih pozicija u kontekstu,

odnosno u ulaznom slijedu kodera. S obzirom na to da pažnja u ovom sloju povezuje ulaz kodera i ulaz dekodera, naziva se unakrsnom pažnjom.

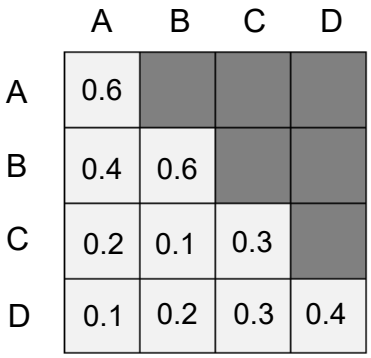
Posljednji sloj bloka je, kao i u koderu, unaprijedna potpuno povezana neuronska mreža. Nakon svakog sloja u bloku primjenjuju se rezidualna veza i normalizacija.

3.5. Maskiranje

U svim slojevima pažnje u *transformeru* primjenjuje se maskiranje pojedinih pozicija čime se sprječava da takve pozicije sudjeluju u izračunu pažnje. U tablici ispod objašnjene su dvije vrste maskiranja koje se primjenjuju u *transformeru*.

Tablica 7 Vrste maskiranja u *transformeru*

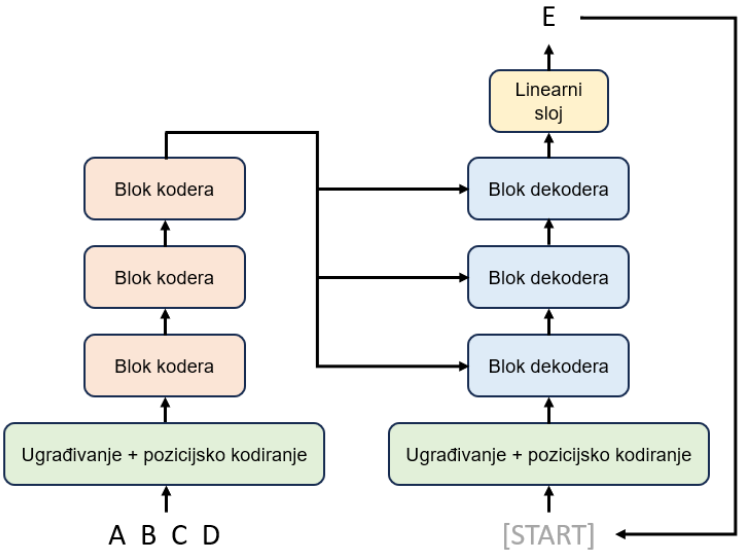
Maskiranje dopune (engl. *padding mask*)	Maskiraju se pozicije na kojima se nalazi *padding* u ulazima kodera i dekodera, s obzirom na to da takve pozicije ne nose nikakvo korisno značenje. Primjenjuje se u svim slojevima pažnje u *transformeru*.
Kauzalno maskiranje (engl. *look-ahead mask*, *causal mask*)	Ova vrsta maskiranja primjenjuje se u sloju kauzalne pažnje u dekoderu. Svaki token koji se generira na izlazu mora ovisiti isključivo o tokenima koji dolaze prije njega. Prilikom izračuna pažnje određenog tokena potrebno je maskirati pozicije koje dolaze nakon tog tokena kako ne bi utjecale na izračun njegove pažnje.



Slika 8 Prikaz kauzalnog maskiranja. Brojevi predstavljaju rezultat skalarnog produkta vektora upita i ključeva različitih parova tokena prilikom izračuna pažnje. To je mjera povezanosti dva tokena u slijedu. Token koji se generira na izlazu dekodera mora ovisiti samo o prethodnim tokenima pa ostale pozicije moraju biti maskirane (obojano sivo).

3.6. Završni linearni sloj

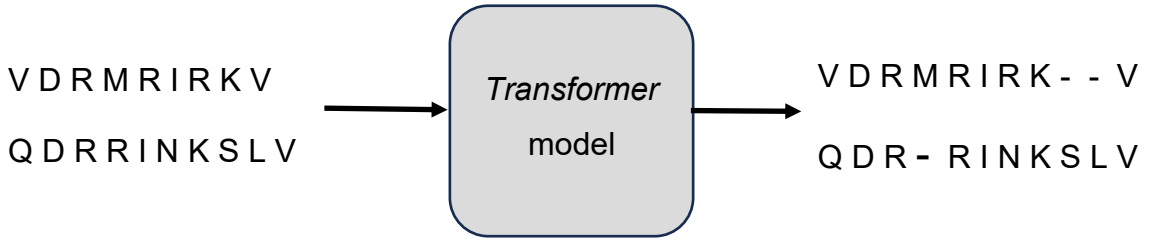
Generiranje novog slijeda odvija se korak po korak, u svakom se koraku generira sljedeći token u slijedu koji ovisi o prethodnim tokenima. Generirani token na izlazu dekodera predstavljen je vektorom čija je dimenzija jednaka dimenziji modela, d_{model}. Taj je vektor na neki način potrebno pretvoriti u konkretan simbol, npr. konkretnu riječ ili konkretnu aminokiselinu. Iz tog se razloga na izlaz *transformera* dodaje još jedan sloj, jednostavna potpuno povezana unaprijedna mreža koja vektor dimenzije d_{model} pretvara linearnom transformacijom u vektor čija je dimenzija jednaka veličini izlaznog rječnika. Izlazni rječnik je popis svih mogućih tokena koji se mogu pojaviti na izlazu modela, odnosno koje dekoder može generirati. Na primjer, ako se radi o prevođenju jezika, to će biti popis svih mogućih riječi jezika na koji se prevodi ulazna rečenica. U vektoru takve dimenzije pohranjene su vrijednosti koje za svaki token u rječniku govore kolika je vjerojatnost da je upravo to sljedeći token koji treba generirati na izlazu. Odabire se onaj token koji ima najveću vrijednost, odnosno najveću vjerojatnost. Ukratko, ovaj sloj određuje konkretan token koji najbolje odgovara apstraktnoj reprezentaciji koju je generirao dekoder.



Slika 9 Prikaz rada *transformer* modela. Model na slici sastoji se od 3 bloka kodera i dekodera. Na ulaz kodera dovodi se početni slijed koji se transformira u apstraktnu reprezentaciju, odnosno kontekst koji će se koristiti za generiranje novog slijeda. U svakom koraku generiranja novog slijeda generira se po jedan novi token. Tako nadopunjeni novi slijed ponovno se dovodi na ulaz dekodera. Na početku procesa dekodiranja novi slijed sadrži samo posebni START token koji označava početak slijeda.

4. Primjena *transformer* arhitekture za poravnanje sljedova

Zadatak poravnanja sljedova može se promatrati kao zadatak prevođenja sljedova iz neporavnatog oblika u poravnati oblik. S obzirom na to da se arhitektura *transformer* s koderom i dekoderom pokazala vrlo uspješnom u zadatcima prevođenja prirodnog jezika, postoji intuicija da se može koristiti i za poravnanje sljedova. Ulaz u *transformer* model je par neporavnatih proteinskih sljedova, a izlaz je isti par sljedova koji su poravnati, kao što prikazuje Slika 10.



Slika 10 Princip poravnanja sljedova korištenjem *transformer* modela

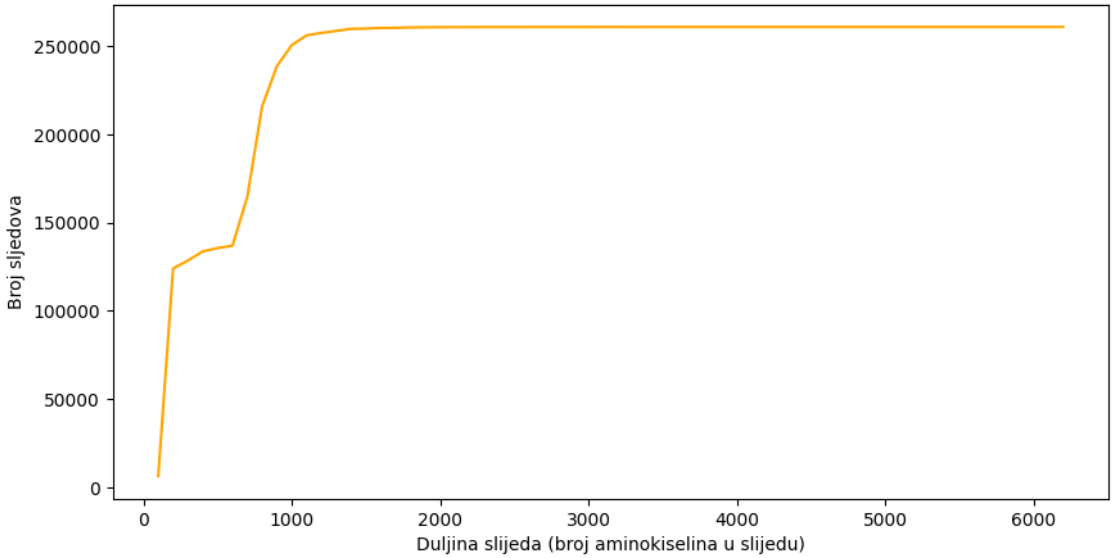
U literaturi već postoji primjer korištenja *transformera* za problem poravnanja sljedova [11]. Dotan et al. razvili su BetaAlign, tehniku za višestruko poravnanje sljedova koja koristi *transformer.* Model su trenirali na nekoliko milijuna simuliranih poravnanja, a rezultati koje su dobili bili su usporedivi s rezultatima najpopularnijih algoritama za poravnanje, a u nekim slučajevima i bolji.

U ovome je radu napravljena vlastita programska implementacija *transformer* modela za poravnanje parova proteinskih sljedova (engl. *pairwise alignment*). Modeli različitih arhitektura trenirani su na skupu unaprijed pripremljenih referentnih poravnanja uzevši u obzir različite mogućnosti reprezentacije sljedova. Napravljena je usporedba vremena potrebnog za treniranje modela te dobivenih rezultata. Sljedeća poglavlja opisuju praktični dio rada, detalje skupa sljedova koji su korišteni za treniranje i testiranje modela te detalje programske implementacije.

5. Priprema skupa podataka

5.1. Priprema referentnih poravnanja

Za potrebe treniranja i testiranja modela korišteni su proteinski sljedovi hemoglobina preuzeti sa stranice američkog centra za biotehnološke informacije (NCBI) [29]. Skup je sadržavao ukupno 260 765 sljedova hemoglobina koji su se bitno razlikovali u svojoj duljini, odnosno broju aminokiselina. Najkraći sljedovi su imali duljinu manju od 100 aminokiselina, dok je najdulji slijed imao duljinu od 6241 aminokiseline.



