Implementacija i analiza arhitekture Mamba za neuronske jezične modele

Krsnik, Lea

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 571

IMPLEMENTACIJA I ANALIZA ARHITEKTURE MAMBA ZA NEURONSKE JEZIČNE MODELE

Lea Krsnik

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 571

IMPLEMENTACIJA I ANALIZA ARHITEKTURE MAMBA ZA NEURONSKE JEZIČNE MODELE

Lea Krsnik

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 571

Pristupnica:	**Lea Krsnik (0036522009)**
Studij:	Računarstvo
Profil:	Znanost o podacima
Mentor:	prof. dr. sc. Jan Šnajder

Zadatak: **Implementacija i analiza arhitekture Mamba za neuronske jezične modele**

Opis zadatka:

Jezični modeli zasnovani na arhitekturi Transformatora (engl. Transformer) i mehanizmu pozornosti postižu izvanredne rezultate u obradi prirodnoga jezika. Međutim, unatoč učinkovitosti učenja, ovi modeli suočavaju se s visokom vremenskom i prostornom složenošću, što ograničava njihovu praktičnu primjenjivost. Premda je predloženo niz poboljšanja koja ciljaju na smanjenje složenosti, ona najčešće ne dostižu prediktivnu točnost izvorne arhitekture Transformatora. U nedavnome radu predložena je nova arhitektura Mamba, koja se ističe kao možebitna zamjena za transformator, nudeći usporedive rezultate s manjom vremenskom složenošću. Cilj rada jest dubinska analiza arhitekture Mamba, uključujući njezine temeljne komponente i usporedbu s prethodno predloženim poboljšanjima arhitekture Transformatora. Implementirati arhitekturu Mamba u programskome jeziku Python. Vrednovati učinkovitost arhitekture na zadatcima modeliranja sljedova. Radu priložiti izvorni i izvršni kod radnog okvira, skupove podataka i programsku dokumentaciju te citirati korištenu literaturu.

Rok za predaju rada: 28. lipnja 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 571

Implementacija i analiza arhitekture Mamba za neuronske jezične modele

Lea Krsnik

Zagreb, srpanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

DIPLOMSKI ZADATAK br. 571

Pristupnica:	**Lea Krsnik (0036522009)**
Studij:	Računarstvo
Profil:	Znanost o podacima
Mentor:	prof. dr. sc. Jan Šnajder

Zadatak: **Implementacija i analiza arhitekture Mamba za neuronske jezične modele**

Opis zadatka:

Jezični modeli zasnovani na arhitekturi Transformatora (engl. Transformer) i mehanizmu pozornosti postižu izvanredne rezultate u obradi prirodnoga jezika. Međutim, unatoč učinkovitosti učenja, ovi modeli suočavaju se s visokom vremenskom i prostornom složenošću, što ograničava njihovu praktičnu primjenjivost. Premda je predloženo niz poboljšanja koja ciljaju na smanjenje složenosti, ona najčešće ne dostižu prediktivnu točnost izvorne arhitekture Transformatora. U nedavnome radu predložena je nova arhitektura Mamba, koja se ističe kao možebitna zamjena za transformator, nudeći usporedive rezultate s manjom vremenskom složenošću. Cilj rada jest dubinska analiza arhitekture Mamba, uključujući njezine temeljne komponente i usporedbu s prethodno predloženim poboljšanjima arhitekture Transformatora. Implementirati arhitekturu Mamba u programskome jeziku Python. Vrednovati učinkovitost arhitekture na zadatcima modeliranja sljedova. Radu priložiti izvorni i izvršni kod radnog okvira, skupove podataka i programsku dokumentaciju te citirati korištenu literaturu.

Rok za predaju rada: 28. lipnja 2024.

Mojoj dragoj obitelji: mami, tati, sestrama i baki,

Hvala vam za svu podršku i ljubav koju ste mi pružali tijekom ovih pet godina studiranja. Kroz ovih pet godina, sa mnom ste dijelili sve moje suze, stres, ali i svu radost i sve lijepe trenutke. Svakodnevno ste mi bili oslonac i vjerovali u mene kad je i meni samoj nedostajalo vjere. Neizmjerno sam vam zahvalna na svemu i ovaj rad želim posvetiti upravo vama. Hvala vam od srca.

SADRŽAJ

1. Uvod

Duboko učenje grana je strojnog učenja, često poistovjećivana s pojmom neuronskih mreža [29]. Proteklih je desetljeća doživjela velik razvoj te transformirala način na koji pristupamo analizi podataka, obradi prirodnog jezika, računalnom vidu i mnogim drugim područjima. Duboko učenje obuhvaća skup tehnika za učenje iz podataka. U svojim početcima, te tehnike bile su inspirirane načinom rada ljudskog mozga, što je i dovelo do naziva *neuronske* mreže. Iako su prve mreže bile nadahnute ljudskom biologijom, razvoj današnjih neuronskih mreža rijetko kada slijedi tu inspiraciju. Umjesto biološkog pogleda, češće se koristi matematički pogled na duboko učenje, prema kojem se neuronske mreže opisuju kao parametrizirane diferencijabilne matematičke funkcije [29]. Parametri tih funkcija uče se optimizacijom određene funkcije cilja na temelju velikih količina podataka.

Značajan pokretač napretka dubokog učenja bilo je područje obrade prirodnog jezika (engl. *Natural Language Processing*, NLP). Područje obrade prirodnog jezika ili NLP obuhvaća tehnike i algoritme za obradu tekstnih podataka te ekstrakciju korisnih informacija iz istih. Samo neki od zadataka koji spadaju u ovo područje su strojno prevođenje [47, 53, 17], sažimanje teksta [52, 60, 46], prepoznavanje sentimenta tekstnih poruka [45, 54], detekcija govora mržnje [15, 26] i slično. Unutar tog područja, razvijene su tehnike i arhitekture mreža čiji je utjecaj izašao izvan okvira NLP-a te doprinio razvoju mnogih drugih područja dubokog učenja, uključujući računalni vid, obradu govora i sličnih.

Dominantan izbor arhitekture za većinu današnjih modela je arhitektura Transformatora (engl. *Transformers*). Nedugo nakon svog predstavljanja 2017. godine u radu *Attention Is All You Need* [56], počela je njezina dominacija u području NLP-a, a zatim i ostalim područjima unutar dubokog učenja. Zbog iznimne uspješnosti Transformatora, istraživačka zajednica posljednjih je godina većinu svojih radova posvetila poboljšanjima arhitekture Transformatora te mehanizma pozornosti (engl. *Attention Mechanism*) [58, 43, 18, 22, 25]. Mehanizam pozornosti mnogi smatraju jednim od ključnih razloga uspješnosti arhitekture Transformatora te se donedavno činilo da su

arhitekture koje koriste mehanizam pozornosti superiornije u odnosu na one koje ga ne koriste [13]. Međutim, u nedavnom radu predstavljena je nova arhitektura nazvana Mamba [33], koja se ne oslanja na mehanizam pozornosti, a pokazuje određene prednosti u odnosu na Transformatore. Bolja vremenska i prostorna složenost Mambe, uz rezultate usporedive onima dobivenih Transformatorima, privukla je pozornost istraživačke zajednice. Štoviše, dodatna intrigantnost ove arhitekture proizlazi iz njezine uske veze s povratnim modelima (engl. *Recurrent Neural Networks*, RNNs), arhitekturom koju su Transformatori *zamijenili*.

U ovom radu, implementirana je arhitektura Mamba u radnom okviru PyTorch [8]. Arhitektura je implementirana po uzoru na originalnu arhitekturu, s razlikom u programskom jeziku. Dok su autori originalnog rada koristili programski jezik C++ za implementaciju, za ovaj rad je odabran programski jezik Python. Programska izvedba arhitekture Mamba u Pythonu dodatno je optimizirana korištenjem programskog jezika Triton [55]. Također, u ovom radu dana je i analiza takve implementacije s aspekta njezine računalne učinkovitosti, ali i uspješnosti u zadatcima obrade prirodnog jezika.

Pregled arhitekture Mambe dan je u kontekstu zadatka modeliranja slijeda koji je ključan u današnjim zadatcima obrade prirodnog jezika. Više o ovom zadatku čitatelju je dano u poglavlju 3. Prije toga, u poglavlju 2, dan je pregled osnova dubokog učenja te povezanost modela dubokog učenja i modernih računala. U nadolazećim poglavljima slijede objašnjenja osnovnih pojmova vezanih za razumijevanje arhitekture Mambe, posebice strukturiranih modela u prostoru stanja u poglavlju 4, te njihove selektivne varijante u poglavlju 5. Konačno, poglavlje 6 donosi opis arhitekture i implementacijske detalje. Arhitektura je analizirana s aspekta vremenske i prostorne složenosti, ispitana na raznim zadatcima obrade prirodnog jezika, te uspoređena s Transformatorima u poglavlju 7.

2. Duboko učenje

Duboko učenje u uskoj je vezi s komponentama modernih računala, ponajviše procesorom i grafičkim procesorom računala. Arhitektura Mamba ističe se po svojoj sposobnosti iskorištavanja detalja operacija na grafičkim procesorima računala, što rezultira boljom računalnom izvedbom u usporedbi s Transformatorom.

Po uzoru na [30], u ovom poglavlju dan je pregled osnovne terminologije koja se koristi u području dubokog učenja, kao i osnove njegove primjene na modernim računalima.

2.1. Osnovna terminologija

Modele dubokog učenja definiramo kao parametrizirane diferencijabilne funkcije. Parametre modela obično označavamo s Θ. Učenje modela potraga je za optimalnim parametrima Θ.

Modele učimo na skupu od N primjera $\mathbf{x}^{(i)}$ i njima pridruženim željenim izlazima $y^{(i)}$. Takav skup nazivamo skupom za učenje i on je oblika $\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N}$. Ulazni primjeri $\mathbf{x}^{(i)}$ općenito su n-dimenzionalni vektori, $\mathbf{x}^{(i)} = (x_1^{(i)}, \ldots, x_n^{(i)})$. Osnovna ideja dubokog učenja je učenje reprezentacije podataka. Umjesto da učimo direktno na sirovim podacima, tražimo reprezentaciju podataka koja će rezultirati boljim radom modela. Na taj način, model istovremeno uči preslikavanje ulaznih primjera u željene izlaze, ali i kako najbolje iskoristiti ulazne primjere $\mathbf{x}^{(i)}$ u svrhu dobivanja željenih izlaza. Ulaz u model tada je $\phi(\mathbf{x}; \theta)$, gdje funkcija ϕ transformira ulazne podatke. Prema [30], na ovaj način imamo parametre θ kojima izabiremo konkretnu funkciju za reprezentaciju podataka ϕ te parametre w koji preslikavaju $\phi(\mathbf{x}; \theta)$ u željeni izlaz. Svi parametri modela tada su dani s $\Theta = \{w, \theta\}$.