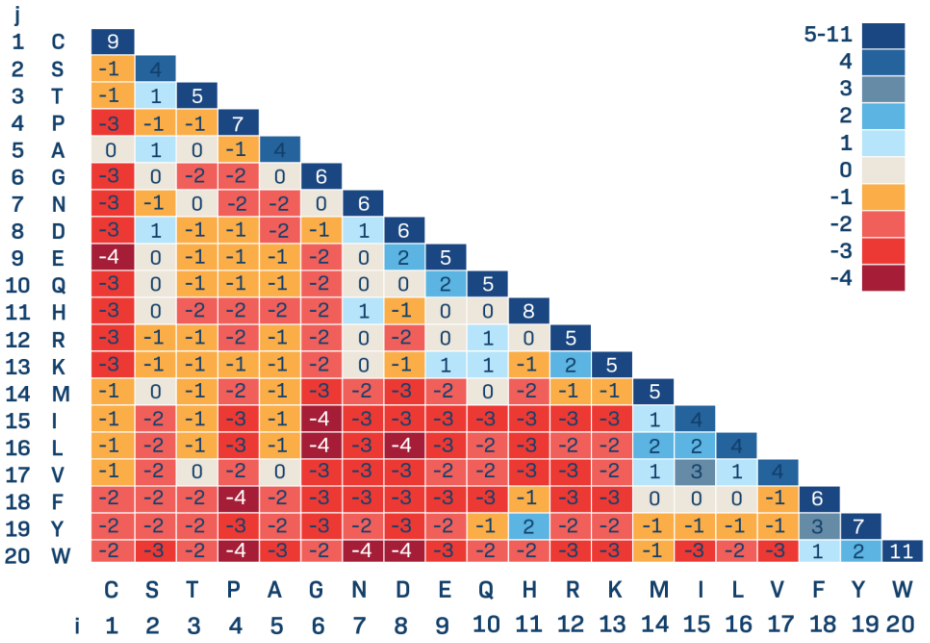
Graf 1 Prikaz duljine sljedova hemoglobina. Na y-osi prikazan je broj sljedova hemoglobina čija je duljina manja ili jednaka duljini navedenoj na x-osi. Iz grafa se može vidjeti kako je duljina većine sljedova manja od 1000 aminokiselina.

Iz skupa svih sljedova hemoglobina odabran je podskup sljedova čija je duljina između 100 i 200 aminokiselina. Sljedovi kraćih duljina odabrani su kako bi treniranje i testiranje modela bilo što brže i efikasnije jer povećanje duljine ulaza u *transformer* očekivano usporava proces generiranja izlaza. Takav podskup sadrži ukupno 117 723 slijeda što čini 45.16% početnog skupa sljedova. Sljedeći korak je uzimanje parova sljedova i određivanje njihovog referentnog poravnanja koje će se koristiti za treniranje modela i računanje točnosti. Jedan primjer u skupu podataka predstavlja jedan par neporavnatih sljedova, a odgovarajući izlaz je par istih

poravnatih sljedova. Za određivanje referentnih poravnanja korištena je Pythonova biblioteka Biopython [30]. Referentna poravnanja dobivena su korištenjem algoritma Needleman-Wunsch [31] Korišteni su sljedeći parametri:

- kazna za otvaranje procjepa (engl. *gap opening penalty*): -5
- kazna za nastavak procjepa (engl. *gap extension penalty*): -1
- substitucijska matrica (engl. *substitution matrix*): BLOSUM62

BLOSUM62 je matrica koja se koristi za računanje ocjene poravnanja dva slijeda, a osmišljena je za proteinske sljedove čija sličnost nije veća od 62% [32].



Slika 11 BLOSUM62 supstitucijska matrica [40]. Vrijednosti predstavljaju vjerojatnost supstitucije jedne aminokiseline drugom, a koriste se za vrednovanje poravnanja.

Iz skupa kratkih sljedova hemoglobina može se dobiti gotovo 7 milijardi parova sljedova, međutim kako proces treniranja ne bi trajao predugo, za potrebe ovog rada na slučajan je način izabrano 100 000 parova za treniranje modela. Prilikom treniranja 10% podataka izdvaja se za validaciju (u ovom slučaju to je 10 000 parova). Skup za validaciju koristi se kako bi se provjerila točnost modela na podatcima na kojima model nije učio, odnosno kako bi se provjerila sposobnost generalizacije modela.

5.2. Načini reprezentacije sljedova

Prvi korak u radu *transformera* je pretvorba ulaznog slijeda u niz tokena. Postoji više načina na koje se to može napraviti, stoga je potrebno odabrati odgovarajući oblik u kojem će se sljedovi dovoditi na ulaz *transformera*, odnosno odgovarajući način reprezentacije.

Dotan et al. u svom radu predlažu nekoliko različitih reprezentacija ulaza (neporavnati par sljedova) i izlaza (poravnati par sljedova) [11]. U ovom se radu koristi jedan način reprezentacije za ulaz, konkatenacija sljedova, te dva različita načina reprezentacije izlaza koja će se zvati parovi i razmaci. Svaka reprezentacija ima svoj rječnik, odnosno popis svih mogućih tokena koji se mogu pojaviti u slijedu takve reprezentacije. Osim tokena vezanih uz konkretnu reprezentaciju, rječnici sadrže i posebne tokene koji su objašnjeni u tablici ispod.

Tablica 8 Posebni tokeni koje sadrži rječnik svake reprezentacije

PAD	Dopuna, odnosno *padding*.
MASK	Maskirane pozicije u slijedu.
UNK	Oznaka nepoznatog tokena. Ako se u slijedu pojavi token kojeg nema u rječniku, zamjenjuje se ovim tokenom.
START	Početak slijeda.
END	Kraj slijeda

Za ulazni slijed je korištena samo jedna vrsta reprezentacije, konkatenacija sljedova, s obzirom na to da je u radu Dotan et al. objašnjeno kako se korištenjem te reprezentacije postižu bolji rezultati nego korištenjem alternativnih načina. Za izlazni slijed se primarno koristi reprezentacija parova, zato što zahtijeva dvostruko manju duljinu slijeda nego reprezentacija razmaka. Modeli koji postižu najbolje rezultate korištenjem parova testirat će se i s reprezentacijom razmaka kako bi se dvije reprezentacije mogle usporediti.

Tablica 9 sadrži opise reprezentacija. Za svaku reprezentaciju je prikazan primjer kodiranja slijeda i navedeno je što sve sadrži njezin rječnik.

Tablica 9 Prikaz različitih načina reprezentacije sljedova

<table>
<tr><td rowspan="1">ULAZ</td><td>Konkatenacija sljedova</td><td>Dva se slijeda konkateniraju, a granicu predstavlja poseban token "|". Svaka aminokiselina predstavlja jedan token.

M S L F E
M N Q H T ⟶ M S L F E | M N Q H T

Rječnik ove reprezentacije sadrži ukupno 26 tokena: 20 mogućih aminokiselina, token granice "|" te 5 posebnih tokena opisanih u Tablici 8.</td></tr>
<tr><td rowspan="2">IZLAZ</td><td>Parovi</td><td>Tokeni su parovi, a jedan par predstavlja jedan stupac u dobivenom poravnanju, odnosno dvije aminokiseline koje se nalaze na istim pozicijama u dva slijeda.

W K - - V
W K S L V ⟶ WW KK -S -L VV

Rječnik ove reprezentacije sadrži ukupno 445 tokena: 400 parova aminokiselina, 40 parova u kojem je na jednom mjestu procjep te 5 posebnih tokena.</td></tr>
<tr><td>Razmaci</td><td>Ova je reprezentacija vrlo slična reprezentaciji pomoću parova, samo što su parovi razdvojeni na dva zasebna tokena tako da je, kao u slučaju konkatenacije sljedova, svaka aminokiselina zaseban token.

W K - - V
W K S L V ⟶ W W K K - S - L V V

Rječnik ove reprezentacije sadrži ukupno 26 tokena: 20 mogućih aminokiselina, procjep te 5 posebnih tokena.</td></tr>
</table>

6. Programska implementacija

Programska implementacija napravljena je pomoću programskog jezika Python i okvira TensorFlow i Keras. TensorFlow je platforma za strojno učenje otvorenog koda koju je razvio Google [33]. Keras je API visoke razine koji se temelji na okviru TensorFlow, odnosno sučelje koje apstrahira funkcije okvira TensorFlow i pojednostavljuje razvoj i treniranje dubokih modela [34]. Implementacija modela *transformer* i programski kod za treniranje modela napisani su uz pomoć uputa i primjera na službenim stranicama okvira TensorFlow [35].

6.1. Implementacija modela

Implementacija modela napravljena je po uzoru na originalnu *transformer* arhitekturu predstavljenu u radu *Attention is all you need*, koja je detaljno opisana u poglavlju 3. Model se sastoji od sloja ugrađivanja ulaza koji obavlja i pozicijsko kodiranje koje koristi funkcije sinusa i kosinusa, kodera i dekodera s proizvoljnim brojem blokova te završnog linearnog sloja.

Skup proteinskih sljedova koji se koristi u ovom radu odabran je tako da je najveća moguća duljina slijeda 200 aminokiselina. Za reprezentaciju ulaza koristi se konkatenacija dva slijeda. Ulaz osim dva slijeda sadrži i oznaku granice između dva slijeda te posebne START i END tokene. Slijedom navedenog najduži mogući slijed koji se može očekivati na ulazu kodera sadrži 403 tokena. *Transformer* očekuje da ulaz uvijek bude fiksne duljine pa se duljina ulaza, N_{input}, postavlja na 403 tokena. Ulazni slijed koji je kraći od navedene duljine nadopunjuje se *padding* tokenima. Duboki modeli su uglavnom implementirani tako da na ulazu očekuju grupe primjera (engl. *batch*) pa je dimenzija tenzora koji *transformer* očekuje na ulazu sljedeća:

$$(N_{batch}, N_{input} = 403)$$

Nakon sloja ugrađivanja dimenzija tenzora postaje ovakva:

$$(N_{batch}, N_{input} = 403, d_{model})$$

Tenzor ostaje ovakvih dimenzija tijekom cijelog procesa kodiranja, zato što su ulazi i izlazi svih blokova kodera te ulazi i izlazi svih slojeva unutar bloka jednaki.

Na ulaz dekodera dovodi se tenzor koji predstavlja prethodno generirani izlaz. Ako se za reprezentaciju izlaza koriste parovi, najveća moguća duljina koja se može očekivati na ulazu dekodera iznosi 402 tokena (ako se uzmu u obzir START i END tokeni). To se dogodi u slučaju u kojemu je svaka aminokiselina sparena s procjepom. Ako se za reprezentaciju izlaza koriste razmaci, najveća moguća duljina koja se u teoriji može očekivati na ulazu dekodera je 802 tokena, a to se ponovno odnosi na situaciju u kojoj je svaka aminokiselina sparena s procjepom. U svakom slučaju dimenzija tenzora na ulazu dekodera je:

$$(N_{batch}, N_{output})$$

Nakon ugrađivanja tenzor poprima sljedeće dimenzije, a iste dimenzije ima i izlaz dekodera:

$$(N_{batch}, N_{output}, d_{model})$$

Konačan izlaz dekodera dobije se nakon primjene završnog linearnog sloja, a dimenzije tog izlaza su:

$$(N_{batch}, N_{output}, N_{output\_vocab})$$

Drugim riječima, model za svaku poziciju u izlaznom slijedu vraća vektor čija je dimenzija veličina rječnika izlaza. To je vektor vjerojatnosti koji govori za koji je token najvjerojatnije da se nalazi na upravo toj poziciji. Odabire se onaj token koji ima najveću vrijednost, odnosno vjerojatnost.

U tablici ispod navedeni su hiperparametri koji određuju arhitekturu *transformer* modela te vrijednosti koje su odabrane za izgradnju modela u ovom radu.

Tablica 10 Hiperparametri koji određuju arhitekturu *transformer* modela. Posljednji stupac sadrži vrijednosti hiperparametara koje su korištene u ovom radu.

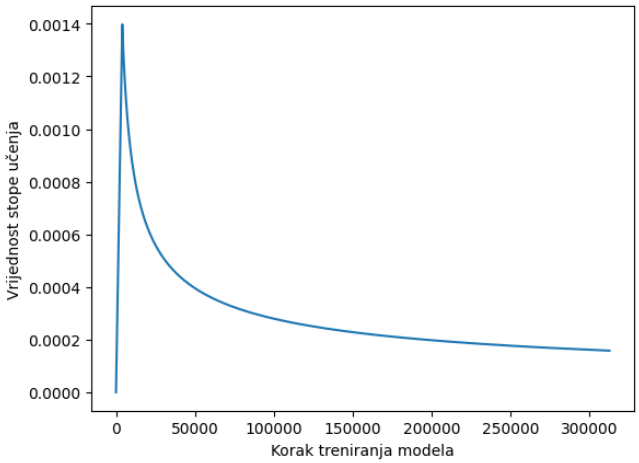
d_{model}	Dimenzija modela	128 i 256
d_{latent}	Dimenzija skrivenog sloja unaprijedne mreže u blokovima kodera i dekodera	512
N_{heads}	Broj glava u slojevima pažnje	4, 8 i 16
N_{blocks}	Broj blokova u koderu i dekoderu	1, 2, 3, 4, 5 i 6

6.2. Postupak treniranja modela

Postupak treniranja modela odvija se u epohama. U jednoj epohi model je vidio sve primjere koji se nalaze u skupu podataka. Epoha se sastoji od više koraka, a nakon svakog koraka provodi se ažuriranje parametara modela pomoću gradijentnog spusta. U jednom koraku model je vidio sve primjere jedne grupe (engl. *batch*).

Stopa učenja (engl. *learning rate*) određuje u kojoj se mjeri težine pomiču niz gradijent prema minimumu funkcije gubitka. Premala stopa učenja uzrokuje sporu konvergenciju, dok prevelika stopa učenja može uzrokovati divergenciju postupka treniranja. Stopa učenja može biti konstantna tijekom cijelog postupka treniranja modela, međutim pokazalo se da se optimizacija može ubrzati primjenom prilagodljive stope učenja čija se vrijednost mijenja ovisno o fazi treniranja [36]. Vrijednost stope učenja postupno se smanjuje kako treniranje modela napreduje, a time se omogućuje finije pretraživanje prostora težina. Kako bi se spriječilo da model zaglavi u suboptimalnom području, odnosno u lokalnom optimumu, na početku treniranja provodi se faza zagrijavanja stope učenja. Ideja je da početne vrijednosti stope učenja budu manje i postupno se povećavaju dok se ne dosegne maksimalna stopa učenja nakon čega se vrijednosti ponovno smanjuju. Izraz (9) opisuje prilagodbu stope učenja kakva se koristila za treniranje originalnog *transformer* modela. Takva je prilagodba implementirana i u ovom radu.

$$lrate = d_{model}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5}) \qquad (9)$$



Graf 2 Prilagodba stope učenja. U ovom primjeru dimenzija modela je 128, a broj koraka zagrijavanja 4000. Početni rast vrijednosti stope učenja predstavlja fazu zagrijavanja.

Optimizator je algoritam koji ažurira parametre modela kako bi se smanjio gubitak. U ovom je radu korišten optimizator Adam (engl. *adaptive moment estimation*) [37]. Isti optimizator korišten je za treniranje originalnog *transformer* modela, kao i za treniranje modela BetaAlign [9], [11]. Za parametre optimizatora uzete su vrijednosti $\beta_1 = 0.9$, $\beta_2 = 0.98$ te $\varepsilon = 10^{-9}$, po uzoru na prethodno spomenute radove.

Za izračun pogreške koristila se funkcija rijetke kategoričke unakrsne entropije (engl. *sparse categorical cross entropy*). Pogreška se u implementaciji *transformer* modela računa tako da se slijed generiranih tokena na izlazu modela uspoređuje s očekivanim izlazom. U generiranom izlazu modela na svakoj se poziciji nalazi vektor čija je dimenzija jednaka veličini izlaznog rječnika. Svaka pozicija predstavlja jedan token, a vrijednost na toj poziciji vjerojatnost da je upravo to token koji je model generirao na izlazu. Kako očekivani izlaz sadrži tokene koji su kodirani cjelobrojnom vrijednošću, za izračun pogreške koristi se rijetka kategorička unakrsna entropija. Kada bi očekivani izlaz sadržavao vektore kodirane *one-hot encoding* postupkom, koristila bi se obična pogreška kategoričke unakrsne entropije.

Za sprječavanje prenaučenosti modela, pojave u kojoj model postiže odlične rezultate na skupu za treniranje, ali vrlo loše rezultate na skupu za testiranje, korištena je tehnika ispadanja (engl. *dropout*) [38]. Ispadanje se primjenjuje na izlaz svakog sloja u *transformeru*, neposredno prije primjene rezidualne veze i normalizacije. Primjenjuje se i nakon postupka ugrađivanja ulaza. Ideja je da se slučajno odaberu pozicije unaprijed zadanom frekvencijom koje će se ugasiti, odnosno njihov će se izlaz postaviti na vrijednost 0. Ispadanje se primjenjuje tijekom treniranja modela, ali ne i tijekom testiranja i korištenja modela.

Tablica 11 Hiperparametri koji definiraju sam proces treniranja *transformer* modela. Svi modeli trenirani su korištenjem istih parametara navedenih u tablici.

Broj epoha	100
Veličina grupe (engl. *batch size*)	32
Frekvencija ispadanja (engl. *dropout rate*)	0.3

6.3. Računalni resursi korišteni za treniranje modela

Za treniranje modela korišten je računalni klaster Isabella koji održava Sveučilišni računski centar. Isabella se sastoji od 135 računalnih čvorova koji ukupno sadrže 270 CPU procesora, 3100 CPU procesorskih jezgri, 12 grafičkih procesora te 16TB radne memorije [39].

Platforma TensorFlow podržava izvođenje operacija na različitim uređajima, uključujući CPU (engl. *central processing unit*) i GPU (engl. *graphical processing unit*). Za treniranje dubokih modela preferira se korištenje grafičkih procesora, s obzirom na to da su takvi procesori specifično dizajnirani za brzo izvođenje složenih računskih operacija. Treniranje modela na grafičkim procesorima općenito traje višestruko manje vremena u odnosu na obične procesore.

Platforma TensorFlow može samostalno detektirati dostupne uređaje za izvođenje operacija. Izvođenje se automatski pokreće na grafičkom procesoru ako je dostupan, bez potrebe za prilagodbom koda.

7. Rezultati

Rezultati će biti predstavljeni korištenjem osnovnih modela. Pojam osnovnih modela u ovom radu označuje 6 modela koji se razlikuju isključivo po broju blokova u koderu i dekoderu (sadrže od 1 do 6 blokova). Svi ostali hiperparametri su im identični: broj glava u slojevima pažnje iznosi 8, dimenzija modela je 128, a latentna dimenzija 512. Trenirani su korištenjem reprezentacije parova na izlazu, dakle veličina ulaza im je 403 tokena, a izlaza 402 tokena. S obzirom na to da postoji velik broj kombinacija hiperparametara, ovi modeli će se koristiti kao referenca za uspoređivanje rezultata. U nastavku će biti prikazano kako promjena pojedinih hiperparametara utječe na točnost modela, kako odabir različite reprezentacije izlaza utječe na točnost modela te kakvi su rezultati ako se u obzir uzmu i ostale metrike, ispravnost te ocjena poravnanja korištenjem supstitucijske matrice.

7.1. Korištene metrike

Točnost (engl. *accuracy*) označava udio tokena u izlazu *transformera* koji se podudaraju sa tokenima u stvarnom izlazu. Na primjer, ako je izlaz *transformera* slijed parova [MM SA LL FF], a stvarni izlaz je [MM SL LF FF], točnost iznosi 50%. Točnost se u kontekstu poravnanja može shvatiti kao ocjena podudaranja stupaca (engl. *column score*). Poravnanje dva slijeda sastoji se od stupaca, a stupci sadrže par znakova iz oba slijeda. Dva različita poravnanja mogu se usporediti korištenjem ove metrike, a to je prikazano na slici ispod.

Slika 12 Ocjena podudaranja stupaca. Zelenom strelicom označeni su stupci koji se podudaraju u dva poravnanja, a crvenom strelicom stupci koji se razlikuju. Ocjena podudaranja stupaca iznosi 2/5 = 0.2.

Pogreška rijetke kategoričke unakrsne entropije (engl. *sparse categorical cross entropy*) računa se prilikom treniranja i validacije modela kao prosjek gubitka rijetke kategoričke unakrsne entropije na svim primjerima.

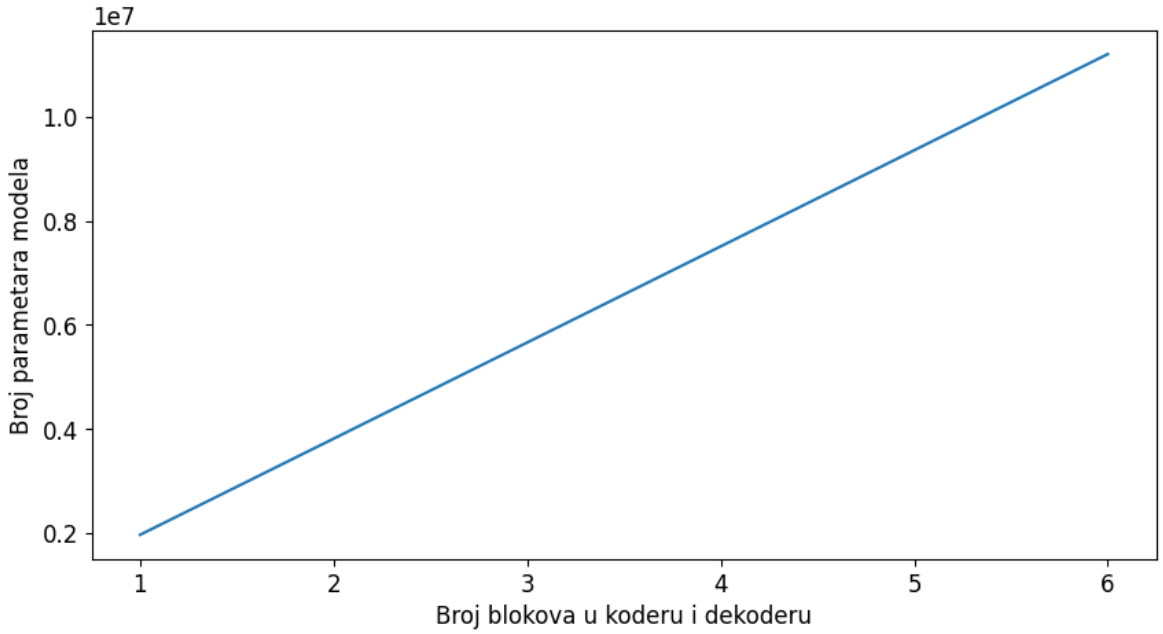
Pojam ispravnosti govori odgovaraju li poravnanja dobivena na izlazu *transformera* sljedovima koji su se pojavili na ulazu modela, nakon što se iz poravnanja maknu oznake procjepa. Nema garancije da će *transformer* na izlazu generirati točno one sljedove koje je dobio na ulazu, stoga je korisno pogledati u koliko slučajeva se generiraju ispravni sljedovi.

Poravnanje dobiveno *transformerom* može se također usporediti s referentnim poravnanjem pomoću supstitucijske matrice BLOSUM62 i parametara opisanih u potpoglavlju 5.1. Pomoću tih parametara može se izračunati ocjena poravnanja dobivenog *transformerom* koja se zatim uspoređuje s ocjenom referentnog poravnanja, a koja je dobivena korištenjem iste matrice i istih parametara.