Modeli dubokog učenja često su kompozicije jednostavnijih funkcija. Način povezivanja tih funkcija definira arhitekturu modela. Postoje razne arhitekture modela. Izbor arhitekture mreže određen je svojstvima ulaznih podataka te zadatku koji model treba naučiti.

Model kao kompoziciju funkcija možemo predstaviti na sljedeći način:

$$f(x;\theta,w) = f_{L+1}(f_L(f_{L-1}(\dots f_1(x;\phi(x;\theta)\dots;w_{L-1});w_L)) \tag{2.1}$$

Pri čemu svaku od funkcija $f_i, \forall i \in 1,\dots,L$ nazivamo slojem modela, gdje je L broj slojeva mreže ili dubina mreže. Zadnji sloj mreže f_{L+1} često nazivamo izlaznim ili završnim slojem, dok ostale slojeve nazivamo skrivenim slojevima mreže.

Uobičajeno se skriveni slojevi mreže označavaju pomoćnim varijablama h_i, dakle:

$$\begin{aligned} h_1 &= f_1(\phi(x;\theta);w_1) \\ &\vdots \\ h_i &= f_i(h_{i-1};w_i) \\ &\vdots \\ h_L &= f_L(h_{L-1};w_L) \\ f(x;\theta,w) &= o(h_L) \end{aligned} \tag{2.2}$$

Pomoćne varijable nazivamo skrivenim ili latentnim značajkama, ponekad i skrivenim stanjima, $h_i \in \mathbb{R}_i^d$, pri čemu d_i zovemo dubinom i-tog sloja.[1] Općenito, slojevi mogu biti proizvoljne diferencijabilne funkcije.

Učenje modela svodi se na pronalazak optimalnih parametara. Optimalni parametri su parametri koji minimiziraju funkciju cilja. Neka je $L(y,\hat{y})$ funkcija gubitka, odnosno mjera pogreške modela nad pojedinim primjerom.

Tada funkciju cilja možemo definirati kao empirijsku pogrešku čiju procjenu očekivanja na skupu od N primjera računamo na sljedeći način:

$$E(\mathcal{D};\Theta) = \frac{1}{N}\sum_{i=1}^{N} L(y^{(i)}, f(x^{(i)};\Theta)) \tag{2.3}$$

Prema tome optimalni su parametri oni za koje vrijedi: $\Theta^* = \underset{\Theta}{\operatorname{argmin}} E(\mathcal{D};\Theta)$.

Optimizacijski postupak tipično je varijanta gradijentnog spusta. Tehnika gradijentnog spusta iterativni je postupak kojim se nastoji poboljšati rješenje tako da se u svakoj iteraciji napravi korak u smjeru suprotnom gradijentu funkcije u trenutnoj točki. Gradijent funkcije $\nabla_\Theta E(\mathcal{D};\Theta)$ predstavlja smjer rasta funkcije E oko Θ. Budući da je cilj minimizacija funkcije E, uzimaju se koraci u smjeru suprotnom gradijentu.

[1] U literaturi je moguće primjetiti istovremeno korištenje oznake h_i za označavanje funkcije, značajke, ali i stanja. Unatoč toj nekonzistentnosti, većinom je i sam kontekst dovoljan za jasno određivanje značenja oznake h_i.

Dakle, ažuriranje parametara Θ svodi se na:

$$\Theta = \Theta - \eta \nabla_{\Theta} E(\mathcal{D}; \Theta), \tag{2.4}$$

gdje je η tzv. stopa učenja koja regulira veličinu koraka u smjeru suprotnom gradijentu.

U modelima dubokog učenja, koji su predstavljeni kao kompozicije velikog broja funkcija, jednostavan i računalno nezahtjevan način računanja gradijenata i ažuriranja parametara pojedinih slojeva naziva se algoritmom prosljeđivanja pogreške unatrag (engl. *Backpropagation Algorithm*).

U procesu optimizacije često se koristi metoda malih grupa (engl. *Mini-batch*), gdje se umjesto cijelog skupa podataka $\mathcal{D}$ za ažuriranje parametara modela koristi manji, nasumično odabrani podskup podataka veličine B, pri čemu je $B \ll N$. Ova metoda omogućuje brže i efikasnije učenje jer smanjuje računalne zahtjeve po iteraciji, a istovremeno može poboljšati generalizaciju modela zbog uvođenja stohastičnosti u optimizaciju.[2]

2.2. Utjecaj računalnih resursa na duboko učenje

U proteklim desetljećima, složenost zadataka koji modeli dubokog učenja uspješno rješavaju drastično se povećala. Trend je to koji se često pripisuje velikoj količini podataka u skupu za učenje te eksponencijalnom rastu veličine modela [30].[3] Međutim, kako veličina modela i količina podataka u skupu za učenje raste, tako je potrebno značajno više računalnih resursa za njihovo učenje.

U ranim fazama, kada su modeli imali manji broj parametara, njihovo učenje provodilo se koristeći samo procesor jednog računala. U današnje vrijeme, kada modeli imaju milijarde parametara, ovaj pristup više nije dovoljan, te se učenje većine modela provodi na grafičkim procesorima jednog ili više računala.

Procesor ili središnja jedinica za obradu podataka (engl. *Central Processing Unit*, CPU) dizajniran je za opću namjenu i izvršavanje manjeg broja operacija visoke složenosti jedne za drugom, slijedno.

S druge strane, grafički procesor ili grafička procesorska jedinica (engl. *Graphical Processing Units*, GPU) optimiziran je za paralelno izvršavanje velikog broja jednostavnijih zadataka istovremeno. Tu se ponajviše misli na zadatke koji ne zahtijevaju

[2] Generalizacija je sposobnost modela da daje dobre predikcije i za neviđene primjere, odnosno primjere izvan skupa za učenje $\mathcal{D}$.

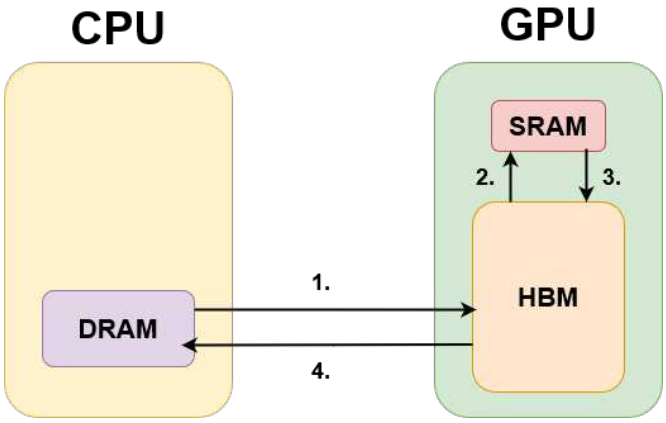
[3] Veličinu modela izražavamo kao broj njegovih parametara.

grananja u kodu, koji se mogu rastaviti na skup manjih podzadataka te potom istovremeno obavljati. Primjer takvog zadatka je matrično množenje. Budući da se algoritmi dubokog učenja često sastoje od velikog broja jednostavnih matematičkih operacija, paralelna priroda GPU-a omogućuje njihovo brže izvođenje u usporedbi s CPU-ima. Osim toga, GPU-ovi su opremljeni velikim brojem jezgara, što ih čini idealnim za istovremenu obradu velikih količina podataka, kao što su slike visoke rezolucije ili složeni skupovi podataka.

Moderni programski okviri za duboko učenje optimiziraju izvođenje dubokih neuronskih mreža putem GPU, omogućujući korisnicima lakši rad bez potrebe za detaljnim poznavanjem tehničkih karakteristika GPU-ova. Ova apstrakcija olakšava korisnicima fokusiranje na modeliranje, umjesto na tehničke aspekte izvođenja njihovih modela. Ipak, u potrazi za modelima s boljim računalnim performancama, primjećuje se postupno napuštanje tih apstrakcija te direktna optimizacija izvršavanja modela na GPU-u. Dakle, za visoko optimizirane modele, ipak je potrebno imati osnovno razumijevanje arhitekture GPU-a.

Kada razmotrimo arhitekturu GPU-a, primjećujemo da moderne grafičke procesorske jedinice sadrže memorije različitih veličina i brzina. Općenito, što je memorija većeg kapaciteta, to je sporija, dok su manje memorije brže. Za potrebe razumijevanja izvedbe dubokih modela, razmatrat ćemo dvije glavne memorije grafičkih procesorskih jedinica [25]. Prvi tip veća je memorija koja služi za pohranu podataka te se naziva memorijom visoke propusnosti (engl. *High Bandwidth Memory*, HBM). Osim nje, razmotrit ćemo i memoriju koja služi za izvršavanje operacija nad podatcima naziva statička memorija s proizvoljnim pristupom (engl. *Static Random Access Memory*, SRAM).

Proces izvođenja određene operacije na GPU-u može se konceptualno podijeliti na nekoliko koraka te je prikazan na slici 2.1. Prvo, ulazni podaci prenose se s glavne memorije računala (naziva CPU DRAM) na HBM. Zatim se ti podatci prenose iz memorije GPU HBM u bržu i manju memoriju SRAM. Nakon toga slijedi izvođenje željene operacije na GPU-u. Nakon završetka operacije, rezultati se prenose iz memorije GPU SRAM natrag u memoriju GPU HBM. Konačno, rezultati se prenose s GPU HBM-a natrag na CPU DRAM.



Slika 2.1: Prikaz slijeda potrebnih prijenosa podataka između različitih vrsta memorije tijekom izvršavanja jedne operacije na GPU-u. Prvi korak uključuje prijenos podataka iz CPU DRAM-a u GPU HBM. U drugom koraku, podaci se prenose iz HBM-a u SRAM. Nakon izvršene operacije, podatci se ponovno prenose iz SRAM-a u HBM, a zatim natrag u DRAM.

S obzirom na opisani proces, operacije koje se izvode na GPU možemo podijeliti u dvije glavne kategorije [25]:

1. operacije ograničene računalnim resursima (engl. *Compute-bound operations*),
2. operacije ograničene memorijom (engl. *Memory-bound operations*)

Operacije ograničene računalnim resursima su operacije u čijem vremenu izvođenja dominira vrijeme potrošeno na operacije aritmetičke prirode.

Operacije ograničene memorijom su operacije u čijem vremenu izvođenja dominira vrijeme potrošeno na operacije čitanja ili pisanja u memoriju.

Mogućnost prepoznavanja vrste operacije odgovorne za loše računalne izvedbe dubokih modela ključna je početna točka njihove optimizacije. Prepoznavanje vrste operacija odgovornih za nastajanje uskog grla u radu dubokih modela navodi nas na djelotvorne načine njihova rješavanja. Operacije sa sobom povlače drugačija rješenja te primjena standardnih prijedloga poboljšanja izvedbe modela, bez razumijevanja uzroka lošije izvedbe, ne bi nužno dovela do stvarnih poboljšanja. Na primjer, jači grafički procesori česti su prijedlog rješenja problema uzrokovanih operacijama ograničenih računalnim resursima. Međutim, takvo rješenje samo će djelomično ublažiti problem ako je usko grlo uzrokovano memorijskim ograničenjima. Dakle, za ispravnu optimizaciju rada modela, ključno je prepoznavanje operacija koje čine usko grlo njihova rada.

3. Modeliranje slijeda

Sraz između Transformatora i ostalih arhitektura počeo je rasti pojavom nove paradigme *predtreniranja* (engl. *Pretraining*) unutar NLP-a [59]. Predtreniranje podrazumijeva učenje modela na velikim skupovima podataka kako bi model stekao jezične kompetencije i opća znanja koja su korisna u različitim domenama. Kasnije se predtrenirani modeli mogu prilagoditi specifičnim domenama kroz postupak koji nazivamo fino prilagođavanje (engl. *Fine-tuning*). Takvi predtrenirani modeli često se nazivaju i temeljnim (engl. *foundation*) modelima [35]. U kontekstu obrade prirodnog jezika nazivamo ih velikim jezičnim modelima (engl. *Large Language Models*, LLMs) [59]. Predtreniranje modela temelji se na zadatku modeliranja slijeda, odnosno u kontekstu NLP-a, modeliranju jezika. Prema tome, zadatak modeliranja slijeda poslužit će nam kao mjerilo prikladnosti pojedine arhitekture kao osnovice *foundation* modela.

Modeliranje slijeda ili modeliranje nizova (engl. *Sequence Modeling*) zadatak je procjene vjerojatnosne distribucije sljedova koji se rješava statističkim metodama te metodama strojnog i dubokog učenja. Primjeri takvih slijednih podataka su vremenski podatci, tekstni podatci, nizovi DNK i slično. Procjena vjerojatnosne distribucije tih sljedova može se koristiti za dobivanje vjerojatnosti nekog slijeda ili za predviđanje sljedećeg elementa slijeda na temelju prethodnih.

Modeli za modeliranje slijeda (engl. *Sequence Models*) na svojemu ulazu dobivaju uređeni niz vektora $(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(L)})$, pri čemu je L duljina slijeda. Element slijeda na k-toj poziciji, $\mathbf{x}^{(t)}$, d-dimenzionalni je vektor, $\mathbf{x}^{(t)} \in \mathbb{R}^d$. U kontekstu obrade prirodnog jezika, ulazni d-dimenzionalni vektori obično su tzv. neuronske reprezentacije riječi (engl. *neural embeddings*). Svaka riječ iz zadanog vokabulara ima svoju jedinstvenu neuronsku reprezentaciju. Postupak mapiranja riječi na ove reprezentacije osmišljen je tako da reflektira stvarne semantičke odnose među riječima, što znači da mala udaljenost između reprezentacija u vektorskom prostoru odgovara maloj razlici u semantici tih riječi. Ove se reprezentacije nazivaju neuronskima jer se dobivaju učenjem neuronskih mreža na velikim količinama tekstnih podataka.

Zadatak je modela za modeliranje slijeda odrediti vjerojatnost zajedničkog doga-

đaja $p(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(L)})$. Na temelju pravila lanca vjerojatnosti, tražena vjerojatnost $p(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(L)})$ može se prikazati na sljedeći način:

$$p(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(L)}) = \prod_{i=1}^{L} p(\mathbf{x}^{(i)} | \mathbf{x}^{(1)}, \ldots \mathbf{x}^{(i-1)}), \tag{3.1}$$

pri čemu je $p(\mathbf{x}^{(i)} | \mathbf{x}^{(1)}, \ldots \mathbf{x}^{(i-1)})$ uvjetna vjerojatnost elementa $\mathbf{x}^{(i)}$ uvjetovana prethodnim elementima slijeda $(\mathbf{x}^{(1)}, \ldots \mathbf{x}^{(i-1)})$. Drugim riječima, ona nam daje do znanja koliko je vjerojatno pojavljivanje elementa $\mathbf{x}^{(i)}$ nakon pojave $(\mathbf{x}^{(1)}, \ldots \mathbf{x}^{(i-1)})$. Slijed $(\mathbf{x}^{(1)}, \ldots \mathbf{x}^{(i-1)})$ nazivamo i povijesti slijeda ili kontekstom za element $\mathbf{x}^{(i)}$.

Ovisno o konkretnom modelu strojnog učenja ili modelu dubokog učenja razlikuje se način dobivanja pojedinih $p(\mathbf{x}^{(i)} | \mathbf{x}^{(1)}, \ldots \mathbf{x}^{(i-1)})$, a time i način dobivanja konačne $p(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(L)})$. U nastavku će naglasak biti na modelima dubokog učenja.

Tijekom povijesti razvijen je velik broj arhitektura modela dubokog učenja pogodnih za modeliranje slijeda [27, 31, 24], a svaka od njih donosi određene prednosti i nedostatke u usporedbi s ostalima. Unatoč raznolikosti postojećih arhitektura tih modela, dominantna arhitektura za temeljne modele i dalje su Transformatori.

U ostatku poglavlja dan je pregled Transformatora, ali i povratnih modela, arhitekture koja je dominirala zadatkom modeliranja slijeda prije pojave Transformatora. Nedostatci tih modela poslužili su kao motivacija za Transformatore. No, nedostatci Transformatora u kombinaciji s prednostima povratnih modela motivirali su pojavu Mambe.

U potpoglavlju 3.1 dan je pregled jednostavnih povratnih modela. Potpoglavlje 3.2 donosi opis Transformatora. Razumijevanje ovih modela ključno je za razumijevanje važnosti i temeljnih značajki arhitekture Mambe.

3.1. Povratni modeli

Modeli s povratnom vezom ili povratni modeli (engl. *Recurrent Neural Networks*, RNN) posebna su klasa neuronskih mreža specijaliziranih za obradu sljedova [30]. Takvi modeli inspirirani su klasičnim dinamičkim sustavima u kojima model ažurira svoje stanje na temelju vanjskih poticaja.

Tijekom godina razvijen je velik broj ovakvih modela. Većina takvih modela nastala je nadogradnjama Elmanove mreže ili jednostavnog povratnog modela koji je, po uzoru na [41], opisan u potpoglavlju 3.1.1. Kao nedostatak tih modela često se navodi njihova slijedna priroda koja rezultira lošijim računalnim izvedbama. Više o računalnoj složenosti, kao i o računalnim izvedbi tih modela dano je u potpoglavlju 3.1.2

3.1.1. Jednostavni povratni modeli

U Elmanovim mrežama ili jednostavnim povratnim modelima (engl. *Simple Recurrent Neural Networks*, SRNN), stanje modela u trenutku t uobičajeno nazivamo skrivenim stanjem $\mathbf{h}^{(t)}$.

Neka je dimenzija elemenata ulaznog slijeda d_x. Dimenzija skrivenog stanja neka je d_h, a dimenzija izlaza neka je d_y. U vremenskom trenutku t, modeli na ulazu primaju element slijeda $\mathbf{x}^{(t)} \in \mathbb{R}^{d_x}$. Koristeći taj element i trenutno skriveno stanje $\mathbf{h}^{(t-1)} \in \mathbb{R}^{d_h}$, ažuriraju skriveno stanje na novu vrijednost $\mathbf{h}^{(t)} \in \mathbb{R}^{d_h}$. Izlaz modela $\mathbf{y}^{(t)} \in \mathbb{R}^{d_y}$ određuje se na temelju ažuriranog stanja $\mathbf{h}^{(t)}$.

Temeljna karakteristika ovih modela je mogućnost dijeljenja parametara kroz vremenske korake t. To znači da je svaki element izlaza iz ovakvih mreža određen na isti način, koristeći isto pravilo ažuriranja te iste matrice težina.

Formalno, povratni model definiran je sljedećim jednadžbama:

1. Ažuriranjem skrivenog stanja

$$\mathbf{h}^{(t)} = f(\mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} + b) \tag{3.2}$$

2. Generiranjem izlaza $\mathbf{y}^{(t)}$

$$\mathbf{y}^{(t)} = g(\mathbf{V}\mathbf{h}^{(t)} + c) \tag{3.3}$$

Pri čemu su f, g nelinearne aktivacijske funkcije, a $\mathbf{W} \in \mathbb{R}^{d_h \times d_h}, \mathbf{U} \in \mathbb{R}^{d_h \times d_x}, \mathbf{V} \in \mathbb{R}^{d_y \times d_h}, \mathbf{b} \in \mathbb{R}^{d_h}, \mathbf{c} \in \mathbb{R}^{d_y}$ parametri modela.

Zadatak matrice $\mathbf{U}$ je projicirati ulaz u prostor reprezentacije stanja. Analogno, zadatak matrice $\mathbf{V}$ je projicirati stanje u prostor reprezentacije izlaza. Matrica $\mathbf{W}$ određuje način na koji mreža koristi prethodni kontekst (prethodno viđene elemente slijeda) za dobivanje izlaza za trenutni ulaz.

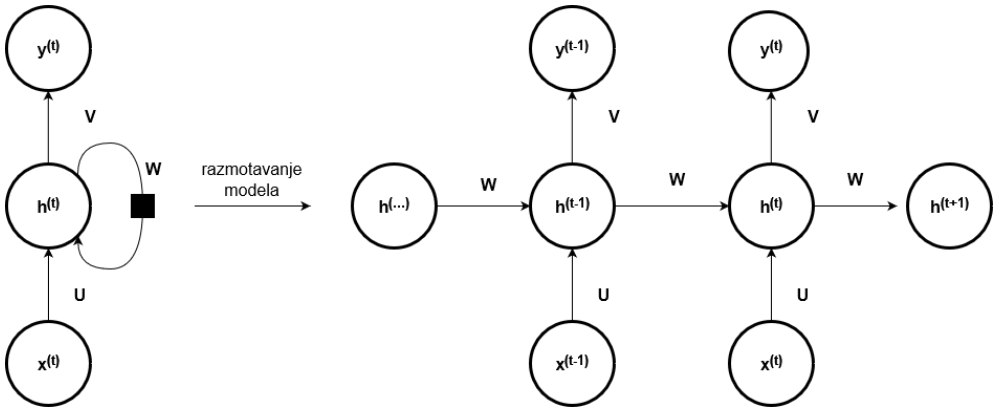
Često se za skup jednadžbi promjene skrivenog stanja te generiranja izlaznog elementa kaže da čine jednadžbe tzv. ćelije RNN-a. Prema tome, RNN-ovi su neuronske mreže koje u svojoj arhitekturi koriste ćelije RNN-a.