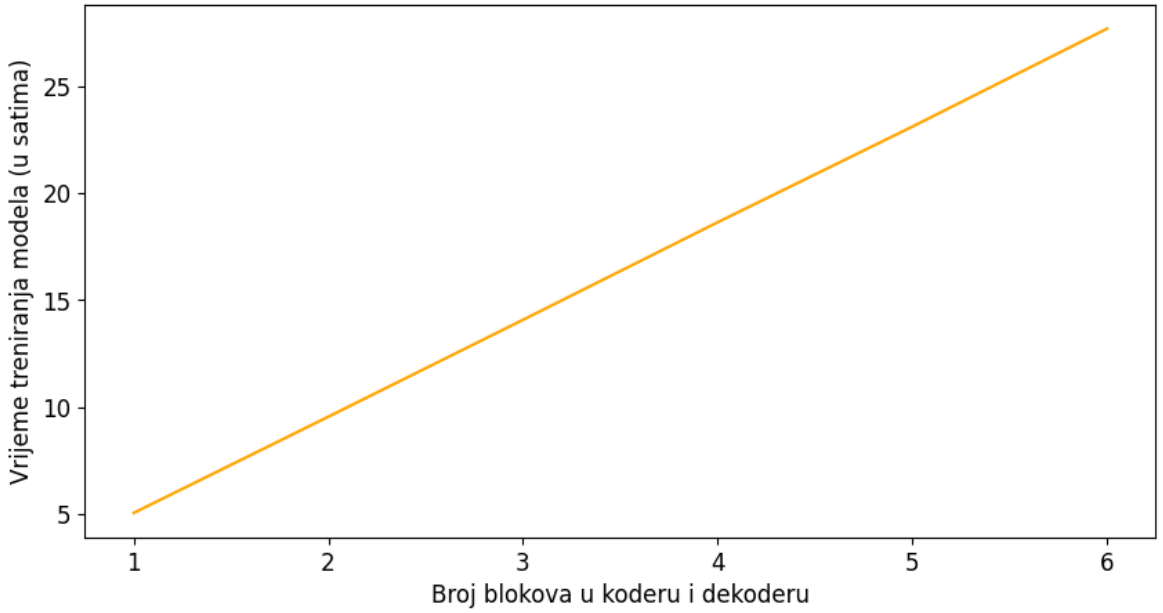
7.2. Utjecaj hiperparametara na broj parametara modela

Broj parametara modela (broj težina koje se ugađaju tijekom treniranja) utječe na vrijeme treniranja modela, memorijsko zauzeće, ali i na efikasnost samog postupka korištenja modela i računanja izlaza. Korisno je pogledati kako pojedini hiperparametri *transformer* arhitekture utječu na ukupan broj parametara modela i vrijeme potrebno za njegovo treniranje.

Dva grafa u nastavku opisuju ukupan broj parametara i vrijeme potrebno za treniranje izraženo u satima šest osnovnih modela koji su opisani na početku ovog poglavlja. Očekivano, broj parametara kao i vrijeme treniranja rastu linearno s porastom broja blokova, s obzirom na to da svaki blok sadrži jednak broj parametara. Postoje implementacije *transformer* modela s blokovima koji sadrže zajedničke dijeljene parametre, ali one nisu razmatrane u ovom radu. Najjednostavniji model s jednim blokom u koderu i dekoderu sadrži 1 964 861 parametara, a njegovo treniranje trajalo je 5.05 sati. Najsloženiji model sa šest blokova sadrži 11 200 701 parametar, a njegovo treniranje trajalo je 27.69 sati. Svi su modeli trenirani korištenjem grafičkih procesora.

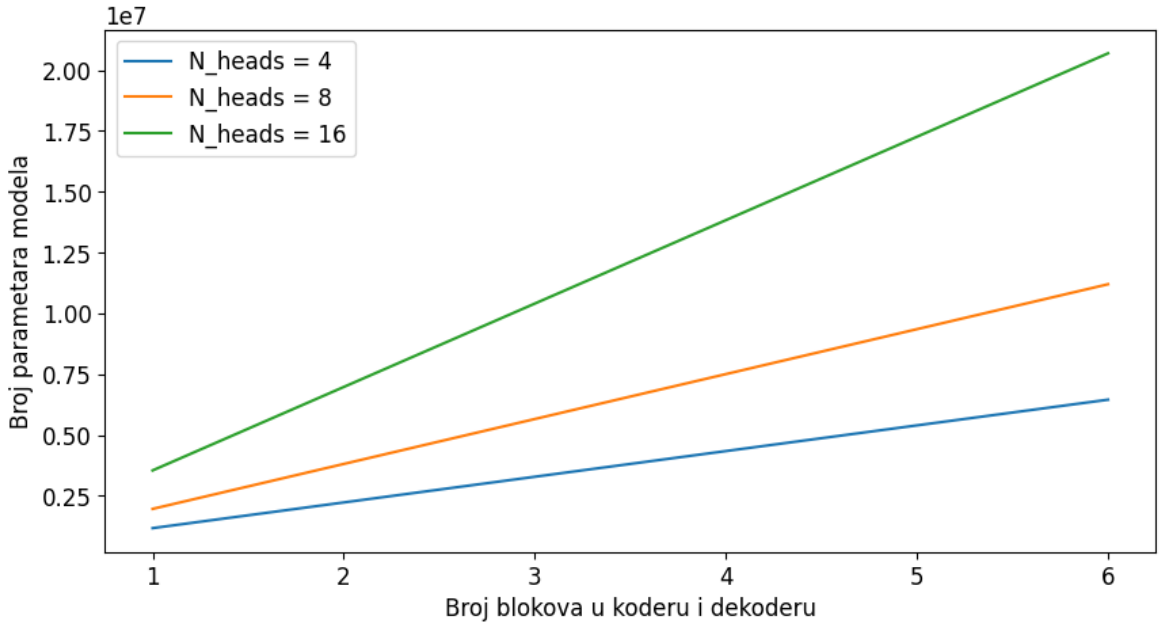


Graf 3 Broj parametara modela ovisno o broju blokova unutar kodera i dekodera



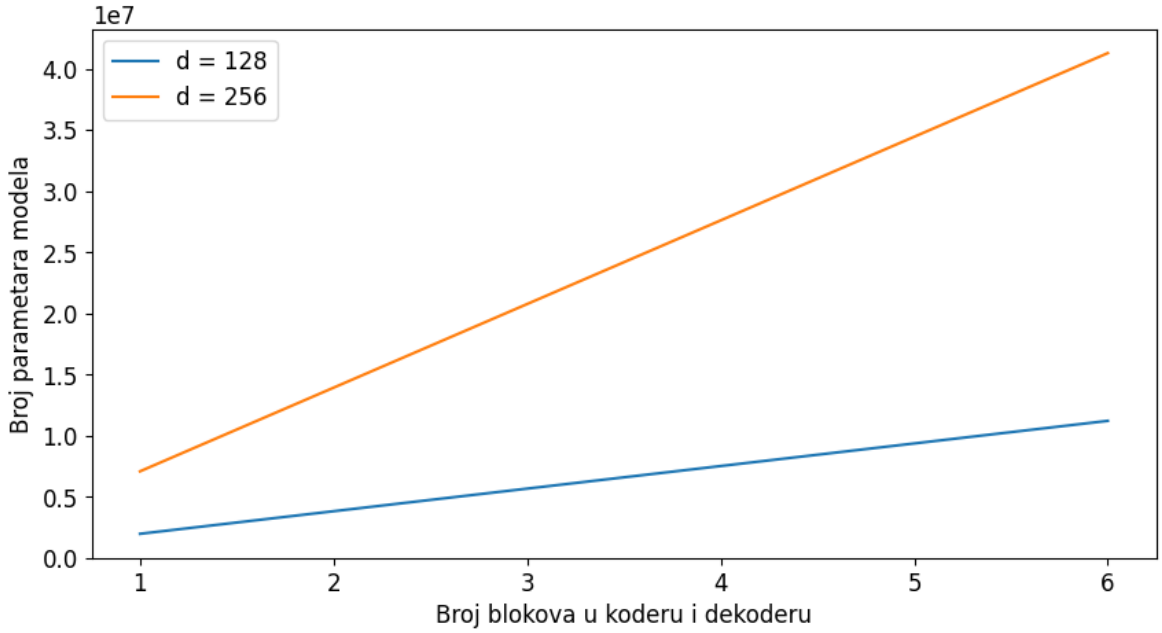
Graf 4 Vrijeme treniranja modela ovisno o broju blokova unutar kodera i dekodera

Graf 5 pokazuje kako se mijenja broj parametara modela ovisno o broju glava u slojevima pažnje i broju blokova u koderu i dekoderu. Svi modeli imaju dimenziju 128.



Graf 5 Broj parametara modela ovisno o broju glava u slojevima pažnje. Dimenzija modela je 128.

Graf 6 pokazuje kako se mijenja broj parametara modela ovisno o njegovoj dimenziji, d_{model}. Svi modeli na ovom grafu imaju isti broj glava pažnje, 8.

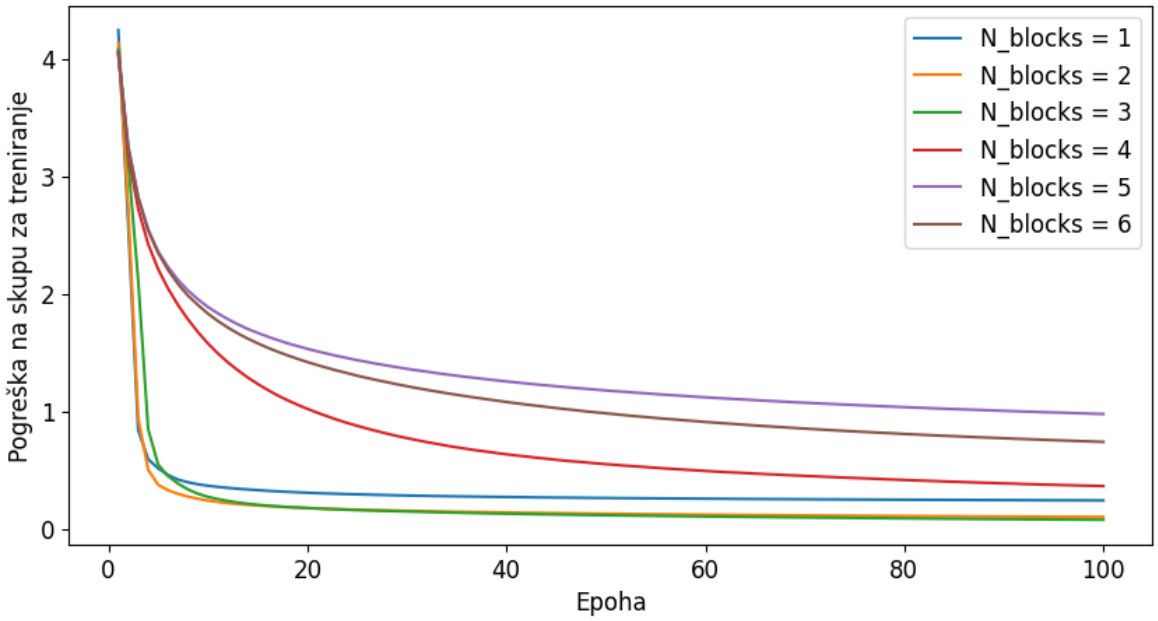


Graf 6 Broj parametara modela ovisno o dimenziji modela. Broj glava pažnje je 8.