U svrhu modeliranja slijeda, funkcija g obično se definira kao funkcija $softmax$, tj. izraz (3.3) postaje $\mathbf{y}^{(t)} = softmax(\mathbf{V}\mathbf{h}^{(t)} + \mathbf{c})$. Na taj način, izlaz modela predstavlja vjerojatnosnu distribuciju elemenata slijeda. Drugi riječima, ako sa $\mathbf{y}_k^{(t)}$ označimo k-ti element izlaza, tada vrijedi:

$$p(\mathbf{x}^{(i)} = k | \mathbf{x}^{(1)}, \dots \mathbf{x}^{(i-1)}) = \mathbf{y}_k^{(t)} \tag{3.4}$$

Mreža na ovaj način koristi svoje skriveno stanje $\mathbf{h}^{(t)}$ kao svojevrsnu memoriju ili kontekst, unutar koje se pohranjuju sve informacije bitne za određivanje najvjerojatnijeg elementa izlaza. Postupkom učenja, mreža uči generirati kontekst koji će moći koristiti za dobivanje točnih izlaznih vrijednosti.

Računski graf jedan je od načina prikaza neuronske mreže. Predstavlja usmjereni graf u kojem korijenski čvorovi predstavljaju ulazne primjere, listovi izlaze modela, dok svi ostali čvorovi predstavljaju matematičke operacije. Usmjereni bridovi prikazuju protok informacija kroz neuronsku mrežu, od ulaza prema izlazima. Postupak uzastopne primjene pravila ažuriranja i generiranja izlaza uobičajeno se naziva *razmotavanjem* modela. Razmotavanje povratnog modela obuhvaća prikaz modela kroz više vremenskih koraka, uklanjajući rekurziju i prikazujući svaku iteraciju zasebno. Razmotavanje modela moguće je prikazati uporabom računskog grafa kao na slici 3.1.



Slika 3.1: Računski graf s primjerom razmotavanja jednostavnog povratnog modela.

Budući da je za dobivanje izlaza u trenutku t potrebno stanje iz prethodnog koraka $t-1$, kažemo da je prolaz unaprijed slijedne prirode. Slijedna priroda algoritma znači da algoritam zahtijeva slijedno provođenje operacija, element po element slijeda.

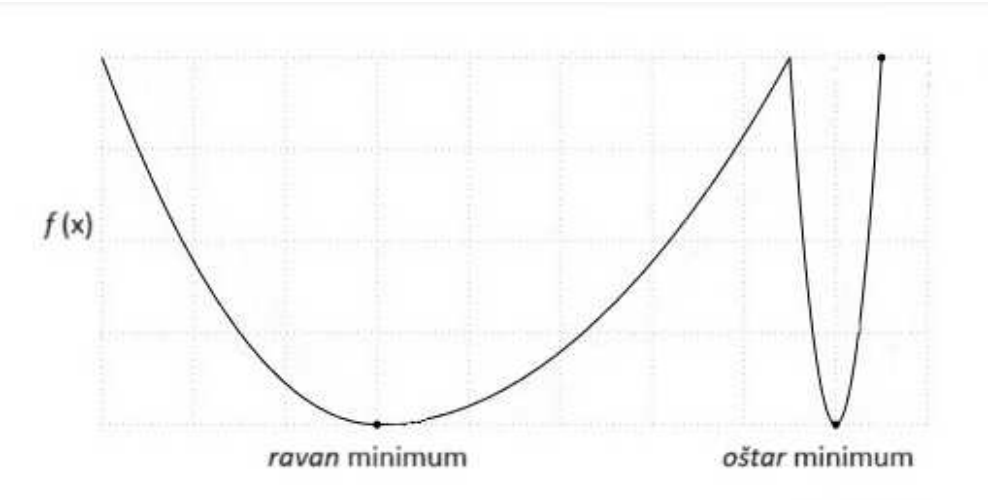
Učenje ovakve mreže provodi se algoritmom propagacije greške unatrag kroz vrijeme, što odgovara uobičajenom algoritmu učenja neuronskih mreža primijenjenom na razmotanom povratnom modelu. Ukratko, ovaj algoritam uključuje izračunavanje gradijenata greške modela u svakom vremenskom koraku i ažuriranje težina mreže kako bi se smanjila ukupna greška modela.

3.1.2. Računalna složenost

Jedna od prednosti povratnih modela linearna je vremenska i prostorna složenost u ovisnosti o duljini ulaznog slijeda. Drugim riječima, vrijeme i prostor potrebni za

obradu slijeda proporcionalni su njegovoj duljini. Međutim, činjenica da se elementi slijeda moraju obrađivati jedan po jedan otežava skaliranje ovih modela na veće skupove podataka i na slijedove većih duljina. Zbog slijednog načina obrade, nije moguće paralelno obrađivati više elemenata slijeda, što znači da obrada traje dulje nego kada bi to bilo moguće. Mogućnost paralelne obrade slijeda omogućila bi i bolje iskorištavanje svih pogodnosti grafičkih procesora. Nemogućnost skaliranja značajno je ograničenje povratnih modela.

Predloženi načini skaliranja ovih modela uključuju paralelnu obradu malih grupa [48]. Povećanjem broja primjera u grupi, vrijeme učenja ovih mreža se smanjuje, čime se poboljšavaju vremenska svojstva mreže. Međutim, smanjenje vremena učenja dolazi uz cijenu veće potrošnje memorije i smanjenih sposobnosti generalizacije modela [44]. Prema [44], uporaba manjih grupa navodi optimizacijski postupak konvergenciji k *ravnijim* (engl. flat) minimumima funkcije cilja, dok veće grupe navode optimizacijski postupak k oštrijim (engl. sharp) minimumima funkcije cilja. Jasno je da *oštriji* minimumi dovode do slabije generalizacije. Razlika *oštrih* i *ravnih* minimumi proizvoljne funkcije $f(x)$ prikazana je na slici 3.2. Parametri modela ažuriraju se na temelju gradijenta procjene očekivanja empirijske pogreške. Povećanje broja primjera na temelju kojih se računa procjena dovodi do bolje procjene. Rad s manjim grupama donosi procjene koje su manje točne, odnosno procijenjeni gradijenti sadrže više šuma. Međutim, procjene s više šuma imaju veću vjerojatnost pomaknuti težine izvan *oštrog* minimuma, odnosno usmjeriti algoritam učenja ka poželjnijem *ravnom* minimumu.



Slika 3.2: Razlika između *ravnih* i *oštrih* minimuma funkcije $f(x)$. Slika preuzeta s [44].

Dakle, za zadržavanje dobre generalizacije potrebno je imati manje grupe. Budući da i manji i veći broj primjera u grupi imaju svoje nedostatke po pitanju računalne iz-

vedbe, mogućnost skaliranja ovih modela paralelnom obradom grupa i dalje ne donosi izvedbu usporedivu s izvedbama modela koji mogu paralelno obrađivati elemente slijeda. Osim toga, iako teorija pokazuje da povratni modeli linearno skaliraju s obzirom na duljinu slijeda, važno je napomenuti praktične probleme koji postaju izraženiji sa stvarnim povećanjem duljine slijeda.

Pokazano je da ovi modeli mogu imati problema s učenjem zadataka koji uključuju dugoročne ovisnosti (engl. *Long-term Dependencies*) [19]. Prema [19], zadatak ima dugoročne ovisnosti ako željeni izlaz modela u trenutku t ovisi o ulazima koje je model obradio u puno ranijem trenutku $\tau \ll t$. Ovaj problem povezan je s nestabilnošću učenja ovih mreža i uzrokovan je nestajućim i eksplodirajućim gradijenatima.

Kod nestajućih gradijenata, vrijednosti gradijenata tijekom propagacije unazad kroz mrežu postaju izuzetno male. To rezultira vrlo sporim ili gotovo nikakvim učenjem ranijih slojeva mreže jer se težine ne ažuriraju učinkovito. S druge strane, kod eksplodirajućih gradijenata, vrijednosti gradijenata postaju izrazito velike, što može dovesti do nestabilnog treniranja jer se težine previše mijenjaju. Oba problema ometaju proces optimizacije i onemogućuju modelu da učinkovito uči dugoročne ovisnosti u podatcima.

Za rješavanje problema nestajućih i eksplodirajućih gradijenata predložene su brojne tehnike. Primjer jedne takve tehnike je i tehnika dodavanja tzv. propusnica (engl. *Gating Mechanisms*) u arhitekturu mreže. Dodavanjem propusnica dobivamo arhitekture s propusnicama. U takvim arhitekturama, osnovna ćelija RNN-a zamijenjena je s ćelijama s propusnicama. Popularne varijante takvih ćelija su ćelija s dugoročnim pamćenjem (engl. *Long Short-term Memory*, LSTM) [40] te propusna ćelija (engl. *Gated Recurrent Units*, GRU) [23].

Intuitivno, problem pamćenja dugoročnih ovisnosti može se povezati s fiksnom dimenzionalnošću skrivenog stanja povratnih modela [17]. Naime, model koristi skriveno stanje kao memoriju za pohranu informacija iz ulaznog slijeda. Stoga, pri povećanju duljine ulaznog slijeda, informacije kodirane u udaljenim trenutcima mogu biti prebrisane novijim. Ideja propusnica leži u reguliranju količine informacija koja će biti propuštena iz ulaznog elementa u skriveno stanje, s ciljem poboljšanja kvalitete generiranog konteksta.

Općenita definicija mehanizma propusnice dana je prema [35]:

$$\mathbf{h}^{(t)} = (1 - \sigma(\mathbf{z}))\mathbf{h}^{(t-1)} + \sigma(\mathbf{z})\mathbf{x}^{(t)} \tag{3.5}$$

gdje je $\mathbf{z}$ proizvoljan realan broj, a σ sigmoidna funkcija dana izrazom $\sigma(x) = \frac{1}{1+\exp(-x)}$. Pokazano je da dodavanje ovog mehanizma rješava problem nestajućih gradijenata. S

druge strane, problem eksplodirajućih gradijenata i dalje je prisutan, ali u praksi znatno rjeđi.

Iako ove metode ublažavaju probleme nestabilnog učenja, oni su i dalje prisutni. Drugim riječima, problem učenja dalekih ovisnosti i dalje postoji, samo je ublažen. Za više detalja o problemima učenja povratnih modela, čitatelja se upućuje na [50].

Važno je napomenuti da za ove arhitekture vrijede iste napomene vezane za računalne izvedbe kao i za jednostavne povratne modele. Odnosno, prati ih isti problem ograničenog skaliranja na veće skupove podataka te dulje sljedove uzrokovan slijednim načinom obrade elemenata slijeda.

3.2. Transformator

Ključna ideja i razlog uspješnosti Transformatora leži u uporabi mehanizma pozornosti (engl. *Attention Mechanism*). Mehanizam pozornosti omogućio je potpuno uklanjanje povratnih veza i slijedne obrade ulaznih nizova u Transformatorima. Od problema povratnih modela, osim povećane mogućnosti skaliranja, mehanizam pozornosti poboljšava i učenje dugoročnih ovisnosti [56]. Iako mehanizam pozornosti donosi brojne prednosti, isto tako predstavlja i usko grlo ove arhitekture zbog svoje visoke vremenske i prostorne složenosti.

Potpoglavlja 3.2.1 i 3.2.2 usmjerena su ka boljem razumijevanju mehanizma pozornosti i višestruke pozornosti. Uporaba tih mehanizama u kontekstu Transformatora objašnjena je u potpoglavlju 3.3.

3.2.1. Mehanizam pozornosti

Mehanizam pozornosti u općenitom slučaju definiramo na sljedeći način.

Neka je $D = \{(\mathbf{k}^{(1)}, \mathbf{v}^{(1)}), (\mathbf{k}^{(2)}, \mathbf{v}^{(2)}), \ldots, (\mathbf{k}^{(m)}, \mathbf{v}^{(m)})\}$ skup m parova vektora. Prvi član i-tog para $\mathbf{k}^{(i)} \in \mathbb{R}^d$ nazovimo ključem, a drugi član $\mathbf{v}^{(i)} \in \mathbb{R}^v$ vrijednosti. Nadalje, definirajmo vektor $\mathbf{q} \in \mathbb{R}^d$ i nazovimo ga vektorom upita. Neka su s d dane dimenzije ključeva i upita te neka v označava dimenziju vrijednosti.[1] Tada je mehanizam pozornosti primijenjen nad skupom D definiran kao:

$$Pozornost(\mathbf{q}, D) = \sum_{i=1}^{m} \alpha(\mathbf{q}, \mathbf{k}^{(i)})\mathbf{v}^{(i)} \tag{3.6}$$

gdje je α funkcija koja računa relevantnost ključa $\mathbf{k}^{(i)}$ za dani upit $\mathbf{q}$. Intuitivno, što

[1] Jednostavnosti radi uzimamo da su ključevi i upiti iste dimenzije d, no to nije nužno [56].

je ključ relevantniji za dani upit, to se veća težina daje vrijednosti tog ključa. Drugim riječima, relevantnijim ključevima daje se više pozornosti.

Vrijednost $\alpha(\mathbf{q}, \mathbf{k}^{(i)}) \in \mathbb{R}$ nazivamo težinom pozornosti. Težine pozornosti posjeduju sljedeća svojstva:

1. $\alpha(\mathbf{q}, \mathbf{k}^{(i)}) \geq 0, \forall i \in \{1, \ldots, m\}$
2. $\sum_{i=1}^{m} \alpha(\mathbf{q}, \mathbf{k}^{(i)}) = 1, \forall i \in \{1, \ldots, m\}$

Uobičajeni način osiguranja svojstava (1) i (2) postiže se njihovom normalizacijom na sljedeći način:

$$\alpha(\mathbf{q}, \mathbf{k}^{(i)}) = softmax(\mathbf{q}, \mathbf{k}^{(i)}) = \frac{\exp\big(\alpha(\mathbf{q}, \mathbf{k}^{(i)})\big)}{\sum_{i=j}^{m} \exp(\alpha(\mathbf{q}, \mathbf{k}^{(j)}))} \tag{3.7}$$

U Transformatoru, za funkciju α uzima se skalirani skalarni produkt vektora $\mathbf{k}^{(i)}$ i vektora upita $\mathbf{q}$:

$$\alpha(\mathbf{q}, \mathbf{k}^{(i)}) = \frac{\mathbf{q}^T \mathbf{k}^{(i)}}{\sqrt{d}} \tag{3.8}$$

Skalarni umnožak dvaju vektora vrijednost je između $-\infty$ i ∞, pri čemu će vrijednost biti to veća što su dva vektora međusobno sličnija. Skaliranje se provodi kako bi se izbjegle moguće nestabilnosti tijekom učenja prouzrokovane velikim vrijednostima skalarnog produkta.

Poopćavanjem na slučaj s n vektora upita dobivamo matricu upita $\mathbf{Q} \in \mathbb{R}^{n \times d}$. Ključeve i vrijednosti također organiziramo u matricu ključeva $\mathbf{K} \in \mathbb{R}^{m \times d}$ i matricu vrijednosti $\mathbf{V} \in \mathbb{R}^{m \times v}$.

Mehanizam pozornosti u Transformatoru tada je dan sljedećim izrazom:

$$Pozornost(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}})\mathbf{V} \in \mathbb{R}^{n \times v} \tag{3.9}$$

U kontekstu Transformatora, matrice upita, ključeva i vrijednosti dobivaju se linearnim transformacijama ulaznih i izlaznih elemenata slijeda. Parametri linearnih transformacija najčešće se slučajno inicijaliziraju, a potom uče standardnim algoritmom propagacije pogreške unatrag.

Umnožak matrica $\mathbf{Q}\mathbf{K}^T$ dovodi do visoke vremenske i prostorne složenosti ovog mehanizma. Točnije, složenost je $O(dnm)$ što se svodi na $O(nm)$ uz konstantnu dimenzionalnost upita i ključeva d. Pod pretpostavkom da je $d \ll n$ i $d \ll m$, prostorna složenost je $O(nm)$ zbog potrebe spremanja rezultirajuće matrice u memoriju.

3.2.2. Višestruki mehanizam pozornosti

Višestruki mehanizam pozornosti (engl. *Multi-Head Attention*, kraće MHA) podrazumijeva višestruku uporabu mehanizma pozornosti nad različitim transformacijama upita, ključeva i vrijednosti.

Svako pojedino izvođenje operacije pozornosti nazvat ćemo jednom *glavom* pozornosti. Svaka glava uzima isti početni skup upita, ključeva i vrijednosti te ih linearno transformira koristeći pripadni skup težina. Ako je s p_q označena dimenzija upita i ključeva pojedine glave te s p_v dimenzija vrijednosti pojedine glave, tada je transformacija početnih upita, ključeva i vrijednosti dana na sljedeći način:

$$\begin{aligned} \mathbf{q_i} &= \mathbf{W}_{qi}\mathbf{q} \in \mathbb{R}^{p_q} \\ \mathbf{k_i} &= \mathbf{W}_{ki}\mathbf{k} \in \mathbb{R}^{p_q} \\ \mathbf{v_i} &= \mathbf{W}_{vi}\mathbf{v} \in \mathbb{R}^{p_v} \end{aligned} \tag{3.10}$$

Gdje su $\mathbf{q_i}, \mathbf{k_i}, \mathbf{v_i}$ transformirane vrijednosti upita, ključeva i vrijednosti i-te glave, a matrice $\mathbf{W}_{qi} \in \mathbb{R}^{p_q \times d}, \mathbf{W}_{ki} \in \mathbb{R}^{p_q \times d}, \mathbf{W}_{vi} \in \mathbb{R}^{p_v \times v}$ pripadne su težine linearne transformacije i-te glave.Težine $\mathbf{W}_{qi}, \mathbf{W}_{ki}, \mathbf{W}_{vi}$ uče se standardnim algoritmom propagacije pogreške unatrag.

Nakon transformacije, dobivene upite, ključeve i vrijednosti možemo posložiti u pripadnu matricu upita $\mathbf{Q}_i \in \mathbb{R}^{n \times p_q}$, matricu ključeva $\mathbf{K}_i \in \mathbb{R}^{m \times p_q}$ te matricu vrijednosti $\mathbf{V}_i \in \mathbb{R}^{m \times p_v}$. Tada je rezultate i-te glave, odnosno i-te uporabe mehanizma pozornosti dan s:

$$\mathbf{h_i} = softmax(\frac{\mathbf{Q}_i\mathbf{K}_i^T}{\sqrt{p_q}})\mathbf{V}_i \in \mathbb{R}^{n \times p_v} \tag{3.11}$$

Neka je h ukupan broj glava. Neka je matrica $\mathbf{H} \in \mathbb{R}^{n \times hp_v}$ definirana kao matrica združenih rezultata pojedinih glava.[2] Tada je konačan rezultat višestrukog mehanizma pozornosti linearna transformacija združenih izlaza glava, odnosno:

$$MHA(\mathbf{Q}, \mathbf{K}, V) = \mathbf{H}{\mathbf{W_o}}^T \in \mathbb{R}^{n \times p_o} \tag{3.12}$$

Gdje je $\mathbf{W_o} \in \mathbb{R}^{p_o \times hp_v}$ matrica čiji je zadatak preslikati združene rezultate glava u prostor dimenzije p_o.

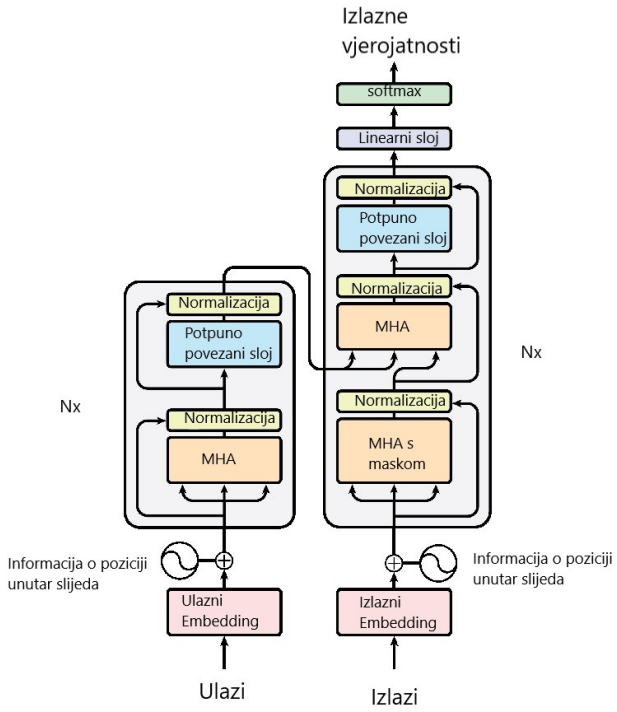
3.2.3. Opis arhitekture

Model arhitekture Transformatora sastoji se od dvije zasebne neuronske mreže.

[2]Za svaki upit $\mathbf{q}$ združujemo rezultate svake pojedine glave, što odgovora vertikalnoj konkatenaciji matrica rezultata pojedinih glava

Prva mreža naziva se mreža čitača ili koder (engl. *Encoder*). Zadatak te mreže je izgraditi najkvalitetniju moguću reprezentaciju ulaznog niza. Druga mreža naziva se mreža pisača ili dekoder (engl. *Decoder*). Mreža pisača koristi reprezentaciju mreže čitača kako bi generirala točne izlaze, oslanjajući se pritom i na svoje prethodno generirane rezultate.

Slika 3.3 prikazuje arhitekturu modela.



Slika 3.3: Prikaz arhitekture modela Transformatora. Naglasak je stavljen na tri različite uporabe sloja višestruke pozornosti (označene s *Multi-Head Attention*). Slika preuzeta iz [56].

Obje mreže, čitač i pisač, sastoje se od više identičnih blokova.[3] Svaki blok predstavlja jednu višeslojnu mrežu koja preslikava ulazni slijed elemenata duljine L, $(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(L)})$, u izlazni slijed elemenata $(\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(L)})$ duljine L.[4] Kada je riječ o jezičnim podatcima, elementi ulaznih i izlaznih slijedova najčešće su neuronske reprezentacije riječi. Blokovi se nadovezuju jedan na drugi, tako da izlaz jednog bloka postaje ulaz sljedećeg bloka.