Iz prethodnih je grafova vidljivo kako najveći utjecaj na broj parametara modela ima dimenzija modela, d_{model}. Pravac na grafu 6 koji predstavlja tu dimenziju ima najbrži rast. Dvostruko povećanje te dimenzije, s vrijednosti 128 na vrijednost 256, uzrokuje povećanje ukupnog broja parametara 3-4 puta, ovisno o tome kakvi su ostali hiperparametri.

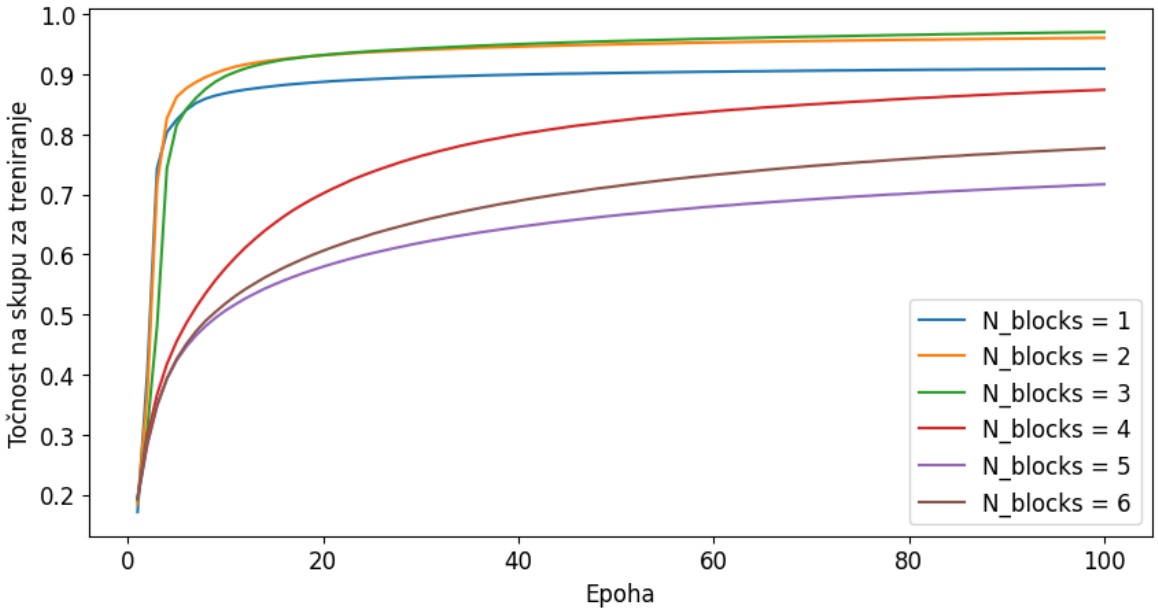
7.3. Rezultati osnovnih modela

U ovom će potpoglavlju biti prikazani rezultati koje 6 osnovnih modela ostvaruju na skupu za treniranje i skupu za validaciju. Prosječni gubitak rijetke kategoričke unakrsne entropije i prosječna točnost računaju se nakon svake epohe, na skupu za treniranje te na skupu za validaciju. Sljedeći graf pokazuje kako se mijenja pogreška na skupu za treniranje tijekom 100 epoha treniranja modela.



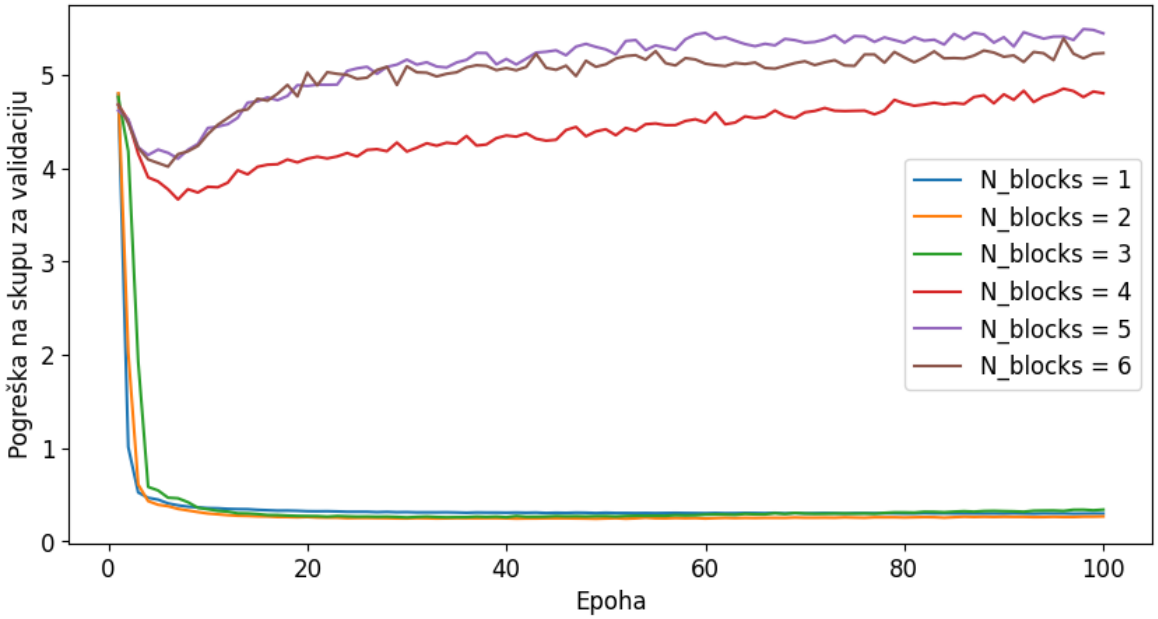
Graf 7 Pogreška osnovnih modela na skupu za treniranje.

Pogreška očekivano opada kako treniranje napreduje. Sljedeći graf pokazuje kako se mijenja točnost modela tijekom 100 epoha treniranja. Točnost očekivano raste za sve modele, kako treniranje napreduje. Može se primijetiti kako jednostavniji modeli (s 1, 2 ili 3 bloka unutar kodera i dekodera) brže konvergiraju.

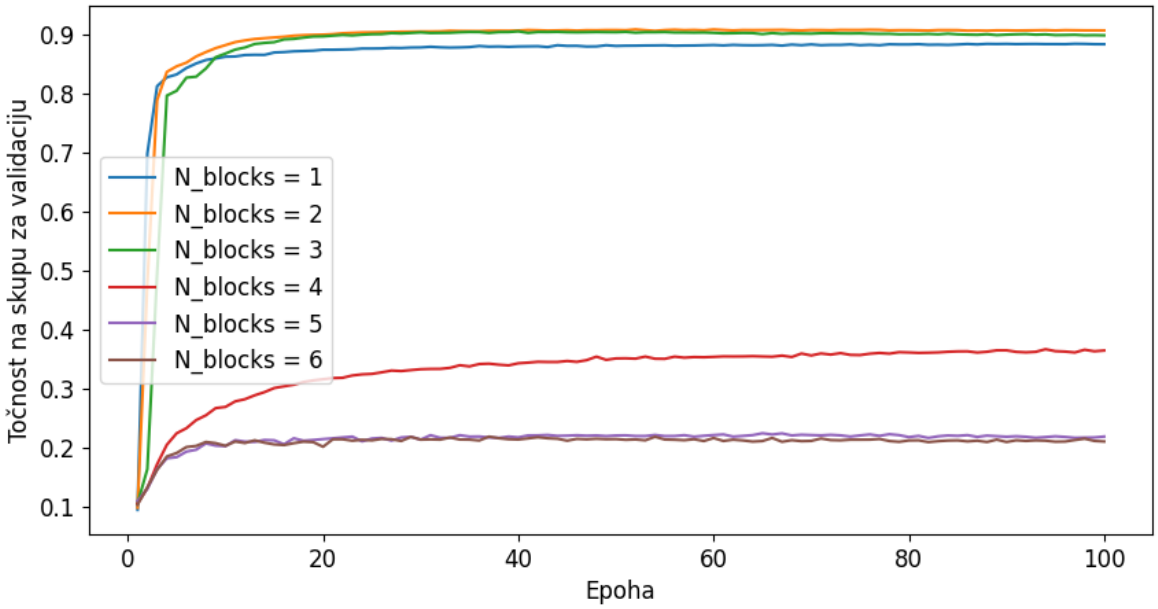


Graf 8 Točnost osnovnih modela na skupu za treniranje

Kako bi se provjerila sposobnost generalizacije modela i kako bi se utvrdilo da model nije prenaučen, koristi se i skup podataka za validaciju modela. Taj skup sadrži primjere na kojima model nije treniran. Sljedeća dva grafa prikazuju pogrešku i točnost na skupu za validaciju.



Graf 9 Pogreška osnovnih modela na skupu za validaciju



Graf 10 Točnost osnovnih modela na skupu za validaciju

Na grafovima se može primijetiti kako su modeli s 5 i 6 blokova presloženi za ovaj zadatak i previše su se prilagodili skupu za treniranje, zbog čega postižu loše rezultate na skupu za validaciju. Model s 4 bloka postiže nešto bolje rezultate, ali i dalje puno lošije u odnosu na tri jednostavnija modela koja postižu vrlo dobre rezultate i ostvaruju točnost od približno 90% na skupu za validaciju. Najveće točnosti koje modeli postižu na skupu za validaciju dodatno su prikazane u tablici ispod.

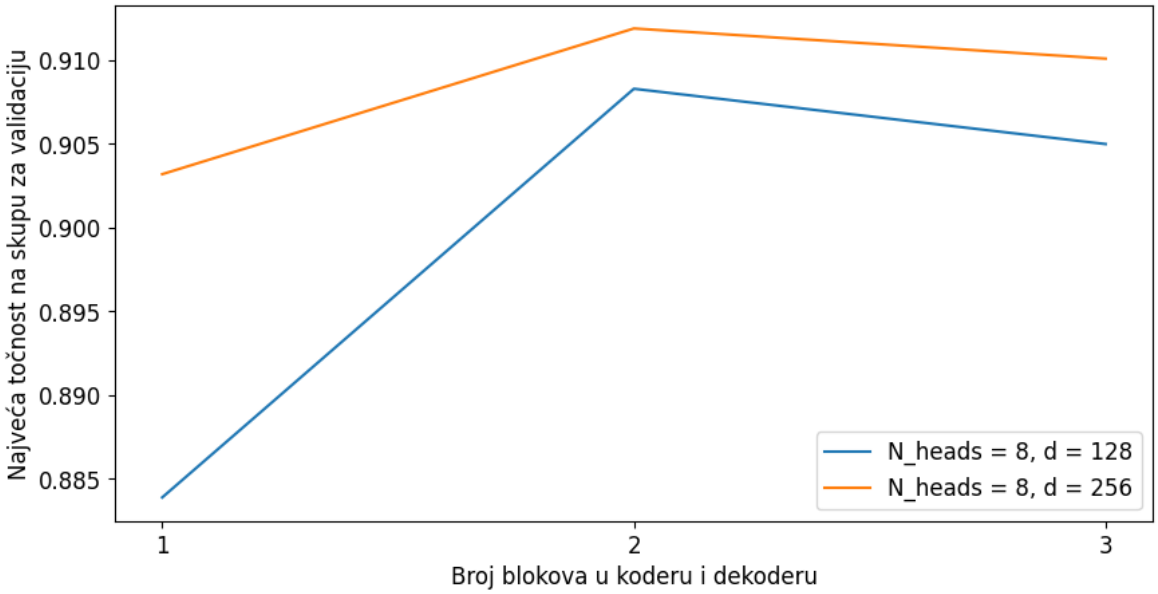
Tablica 12 Najveće točnosti koje osnovni modeli ostvaruju na skupu za validaciju

N_blocks = 1	best validation accuracy = 0.8839
N_blocks = 2	best validation accuracy = 0.9083
N_blocks = 3	best validation accuracy = 0.905
N_blocks = 4	best validation accuracy = 0.3663
N_blocks = 5	best validation accuracy = 0.2239
N_blocks = 6	best validation accuracy = 0.2182

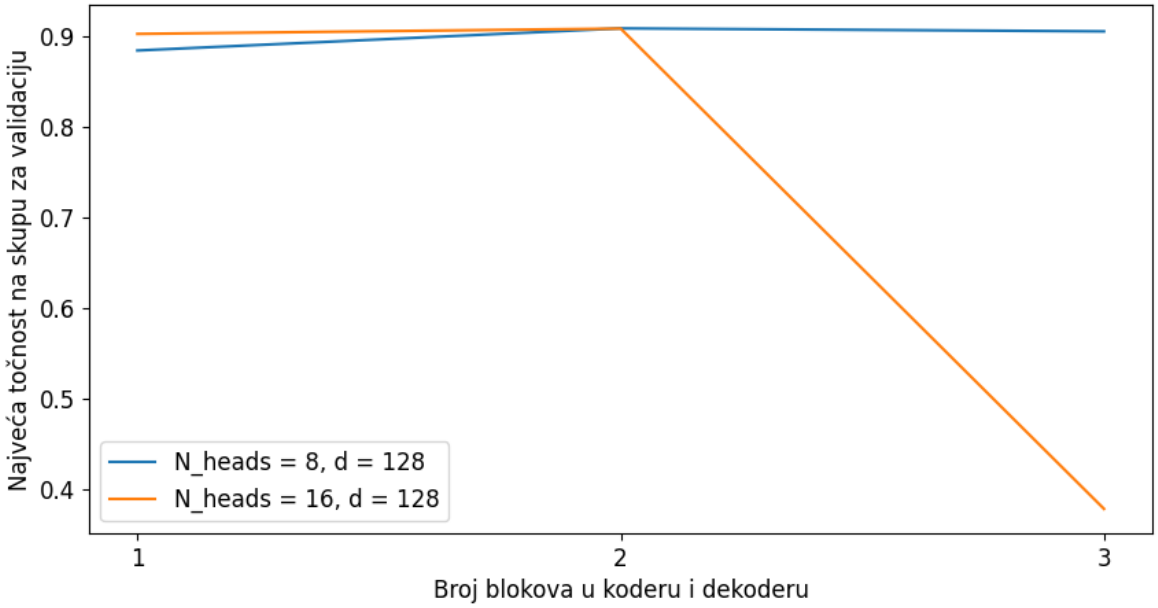
Najbolju točnost ostvaruje model s 2 bloka.

7.4. Utjecaj hiperparametara na rezultate

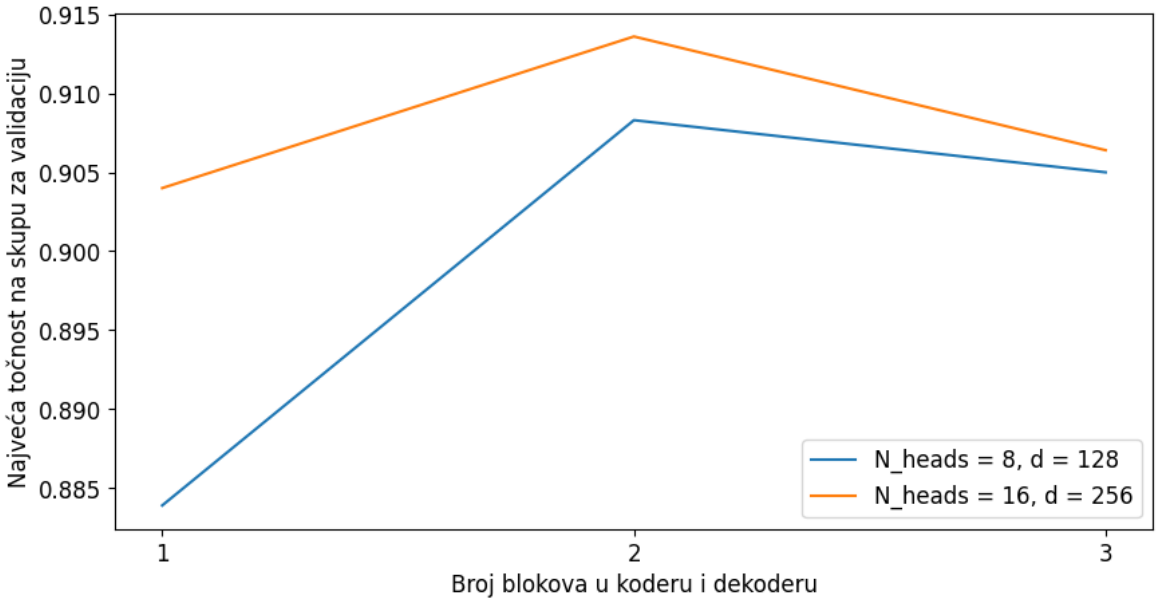
U nastavku rada će fokus biti na 3 modela koja su ostvarila najbolje rezultate na skupu za validaciju. Sljedeći grafovi prikazuju kako povećanje pojedinih hiperparametara utječe na najbolju točnost koju model ostvaruje na skupu za validaciju.



Graf 11 Utjecaj povećanja dimenzije na točnost modela

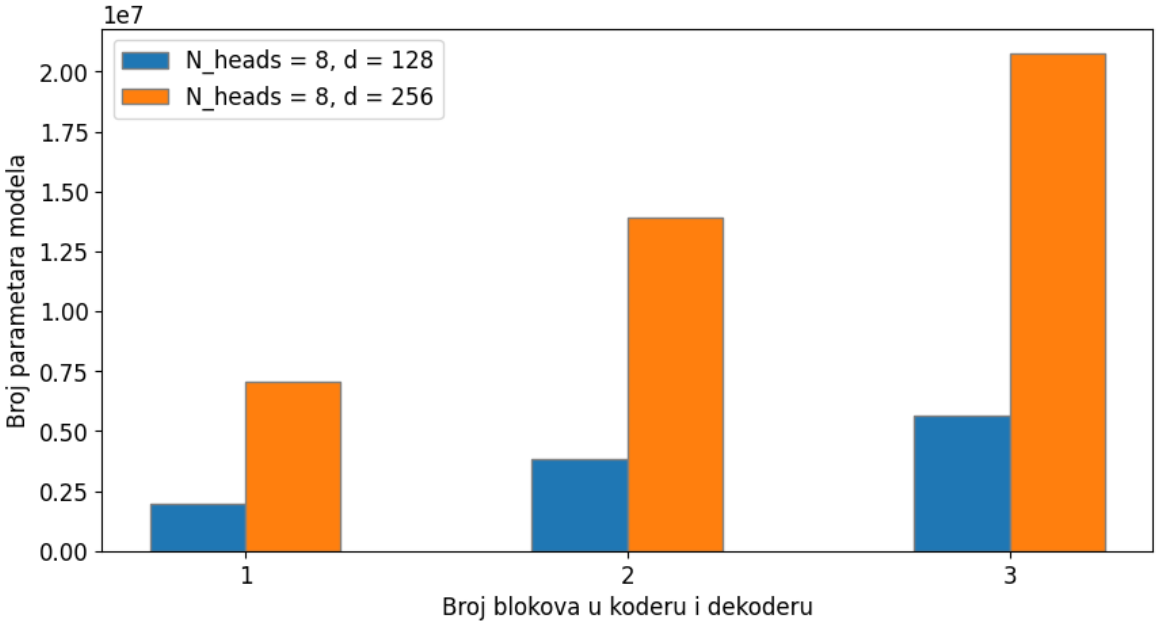


Graf 12 Utjecaj povećanja broja glava na točnost modela



Graf 13 Utjecaj povećanja dimenzije i broja glava na točnost modela

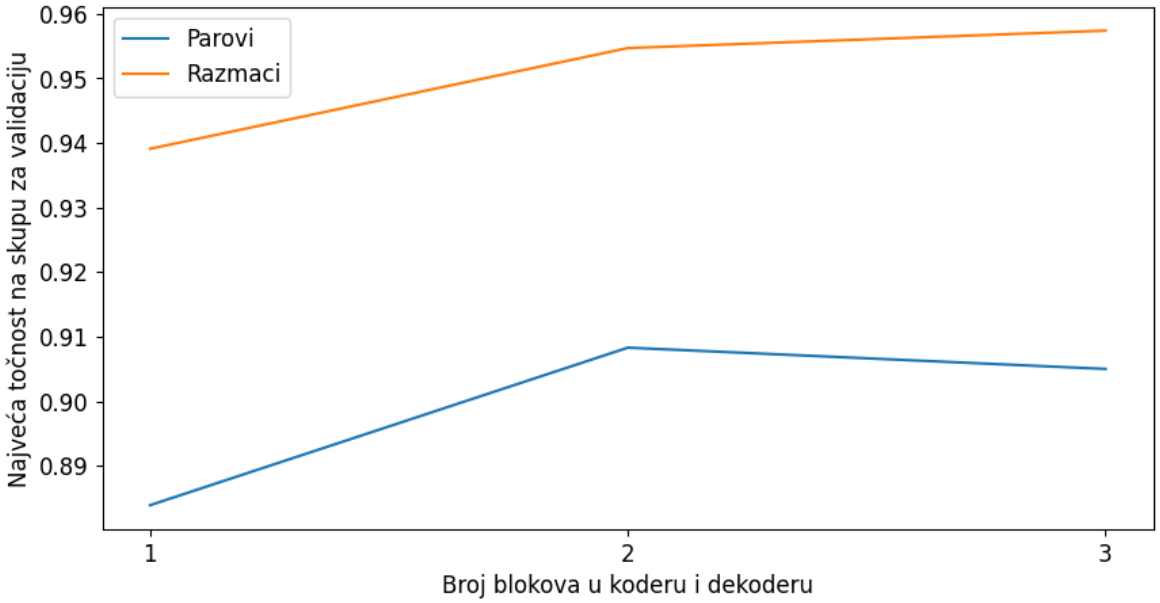
Povećanje dimenzije modela uvijek je rezultiralo nešto boljom točnošću, dok povećanje broja glava u sloju pažnje nije imalo nekog pravilnog utjecaja na rezultate. Razlike u točnosti uzrokovane različitom dimenzijom modela najviše dolaze do izražaja kod najjednostavnijeg modela, dok kod modela s 2 i 3 bloka razlike u točnosti nisu toliko izražene, ali je izražena razlika u broju parametara koje prikazuje graf ispod.