Svaki blok sastoji se od potpuno povezanih slojeva, rezidualnih veza, normalizacijskih slojeva te slojeva višestruke pozornosti. Rezidualne ili povratne veze omogućuju

[3] Mreža čitač sastoji se od N identičnih *koderskih* blokova, mreža pisač sastoji se od N identičnih *dekoderskih* blokova

[4] Radi jednostavnosti, pretpostavljamo da je duljina ulaznog slijeda jednaka duljini izlaznog slijeda, no to nije nužno.

lakši protok gradijenata kroz mrežu, čime se olakšava treniranje dubokih modela te poboljšava učinkovitost učenja. Normalizacijski slojevi uvode se kako bi stabilizirali i ubrzali proces učenja, osiguravajući konzistentnu distribuciju aktivacija kroz mrežu. Slojevi višestruke pozornosti srž su blokova čitača i pisača te će na njih biti usmjeren ostatak potpoglavlja.

Mehanizam višestruke pozornosti (MHA) primjenjuje se i u mreži čitača i u mreži pisača, no s različitim ciljevima.

U mreži čitača, MHA se koristi za izgradnju reprezentacije ulaznog niza. U tom procesu, svaki element ulaza (ili izlaz prethodnog sloja) ima pridružene vektore upita, ključa i vrijednosti. Reprezentacija svakog elementa nastaje kao težinska kombinacija njegovog upita s vrijednostima ostalih elemenata u slijedu, prema izrazu danom u 3.12. Na taj način, reprezentacija svakog ulaznog elementa slijeda ovisi o samom tom elementu.

U mreži pisača, MHA ima dvojaku ulogu.

Prvi podsloj s višestrukom pozornosti mreža pisača koristi za generiranje izlaza na temelju reprezentacija dosad generiranih izlaza i ulaznih reprezentacija. U ovom slučaju, svaki element izlazne reprezentacije djeluje kao upit, dok svaki element ulazne reprezentacije ima pridružene vektore ključa i vrijednosti. U zadatcima kao što je modeliranje slijeda, bitno je da model ne koristi buduće elemente izlaza. U tim slučajevima, ovaj podsloj koristi dodatnu masku koja sprječava korištenje budućih elemenata izlaza. Ovaj mehanizam omogućuje mreži pisača da, prilikom generiranja svakog novog elementa izlaza, uzme u obzir relevantne dijelove ulaznog niza. Za razliku od povratnih modela, gdje se isti kontekst koristi za sve izlazne elemente, ovdje svaki izlazni element ima zaseban kontekst koji se fokusira na relevantne dijelove ulaza.

Drugi podsloj s višestrukom pozornosti zadužen je za izradu kvalitetne reprezentacije dosad generiranih izlaza (ili elemenata prethodnog sloja). Na sličan način kao u mreži čitača, ovaj podsloj omogućava modelu da analizira kontekst dosad generiranih podataka te stvori kontekstno ovisnu reprezentaciju svakog elementa izlaznog slijeda.

Za dobivanje vjerojatnosne distribucije elemenata slijeda, izlaz Transformatora (zadnjeg bloka mreže pisača), propušta se kroz linearni sloj te se uporabom aktivacijske funkcije $softmax$ dobiva tražena vjerojatnosna distribucija.

3.2.4. Računalna složenost

Transformatori su postali popularni za modeliranje slijednih podataka zbog svojih izvanrednih računalnih izvedbi i skalabilnosti.

Jedna od glavnih prednosti Transformatora je njihova sposobnost paralelne obrade podataka. Za razliku od povratnih modela, transformatori ne obrađuju elemente ulaznog slijeda jedan po jedan, već mogu paralelno obraditi sve elemente ulaznog slijeda koristeći prethodno opisani mehanizam pozornosti. Ovaj pristup značajno smanjuje vrijeme potrebno za obradu dugih sljedova.

Iako mehanizam pozornosti donosi veću proširivost (engl. *Scalability*) modela, on je u isto vrijeme i usko grlo ovih modela. Naime, zbog njega, složenost transformatora u odnosu na duljinu slijeda je kvadratna. To je glavna mana transformatora u usporedbi s povratnim modelima koji imaju linearnu složenost. Ipak, prednosti paralelizacije često nadmašuju ovaj nedostatak u praksi, ponajviše zbog grafičkih procesora koji postaju sve učinkovitiji u paralelnom izvršavanju operacija.

Tijekom godina, predložene su brojne varijante mehanizma pozornosti s ciljem poboljšanja visoke vremenske i prostorne složenosti [58, 25, 42]. Iako donose linearne složenosti, pokazalo se da ove varijante nisu jednako uspješne kao originalni mehanizam pozornosti [33].

4. Modeli u prostoru stanja

Model u prostoru stanja (engl. *State Space Model*, SSM) linearna je transformacija ulaznih signala koja istovremeno posjeduje svojstva modela s povratnom vezom te konvolucijskih modela (engl. *Convolutional neural networks*, CNN). U ovom poglavlju, modele u prostoru stanja promatrat ćemo u kontekstu modela za modeliranje slijeda. U tom slučaju, modeli u prostoru stanja nazivaju se i slijednim modelima u prostoru stanja (engl. *State Space Sequence Models*, kraće SSSM).[1]

Pregled slijednih modela u prostoru stanja dan je po uzoru na [32]. Radi jednostavnosti, u nastavku ćemo se referirati na slijedne modele u prostoru stanja samo kao modele u prostoru stanja te koristiti kraticu SSM.

Modeli u prostoru stanja imaju tri međusobno ekvivalentne formulacije: vremenski kontinuiranu, povratnu te konvolucijsku formulaciju. Formulacije modela SSM-a opisane su u narednim potpoglavljima.

4.1. Vremenski kontinuirana formulacija

Osnovni oblik modela u prostoru stanja djeluje nad kontinuiranim signalima te ga nazivamo vremenski kontinuiranim modelom u prostoru stanja ili vremenski kontinuiranom formulacijom modela.

$$\begin{aligned} \frac{d}{dt}h(t) &= \mathbf{A}h(t) + \mathbf{B}x(t) \\ y(t) &= \mathbf{C}h(t) + \mathbf{D}x(t) \end{aligned} \tag{4.1}$$

Jednadžbe vremenski kontinuiranog modela (4.1) definiraju linearno preslikavanje kontinuiranog jednodimenzionalnog ulaznog signala $\mathbf{x}(t) \in \mathbb{R}$ u kontinuirani jednodimenzionalni izlazni signal $y(t) \in \mathbb{R}$ preko skrivenog stanja $h(t) \in \mathbb{R}^{d_h}$.

Matrice $\mathbf{A} \in \mathbb{R}^{d_h \times d_h}, \mathbf{B} \in \mathbb{R}^{d_h \times 1}, \mathbf{C} \in \mathbb{R}^{1 \times d_h}, \mathbf{D} \in \mathbb{R}$ parametri su kontinuiranog

[1] Izraz *model u prostoru stanja* označava različite pojmove, ovisno o području unutar kojeg se spominje. Ovdje je dana definicija u skladu s ostatkom rada.

modela. Dimenzionalnost skrivenog prostora tada je označena s d_h, dok je dimenzionalnost ulaznog i izlaznog prostora dana s d.

Da bi se definicija proširila na višedimenzionalne ulazne $\mathbf{x} \in \mathbb{R}^d$ i izlazne signale $y \in \mathbb{R}^d$, jednadžbe kontinuiranog modela SSM-a primjenjuju se zasebno na svaku dimenziju ulaznog signala.

Dakle, za preslikavanje kontinuiranog ulaznog signala $\mathbf{x} \in \mathbb{R}^d$ u kontinuirani izlazni signal $y \in \mathbb{R}^d$ preko skrivenog stanja $h \in \mathbb{R}^{d_h}$, za svaku dimenziju d koristit će se zasebni parametri $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ i $\mathbf{D}$.

Parametri modela $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ redom se nazivaju matricom stanja, matricom ulaza, matricom izlaza te matricom povratne veze.

Matricom stanja $\mathbf{A}$ modeliramo utjecaj skrivenog stanja na vlastite promjene tijekom vremena. Ulazni signal na skriveno stanje utječe preko matrice ulaza $\mathbf{B}$. Izlazna matrica $\mathbf{C}$ opisuje utjecaj skrivenog stanja na izlazni signal, dok matrica $\mathbf{D}$ modelira izravan utjecaj ulaznog signala na izlazni.

U praksi se često, radi jednostavnosti, parametar $\mathbf{D}$ izbacuje iz formulacije modela u prostoru stanja. Prema tome, izraz (4.1) postaje:

$$\begin{aligned} \frac{d}{dt}h(t) &= \mathbf{A}h(t) + \mathbf{B}\mathbf{x}(t) \\ y(t) &= \mathbf{C}h(t) \end{aligned} \tag{4.2}$$

Kako bismo ovakve modele mogli koristiti za stvarne slijedne podatke, koji predstavljaju diskretne signale, vremenski kontinuirani oblik (4.2) potrebno je najprije diskretizirati.

4.2. Povratna formulacija

Diskretizacijom vremenski kontinuiranog modela SSM-a dobivamo povratnu formulaciju modela koju potom možemo koristiti kao model za slijedne podatke.