Graf 14 Razlika u broju parametara modela s dimenzijama 128 i 256

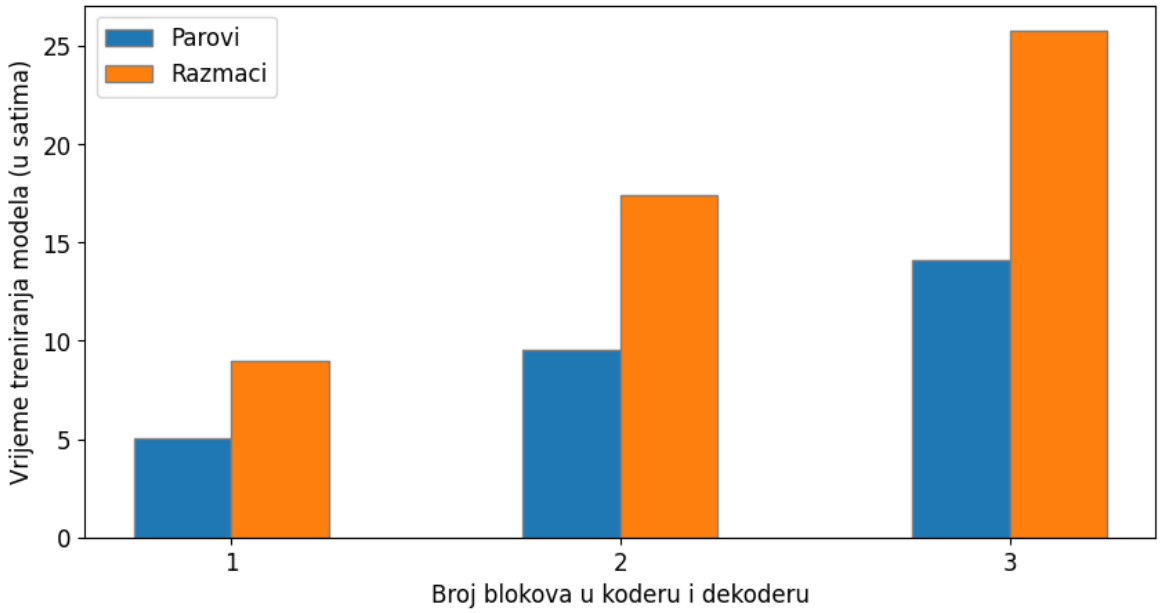
7.5. Utjecaj reprezentacije izlaza na rezultate

Tri osnovna modela trenirana su korištenjem oba načina reprezentacije izlaza koja su objašnjena u potpoglavlju 5.2.



Graf 15 Usporedba točnosti modela koji koriste različite reprezentacije izlaza. Ostali hiperparametri su N_heads = 8 i d = 128.

Gornji graf pokazuje kako se korištenjem reprezentacije izlaza pomoću razmaka postižu bolji rezultati u odnosu na reprezentaciju pomoću parova. To bi se moglo objasniti činjenicom da je korištenjem razmaka izlaz modela puno sličniji ulazu, s obzirom na to da je i u reprezentaciji ulaza svaka aminokiselina zaseban token, pa je lakše detektirati ovisnosti između pojedinih aminokiselina. Modeli koji koriste reprezentaciju pomoću razmaka imaju nešto manji broj parametara jer je izlaz završnog linearnog sloja modela manjih dimenzija, zbog manje veličine izlaznog rječnika. Međutim, treniranje modela koji koriste reprezentaciju pomoću razmaka traje nešto duže u odnosu na treniranje modela koji koriste parove, zato što je ulaz u dekoder modela u slučaju korištenja reprezentacije razmaka dvostruko dulji i potrebno je više vremena da se napravi obrada cijelog ulaza, odnosno obavi se dvostruko više prolazaka kroz dekoder dio. Graf u nastavku prikazuje usporedbu vremena potrebnog za treniranje modela koji koriste različite reprezentacije.



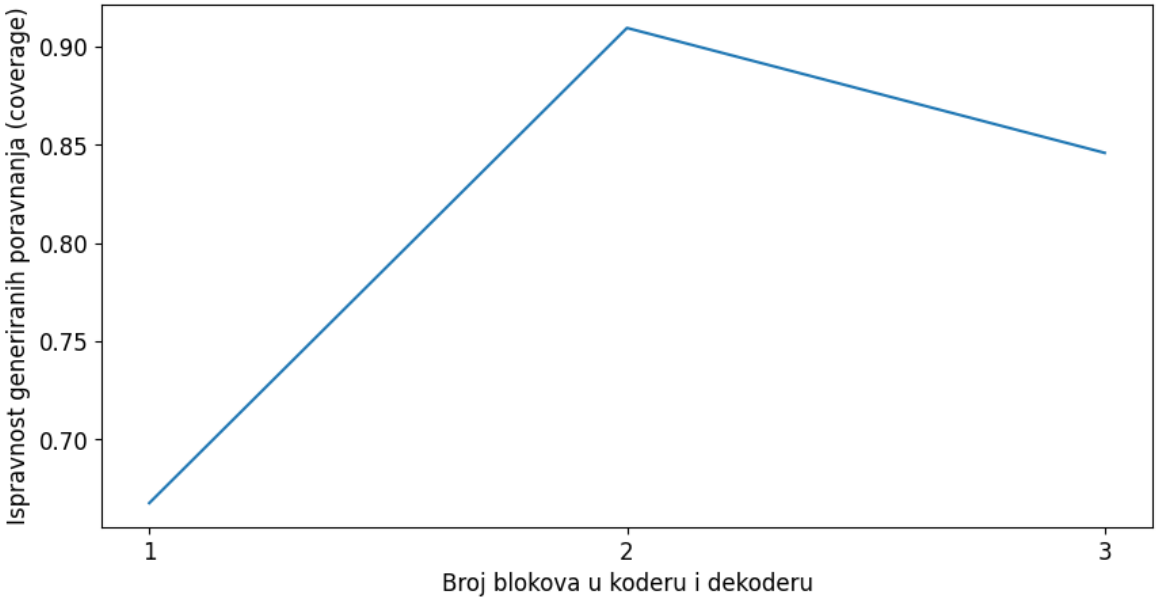
Graf 16 Vrijeme potrebno za treniranje modela koji koriste različite reprezentacije izlaza

7.6. Rezultati na skupu za testiranje

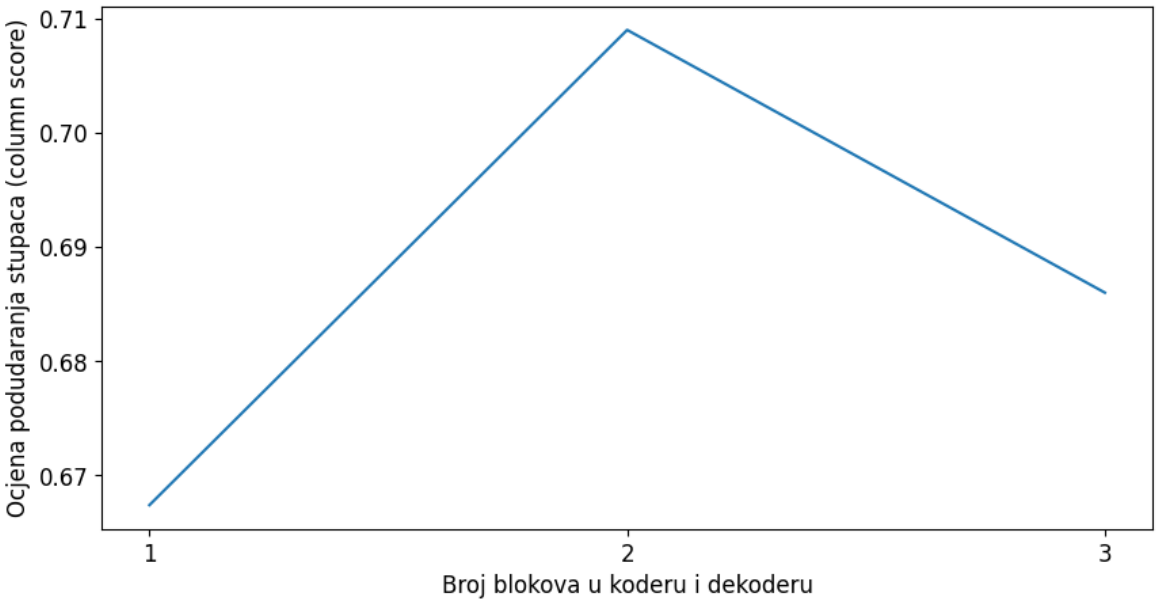
Kako bi se dodatno provjerila uspješnost modela, koristi se skup za testiranje. Na skupu za treniranje ne može se provjeriti sposobnost generalizacije modela, a skup za validaciju koristi se za odabir optimalnih hiperparametara modela (kao što je ovdje odabran optimalan broj blokova u koderu i dekoderu modela) pa je na neki način upleten u proces treniranja. Za testiranje modela potrebno je imati skup koji je ostao netaknut za vrijeme treniranja. Zato se koristi skup novih primjera, skup za testiranje.

Skup za testiranje u ovom radu sadržavao je 1000 slučajno odabranih parova sljedova, pri čemu je naravno vođeno računa o tome da se sljedovi koji se pojavljuju u skupu za testiranje nisu pojavili u skupovima za treniranje i validaciju. Poželjno je uvijek imati što veći skup podataka kako bi dobiveni rezultati što bolje odražavali stvarnu uspješnost modela, ali zbog vremenskih i resursnih ograničenja odabran je manji skup od 1000 sljedova. Na ovom je skupu provjerena uspješnost najbolja tri osnovna modela u ostalim metrikama osim točnosti, a to su ispravnost generiranih sljedova i usporedba ocjene poravnanja s ocjenom referentnog poravnanja.

Sljedeća dva grafa prikazuju kakva je ispravnost generiranih poravnanja i kakva je točnost osnovnih modela na skupu za testiranje.



Graf 17 Ispravnost izlaza osnovnih modela na skupu za testiranje



Graf 18 Točnost (ocjena podudaranja stupaca) osnovnih modela na skupu za testiranje

Svi rezultati ostvareni na skupu za testiranje prikazani su u tablici ispod.

Tablica 13 Prikaz rezultata ostvarenih na skupu za testiranje

	Model s 1 blokom	Model s 2 bloka	Model s 3 bloka
Točnost	0.66741	0.70899	0.68600
Ispravnost	0.66767	0.90941	0.84585
Srednja apsolutna razlika ocjene poravnanja	34.00000	21.03097	24.17
Prosječno odstupanje od srednje apsolutne razlike ocjene poravnanja	35.55453	23.72751	24.62778
Srednja relativna razlika ocjene poravnanja	0.30057	0.20142	0.23866
Prosječno odstupanje od srednje relativne razlike ocjene poravnanja	0.43101	0.25867	0.28854

Apsolutna razlika ocjene poravnanja računa se tako da se ocjena referentnog poravnanja i ocjena poravnanja generiranog *transformerom* jednostavno oduzmu. Relativna razlika ocjene poravnanja računa se tako da se vrijednost apsolutne razlike dodatno podijeli s većom vrijednošću ocjene poravnanja.

Najbolje rezultate pokazuje model koji sadrži 2 bloka u koderu i dekoderu. Lošiji rezultati svih modela na skupu za testiranje u odnosu na skup za validaciju mogli bi se objasniti manjim nereprezentativnim skupom podataka, iako je ocjena podudaranja stupaca od 0.7 vrlo dobar rezultat, a najbolji model ostvaruje i vrlo visoku ispravnost što govori kako su generirana poravnanja smislena, ali nisu optimalna na što bi se trebalo fokusirati u budućem radu.

Zaključak

U ovom je radu pokazano kako se *transformer* arhitektura može primijeniti na zadatak poravnanja parova proteinskih sljedova. Dotan et al. predstavili su ovu ideju u svom radu i trenirali su *transformer* model na simuliranim poravnanjima, a u ovom su radu za treniranje modela korišteni stvarni sljedovi hemoglobina. Rezultati pokazuju da je moguće izgraditi model dubokog učenja koji može generirati smislena i ispravna poravnanja, ali ima prostora za napredak u pogledu kvalitete dobivenih poravnanja te vremena potrebnog za određivanje poravnanja. Treba uzeti u obzir činjenicu da je za treniranje modela u ovom radu korišten ograničen skup srodnih sljedova hemoglobina koji su međusobno vrlo slični, pa je upitno kakve bi rezultate modeli postigli u slučaju korištenja novih sljedova različitih od hemoglobina.