Diskretizacija podrazumijeva transformaciju parametara modela $\mathbf{A}, \mathbf{B}, \mathbf{C}$ u njihove diskretne inačice $\overline{\mathbf{A}}, \overline{\mathbf{B}}, \overline{\mathbf{C}}$ s pomoću diskretizacijskog pravila i dodatnog skalarnog parametra, tzv. koraka diskretizacije Δ. Korak diskretizacije predstavlja rezoluciju ulaza. Konceptualno, ulaze $\mathbf{x}^{(k)}$ možemo zamisliti kao uzorke uniformno uzorkovane iz implicitnog kontinuiranog signala $x(t)$ tako da je $x^{(k)} = x(k\Delta)$. Bitno je napomenuti kako diskretizaciju obavljamo nad svakom od dimenzija ulaznog signala zasebno. Dakle, za d dimenzionalne signale imat ćemo d koraka diskretizacije. Diskretizacijsko pravilo skup je fiksnih funkcija (f_A, f_B, f_C) koje na temelju fiksnog koraka diskretiza-

cije Δ transformiraju parametre modela:

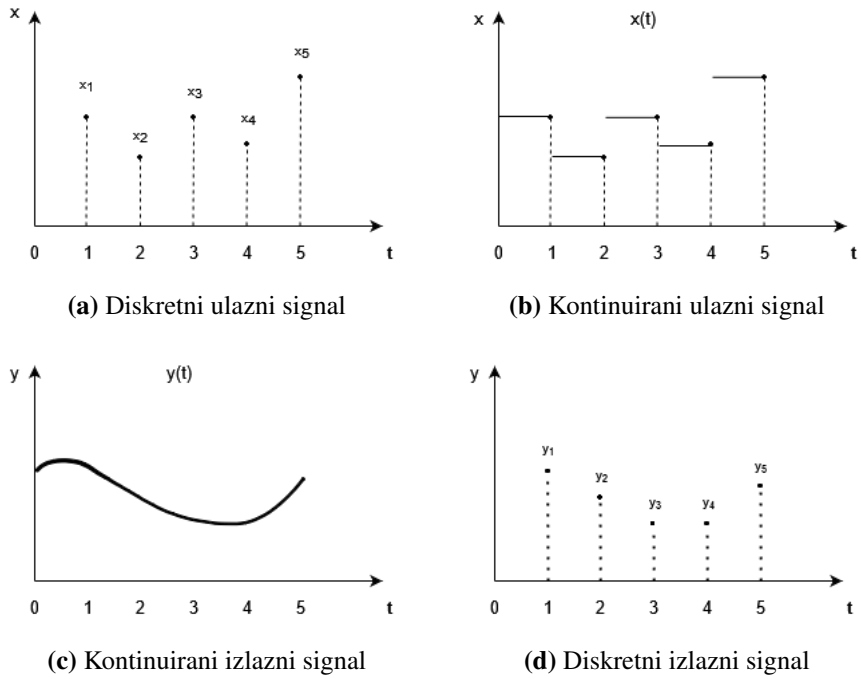
$$\overline{\mathbf{A}} = f_A(\mathbf{A}, \Delta)$$
$$\overline{\mathbf{B}} = f_B(\mathbf{B}, \Delta)$$
$$\overline{\mathbf{C}} = f_C(\mathbf{C}, \Delta)$$

Matrice $\overline{\mathbf{A}}, \overline{\mathbf{B}}, \overline{\mathbf{C}}$ stalne su za svaki element ulaznog slijeda $x^{(k)}$. Svojstvo stalnih parametara modela za sve vremenske korake naziva se se linearnom vremenskom nepromjenjivosti (engl. Linear Time Invariance, LTI).

Od brojnih diskretizacijskih metoda izdvojit ćemo metodu zvanu *zero-order hold* ili ZOH. Diskretizacijsko pravilo prema ZOH dano je sljedećim preslikavanjima:

$$\begin{aligned} \overline{\mathbf{A}} &= \exp(\Delta \cdot \mathbf{A}) \\ \overline{\mathbf{B}} &= (\Delta \cdot \mathbf{A})^{-1}(\exp(\Delta \cdot \mathbf{A})) \cdot \Delta \cdot \mathbf{B} \\ \overline{\mathbf{C}} &= \mathbf{C} \end{aligned} \tag{4.3}$$

Intuitivno, primjenu vremenski kontinuiranog modela u prostoru stanja nad diskretnim signalima možemo shvatiti kao slijed koraka vidljivih na slici 4.1 [12]. Neka je ulaz diskretni signal kao na slici 4.1a. Takav diskretni signal transformiramo u kontinuiran signal tako da svaki uzorak zadržavamo sve dok ne dobijemo sljedeći uzorak. U tom slučaju, Δ predstavlja vrijeme zadržavanja uzorka. Na slici 4.1b vidljiv je dobiveni kontinuirani signal, koji je step funkcija s veličinom koraka Δ. Nakon transformacije u kontinuirani signal, primjenjujemo jednadžbe kontinuiranog modela dane u (4.2). Dobiveni izlazni signal u tom je slučaju kontinuiran. Neka je kontinuirani izlazni signal nalik na signal na slici 4.1c. Takav signal potrebno je transformirati u diskretni signal. Transformaciju izlaznog signala u diskretni signal dobit ćemo uzimanjem njegovih uzoraka s istom stopom uzorkovanja Δ. Konačno, takav dobiveni diskretni izlazni signal vidljiv je na 4.1d. Primjena vremenski kontinuiranog modela SSM-a nakon transformacije diskretnog signala u kontinuirani potpuno je ekvivalentna izravnoj primjeni modela SSM-a s diskretiziranim parametrima nad originalnim diskretnim ulaznim signalom.



(a) Diskretni ulazni signal **(b)** Kontinuirani ulazni signal

(c) Kontinuirani izlazni signal **(d)** Diskretni izlazni signal

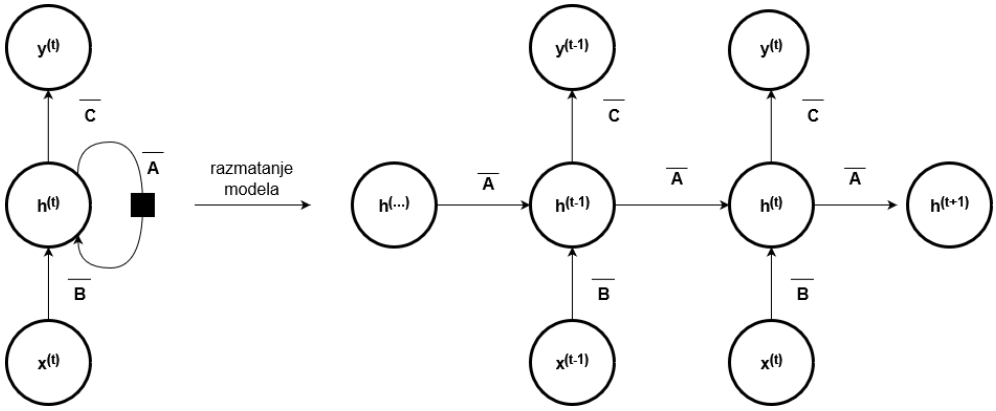
Slika 4.1: Korištenje kontinuiranog modela SSM-a nad diskretnim signalima. Inutitivno se to može predočiti kroz postupak transformacije početnog diskretnog signala (danog u 4.1a) u kontinuirani signal (dan u 4.1b). Primjenom izraza za kontinuirani model SSM-a, dobiveni kontinuirani izlazni signal (dan u 4.1c) potom se uniformno uzrokuje kako bi se dobio konačni diskretni izlazni signal (dan u 4.1d). Dobiveni izlazni signal ekvivalentan je izlaznom signalu koji bismo dobili izravnom primjenom modela SSM-a s diskretiziranim parametrima.

Oblik dobiven nakon diskretizacije predstavlja povratnu formulaciju modela te je ona dana sa sljedećim jednadžbama:

$$\begin{aligned} \mathbf{h}^{(t)} &= \overline{\mathbf{A}}\mathbf{h}^{(t-1)} + \overline{\mathbf{B}}\mathbf{x}^{(t)} \\ \mathbf{y}^{(t)} &= \overline{\mathbf{C}}\mathbf{h}^{(t)} \end{aligned} \tag{4.4}$$

U povratnom načinu rada SSM-a, skriveno stanje $\mathbf{h}^{(t)}$ računa se na temelju prethodnog skrivenog stanja $\mathbf{h}^{(t-1)}$. Utjecaj prethodnog skrivenog stanja na novo skriveno stanje reguliran je matricom stanja $\overline{\mathbf{A}}$. Utjecaj trenutnog elementa ulaza $\mathbf{x}^{(t)}$ na skriveno stanje modeliran je s pomoću matrice ulaza $\overline{\mathbf{B}}$. Skriveno stanje $\mathbf{h}^{(t)}$ predstavlja sažetak ulaznog slijeda.

Slično kao i kod povratnih modela, konkretan utjecaj svakog od parametara modela na promjene skrivenog stanja $\mathbf{h}^{(t)}$ i izlaznih elemenata slijeda $\mathbf{y}^{(t)}$ moguće je vidjeti *razmotavanjem* modela. Navedeni postupak prikazan je na slici 4.2.



Slika 4.2: Prikaz povratnog načina rada SSM-a kroz više vremenskih koraka. U svakom trenutku t, skriveno stanje modela $\mathbf{h}^{(t)}$ računa se na temelju prethodnog skrivenog stanja $\mathbf{h}^{(t-1)}$ te trenutnog elementa ulaza $\mathbf{x}^{(t)}$. Trenutno skriveno stanje $\mathbf{h}^{(t)}$ potom se koristi za dobivanje novog izlaznog elementa $\mathbf{y}^{(t)}$.

Razmatanjem modela moguće je primijetiti sličnost između SSM-a i RNN-a. Na prvi pogled, računski grafovi razmotanih modela izgledaju identično, osim u nazivlju parametara. Istu sličnost moguće je primjetiti usporedbom izraza za SSM danog u (4.4) te izraza za RNN danih u (3.2) i (3.3). No, postoje određene suptilne razlike.

Prva razlika u odnosu na jednostavne povratne modele vidljiva je u jednadžbama ažuriranja stanja. Naime, u općenitom slučaju, povratni modeli nelinearne su transformacije sljedova, što je vidljivo uporabom nelinearnih aktivacijskih funkcija u jednadžbi ažuriranja skrivenog stanja. S druge strane, modeli u prostoru stanja isključivo su linearne transformacije.

Usporedimo li SSM s linearnim povratnim modelom, odnosno povratnim modelom koji ne koristi nelinearne aktivacijske funkcije, razlika i dalje postoji.

Budući da je SSM u svom osnovnom obliku kontinuirani model, parametri diskretnog modela SSM-a u povratnoj formulaciji definirani su kroz postupak diskretizacije. To znači da su parametri tih modela povezani dodatnim parametrom koraka diskretizacije, Δ, kroz odabranu metodu diskretizacije.

Međutim, zanimljivo je napomenuti da se može pokazati da mehanizmi propusnica predstavljaju jedan oblik oblik diskretizacije [32]. Prema tome, modeli SSM-a u povratnoj formulaciji mogu se shvatiti i kao generalizacija običnih povratnih modela.

Unatoč tome, praksa je pokazala da modeli SSM-a, kod kojih je matrica stanja $\mathbf{A}$ inicijalizirana posebnim metodama, postižu bolje rezultate od dosadašnjih povratnih arhitektura u zadatcima s dugoročnim ovisnostima [36, 38, 37]. Više o tome bit će objašnjeno kasnije.

Model prostora stanja i povratni model oboje imaju linearnu vremensku složenost s obzirom na duljinu slijeda. Dakle, za generiranje jednog elementa izlaza potrebno je, osim parametara modela, prethodno skriveno stanje $\mathbf{h}^{(t-1)}$ te trenutni element ulaza $\mathbf{x}^{(t)}$. To znači da za ulazni slijed duljine L, ova operacija ima vremensku složenost $\mathcal{O}(L)$. Međutim, kao i kod povratnih modela, povratna formulacija modela SSM-a je ograničene proširivosti zbog nemogućnosti paralelne obrade više elemenata slijeda. Ipak, izostanak nelinearnih aktivacijskih funkcija omogućuje bolje skaliranje ovih modela na dulje slijedove i veće skupove podataka u usporedbi s klasičnim jednostavnim povratnim modelima.