Isprobane su različite arhitekture *transformer* modela i pokazalo se da i arhitekture puno jednostavnije od one originalnog *transformer* modela mogu ostvariti dobre rezultate. Jednostavnije arhitekture imaju manji broj parametara što olakšava proces treniranja, smanjuje memorijsko zauzeće, olakšava korištenje modela u produkciji, ali omogućuje i bolju interpretabilnost. Najbolje rezultate postigla je arhitektura koja se sastoji od 2 bloka unutar kodera i dekodera. Pokazalo se da korištenje različitih načina reprezentacije parova sljedova na izlazu modela može poboljšati točnost. Reprezentacija izlaza koja je za potrebe ovog rada nazvana razmaci ostvaruje nešto bolje rezultate u pogledu točnosti i ispravnosti poravnanja, ali treba uzeti u obzir povećano vrijeme potrebno za obradu sljedova na izlazu modela koji su u slučaju korištenja te reprezentacije dvostruko dulji.

Transformer modeli koji se koriste za poravnanje sljedova trenutno nisu usporedivi s najčešće korištenim algoritmima za poravnanje, najviše zbog vremena potrebnog za generiranje poravnanja, ali konceptualno su vrlo zanimljivi i otvaraju brojne nove mogućnosti u bioinformatici.

Literatura

[1] J. Pevsner, *Bioinformatics and Functional Genomics*, Third Edition. Chichester: John Wiley & Sons Inc, 2015.

[2] A. Lesk, *Introduction to Bioinformatics*. New York: Oxford University Press, 2002.

[3] W. R. Pearson, “An introduction to sequence similarity (‘homology’) searching,” *Curr Protoc Bioinformatics*, no. SUPPL.42, 2013, doi: 10.1002/0471250953.bi0301s42.

[4] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, “Basic local alignment search tool,” *J Mol Biol*, vol. 215, no. 3, pp. 403–410, 1990, doi: 10.1016/S0022-2836(05)80360-2.

[5] W. R. Pearson and D. J. Lipman, “Improved tools for biological sequence comparison.,” *Proc Natl Acad Sci U S A*, vol. 85, no. 8, pp. 2444–2448, 1988, doi: 10.1073/pnas.85.8.2444.

[6] A. L. Delcher, S. Kasif, R. D. Fleischmann, J. Peterson, O. White, and S. L. Salzberg, “Alignment of whole genomes,” 1999.

[7] K. Katoh, K. Misawa, K.-I. Kuma, and T. Miyata, “MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform.”

[8] J. Chao, F. Tang, and L. Xu, “Developments in Algorithms for Sequence Alignment: A Review,” *Biomolecules*, vol. 12, no. 4. MDPI, Apr. 01, 2022. doi: 10.3390/biom12040546.

[9] A. Vaswani *et al.*, “Attention Is All You Need,” Jun. 2017, [Online]. Available: http://arxiv.org/abs/1706.03762

[10] S. Zhang, R. Fan, Y. Liu, S. Chen, Q. Liu, and W. Zeng, “Applications of transformer-based language models in bioinformatics: a survey,” *Bioinformatics Advances*, vol. 3, no. 1, Jan. 2023, doi: 10.1093/bioadv/vbad001.

[11] E. Dotan *et al.*, “MULTIPLE SEQUENCE ALIGNMENT AS A SEQUENCE-TO-SEQUENCE LEARNING PROBLEM.”

[12] S. Min, B. Lee, and S. Yoon, “Deep learning in bioinformatics,” *Briefings in bioinformatics*, vol. 18, no. 5. pp. 851–869, Sep. 01, 2017. doi: 10.1093/bib/bbw068.

[13] I. H. Sarker, “Machine Learning: Algorithms, Real-World Applications and Research Directions,” *SN Computer Science*, vol. 2, no. 3. Springer, May 01, 2021. doi: 10.1007/s42979-021-00592-x.

[14] Y. Lecun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553. Nature Publishing Group, pp. 436–444, May 27, 2015. doi: 10.1038/nature14539.

[15] J. Patterson and A. Gibson, “Deep Learning A Practitioner’s Approach,” 2017. [Online]. Available: http://oreilly.com/safari

[16] M. P. Hosseini, S. Lu, K. Kamaraj, A. Slowikowski, and H. C. Venkatesh, “Deep Learning Architectures,” in *Studies in Computational Intelligence*, Springer Verlag, 2020, pp. 1–24. doi: 10.1007/978-3-030-31756-0_1.

[17] J. Devlin, M.-W. Chang, K. Lee, K. T. Google, and A. I. Language, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” [Online]. Available: https://github.com/tensorflow/tensor2tensor

[18] A. R. Openai, K. N. Openai, T. S. Openai, and I. S. Openai, “Improving Language Understanding by Generative Pre-Training.” [Online]. Available: https://gluebenchmark.com/leaderboard

[19] C. Raffel *et al.*, “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer,” Oct. 2019, [Online]. Available: http://arxiv.org/abs/1910.10683

[20] A. Dosovitskiy *et al.*, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale,” Oct. 2020, [Online]. Available: http://arxiv.org/abs/2010.11929

[21] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, “Training data-efficient image transformers & distillation through attention,” Dec. 2020, [Online]. Available: http://arxiv.org/abs/2012.12877

[22] Y. Gong, Y.-A. Chung, and J. Glass, “AST: Audio Spectrogram Transformer,” Apr. 2021, [Online]. Available: http://arxiv.org/abs/2104.01778

[23] N. Brandes, D. Ofer, Y. Peleg, N. Rappoport, and M. Linial, “ProteinBERT: a universal deep-learning model of protein sequence and function,” *Bioinformatics*, vol. 38, no. 8, pp. 2102–2110, Apr. 2022, doi: 10.1093/bioinformatics/btac020.

[24] Ş. B. Marcu, S. Tăbîrcă, and M. Tangney, “An Overview of Alphafold’s Breakthrough,” *Frontiers in Artificial Intelligence*, vol. 5. Frontiers Media S.A., Jun. 09, 2022. doi: 10.3389/frai.2022.875587.

[25] T. Lin, Y. Wang, X. Liu, and X. Qiu, “A Survey of Transformers,” Jun. 2021, [Online]. Available: http://arxiv.org/abs/2106.04554

[26] A. Huang, S. Subramanian, J. Sum, K. Almubarak, S. Biderman, and S. Rush, “The Annotated Transformer,” 2022. https://nlp.seas.harvard.edu/annotated-transformer/ (accessed Apr. 24, 2023).

[27] J. Alammar, “The Illustrated Transformer,” 2018. https://jalammar.github.io/illustrated-transformer/ (accessed Apr. 24, 2023).

[28] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer Normalization,” Jul. 2016, [Online]. Available: http://arxiv.org/abs/1607.06450

[29] NCBI, “National Library of Medicine.” https://www.ncbi.nlm.nih.gov/ (accessed May 16, 2023).

[30] “Biopython Documentation.” https://biopython.org/ (accessed May 22, 2023).

[31] S. B. Needleman and C. D. Wunsch, “A general method applicable to the search for similarities in the amino acid sequence of two proteins,” *J Mol Biol*, vol. 48, no. 3, pp. 443–453, Mar. 1970, doi: 10.1016/0022-2836(70)90057-4.

[32] S. R. Eddy, “Where did the BLOSUM62 alignment score matrix come from?,” *Nat Biotechnol*, vol. 22, no. 8, pp. 1035–1036, Aug. 2004, doi: 10.1038/nbt0804-1035.

[33] M. Abadi *et al.*, “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems.” [Online]. Available: www.tensorflow.org.

[34] F. Chollet and others, “Keras.” 2015. Accessed: Jun. 07, 2023. [Online]. Available: https://keras.io/

[35] “Neural machine translation with a Transformer and Keras.” https://www.tensorflow.org/text/tutorials/transformer (accessed Jun. 01, 2023).

[36] L. Liu *et al.*, “On the Variance of the Adaptive Learning Rate and Beyond,” Aug. 2019, [Online]. Available: http://arxiv.org/abs/1908.03265

[37] D. P. Kingma and J. Lei Ba, “ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION.”

[38] N. Srivastava, G. Hinton, A. Krizhevsky, and R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” 2014.

[39] “Tehničke specifikacije klastera Isabelle.” https://wiki.srce.hr/pages/viewpage.action?pageId=49284364 (accessed Jun. 28, 2023).

[40] LabXchange © The President and Fellows of Harvard College, “BLOSUM62 Substitution Matrix.” https://www.labxchange.org/library/items/lb:LabXchange:24d0ec21:lx_image:1 (accessed Jun. 21, 2023).

Popis tablica

Popis slika

Popis grafova

Sažetak

Naslov

Određivanje poravnanja parova proteinskih sljedova korištenjem modela dubokog učenja

Sažetak

Poravnanje sljedova važan je postupak u bioinformatici koji se može opisati kao određivanje korespondencija (podudaranje, nepodudaranje, umetanje ili brisanje znakova) između pojedinih elemenata dvaju ili više bioloških sljedova. Svrha poravnanja je procjena sličnosti sljedova na temelju čega se može odrediti jesu li sljedovi homologni, odnosno jesu li evolucijski srodni. Ovaj rad istražuje mogućnost poravnanja parova proteinskih sljedova korištenjem transformer modela. Transformer modeli originalno su korišteni za zadatke u području obrade prirodnog jezika, međutim pokazali su se vrlo uspješnima i u drugim područjima, poput računalnog vida, obrade zvuka i bioinformatike. U sklopu rada napravljena je vlastita implementacija transformer modela korištenjem okvira TensorFlow i Keras. Za potrebe treniranja i testiranja modela pripremljeni su skupovi neporavnatih i poravnatih proteinskih sljedova hemoglobina, a referentna poravnanja dobivena su algoritmom Needleman-Wunsch. Uspoređeni su različiti načini reprezentacije poravnatih i neporavnatih sljedova kao i različite arhitekture transformer modela. Ocjena podudaranja stupaca koristila se kao mjera za usporedbu s referentnim poravnanjem, a za najbolji model na skupu za testiranje iznosila je 71%.

Ključne riječi

bioinformatika; poravnanje sljedova; duboko učenje; transformer

Summary

Title

Deep learning-based pairwise alignment of protein sequences

Summary

Sequence alignment is an important process in bioinformatics that can be described as the assignment of correspondences (match, mismatch, insertion or deletion) between nucleotides or amino acids. The goal of sequence alignment is to assess the similarity between aligned sequences, which can later be used to determine whether the sequences are homologous. This thesis explores the possibility of using transformer models for pairwise protein sequence alignment. Transformer models were originally used for natural language processing tasks, but they proved to be successful in many different fields, such as computer vision, sound analysis and bioinformatics. In this thesis TensorFlow and Keras libraries were used for transformer model implementation. Datasets consisting of unaligned and aligned hemoglobin protein sequences were created for model training and testing. Reference alignments were obtained using the Needleman-Wunsch algorithm. Different ways of representing unaligned and aligned sequences were compared as well as different model architectures. Column score was used as a measure of similarity with reference alignments. Best model achieved a column score of 71% on test data.

Keywords

bioinformatics; sequence alignment; deep learning; transformer