Razmatranjem samo povratne formulacije, poznato je postojanje efikasnih algoritama za paralelno izvršavanje linearnih povratnih modela [48]. Međutim, ovaj način i dalje zahtijeva materijalizaciju skrivenog stanja dimenzija $(L \times d \times d_h)$, što je d_h puta više od dimenzionalnosti ulaznog i izlaznog slijeda, $(L \times d)$ [33]. Dakle, uz korištenje isključivo povratne formulacije, proširivost je i dalje ograničena kao i kod povratnih modela.

Glavna prednost modela u prostoru stanja proizlazi iz postojanja druge formulacije modela, koja je potpuno ekvivalentna povratnoj formulaciji. Ta druga formulacija naziva se konvolucijskom formulacijom modela. Prednost ove formulacije je u jednostavnoj paralelizaciji te znatno većoj mogućnosti skaliranja.

4.3. Konvolucijska formulacija

Konvolucijska formulacija modela lako se izvodi analitičkim razmatanjem povratne formulacije modela.

Raspisivanjem nekoliko koraka rekurzije:

$$\begin{aligned}
&\mathbf{h}^{(0)} = \overline{\mathbf{B}}\mathbf{x}^{(0)} \quad && \mathbf{h}^{(1)} = \overline{\mathbf{AB}}\mathbf{x}^{(0)} + \overline{\mathbf{B}}\mathbf{x}^{(1)} && \mathbf{h}^{(2)} = \overline{\mathbf{A}}^2\overline{\mathbf{B}}\mathbf{x}^{(0)} + \overline{\mathbf{AB}}\mathbf{x}^{(1)} + \overline{\mathbf{B}}\mathbf{x}^{(2)} \\
&\mathbf{y}^{(0)} = \overline{\mathbf{CB}}\mathbf{x}^{(0)} \quad && \mathbf{y}^{(1)} = \overline{\mathbf{CAB}}\mathbf{x}^{(0)} + \overline{\mathbf{CB}}\mathbf{x}^{(1)} && \mathbf{y}^{(2)} = \overline{\mathbf{CA}}^2\overline{\mathbf{B}}\mathbf{x}^{(0)} + \overline{\mathbf{CAB}}\mathbf{x}^{(1)} + \overline{\mathbf{CB}}\mathbf{x}^{(2)}
\end{aligned} \tag{4.5}$$

Vidljivo je postojanje zatvorene forme izlaznog signala, ovisno samo o ulazima i parametrima modela:

$$\mathbf{y}^{(k)} = \overline{\mathbf{CA}}^k\overline{\mathbf{B}}\mathbf{x}^{(0)} + \overline{\mathbf{CA}}^{k-1}\overline{\mathbf{B}}\mathbf{x}^{(1)} ... + \overline{\mathbf{CAB}}\mathbf{x}^{(k-1)} + \overline{\mathbf{CB}}\mathbf{x}^{(k)} \tag{4.6}$$

Ako definiramo operaciju konvolucije na sljedeći način:

$$\mathbf{y}^{(n)} = \sum_{k=0}^{n} \mathbf{h}^{(k)}\mathbf{K}^{(n-k)} \tag{4.7}$$

Onda dobivamo sljedeću konvolucijsku inačicu modela u prostoru stanja:

$$\begin{aligned} \mathbf{y} &= \mathbf{h} * \overline{\mathbf{K}} \\ \overline{\mathbf{K}} &= (\overline{\mathbf{CB}}, \overline{\mathbf{CAB}}, ..., \overline{\mathbf{CA}}^{L-1}\overline{\mathbf{B}}) \end{aligned} \tag{4.8}$$

gdje je $*$ operator konvolucije nad diskretnim signalima $\mathbf{h}$ i $\overline{\mathbf{K}}$. Element rezultirajućeg izlaznog signala $\mathbf{y}$ na n-toj poziciji dan je izrazom (4.7). U operaciji konvolucije $\overline{\mathbf{K}}$ se naziva konvolucijskom jezgrom. Konvolucijska jezgra često se još naziva jezgrom ili filtrom SSM-a ili jezgrom u prostoru stanja (*state space kernel*, SSK).

Konvolucijskim načinom rada izbjegavamo izračun i pohranu skrivenih stanja veličine $(L \times d \times d_h)$. Umjesto toga, pohranjujemo samo konvolucijsku jezgru veličine $(L \times d)$, što omogućuje znatno ubrzanje i smanjuje memorijsku potrošnju u usporedbi s povratnim načinom rada modela. Kako bi konvolucijska formulacija bila efikasnija, nužno je postavljanje dodatnih uvjeta na strukturu matrice stanja $\mathbf{A}$.

4.4. Strukturirani modeli u prostoru stanja

Modeli u prostoru stanja koji postavljaju dodatne uvjete na strukturu matrice stanja nazivaju se strukturirani modeli u prostoru stanja (engl. *Structured state space models*, SSSM).

Najčešće se za matricu stanja $\mathbf{A}$ koristi posebno inicijalizirana dijagonalna matrica [36, 38, 37]. Korištenje dijagonalne matrice ključno je za efikasnost konvolucijske formulacije. Naime, zbog opetovanog množenja matrice $\mathbf{A}$ u izrazu konvolucijske jezgre (4.8), za općenite matrice stanja $\mathbf{A}$ izračun konvolucijske jezgre $\mathbf{K}$ ima složenost $O(d_h^2 L)$, dok je za dijagonalne matrice složenost samo $O(d_h + L)$.

Potreba za posebnom metodom inicijalizacije matrica stanja $\mathbf{A}$ javlja se kao rješenje problema nestabilnog učenja ovakvih modela. Pokazano je da odgovarajuće inicijalizacije matrice stanja $\mathbf{A}$ rješavaju problem nestajućih gradijenata te omogućuju modelu lakše učenje problema s dugoročnim ovisnostima [34].

Najčešća metoda inicijalizacije matrice stanja $\mathbf{A}$ je inicijalizacija metodom projekcije polinoma višeg stupnja (engl. *High-Order Polynomial Projection Operator*,*HiPPO*). Pojednostavljeno, matrica se inicijalizira tako da omogući sažimanje ulaznog slijeda u skriveno stanje koje ima dovoljno informacije za približnu rekonstrukciju cijele povijesti. Više o inicijalizaciji s metodom zvanom *HiPPO* čitatelj može pronaći u pratećem radu [38].

5. Selektivni modeli u prostoru stanja

Tri formulacije prethodno opisanih strukturiranih modela u prostoru stanja (povratna, konvolucijska i vremenski kontinuirana) dijele svojstvo linearne vremenske nepromjenjivosti (engl. *Linear Time Invariance*).

Linearna vremenska nepromjenjivost očituje se u parametrima modela koji su stalni kroz sve vremenske korake t. Ovo svojstvo ključno je za učinkovitost modela u prostoru stanja. Bez njega, efikasno učenje putem konvolucijske formulacije ne bi bilo moguće. Naime, konvolucijska jezgra bi u tom slučaju bila različita za svaki vremenski korak t te bi se morala ponovno računati u svakom vremenskom koraku. Dakle, učinkovitost proizlazi iz stalne konvolucijske jezgre $\mathbf{K}$.

Iako ograničenje stalnih parametara strukturiranih modela u prostoru stanja dovodi do njihove računalne učinkovitosti, ono također znatno smanjuje njihovu ekspresivnost i mogućnost uporabe u dubokim neuronskim mrežama. Kod ovih modela, informacije iz elemenata ulaznog slijeda uvijek se na jednak način propagiraju u sljedeće skriveno stanje. Također, informacije iz skrivenog stanja uvijek se na jednak način propagiraju u sljedeći element izlaznog slijeda. Ova uniformnost u propagaciji informacija onemogućava modelima da zaključuju na temelju sadržaja elemenata slijeda (engl. *Content-based Reasoning*).

Povećanje ekspresivnosti ovih modela moguće je dodavanjem tzv. mehanizma selekcije (engl. *Selection Mechanism*). Dodavanjem mehanizma selekcije u formulaciju strukturiranih modela u prostoru stanja dobivamo selektivne modele u prostoru stanja. Selektivni modeli u prostoru stanja predstavljeni su u sklopu originalne Mambe te će biti predstavljeni po uzoru na originalni rad [33]. Kao što je mehanizam pozornosti ključ uspjeha Transformatora, tako su i selektivni modeli u prostoru stanja ključ uspjeha arhitekture Mamba.

5.1. Mehanizam selekcije

Autori Mambe motiviraju mehanizam selekcije putem kompromisa između učinkovitosti i efektivnosti modela za obradu sljedova [33]. Teorijska uspješnost mehanizma pozornosti leži u činjenici da ne dolazi do sažimanja ulaznog slijeda. Pri izgradnji konteksta, svaki element slijeda koristi informacije iz drugih elemenata slijeda bez sažimanja. Međutim, bez sažimanja, mehanizam pozornosti pati od izuzetne neučinkovitosti zbog kvadratne vremenske i prostorne složenosti. Paralelno učenje nudi mogućnost rješavanja ovog nedostatka. S druge strane, povratni modeli su vrlo učinkoviti u smislu vremenske i prostorne složenosti jer iz ulaznog slijeda grade kontekst fiksne veličine. Međutim, isto to sažimanje ulaznog slijeda uzrok je njihove manje efektivnosti. Da bi povratni modeli bili uspješniji, trebali bi imati veće skriveno stanje kako bi mogli sačuvati veću količinu informacija iz ulaznog slijeda. Dakle, iako linearne računalne složenosti, nemogućnost paralelne obrade slijeda uparena s manjom efektivnosti razlog je zašto se smatraju inferiornijima Transformatorima.

Autori zaključuju da je za očuvanje učinkovitosti neophodno imati kontekst fiksnih dimenzija, ali i mogućnost paralelne obrade elemenata slijeda, no ističu potrebu za poboljšanjem kvalitete generiranog konteksta radi povećanja efektivnosti modela [33]. Autori vide uvođenje mehanizma selekcije u formulaciju strukturiranih modela u prostoru stanja ključnim za poboljšanje kvalitete generiranog konteksta.

Mehanizam selekcije je mehanizam koji omogućava selektivnu propagaciju informacija iz elemenata ulaznog slijeda u skriveno stanje strukturiranog modela u prostoru stanja, kao i selektivnu propagaciju informacija iz skrivenog stanja u elemente izlaza. U selektivnim modelima u prostoru stanja, parametri $\mathbf{B}, \mathbf{C}, \Delta$ su vremenski promjenjivi, dok matrica stanja $\mathbf{A}$ ostaje vremenski nepromjenjiva.

Mehanizam selekcije očituje se u dodavanju dodatnog koraka prije provođenja diskretizacijskog pravila:

$$\begin{aligned} \mathbf{B} &= s_B(\mathbf{x}) \\ \mathbf{C} &= s_C(\mathbf{x}) \\ \Delta &= softplus(s_\Delta(\mathbf{x})) \end{aligned} \tag{5.1}$$

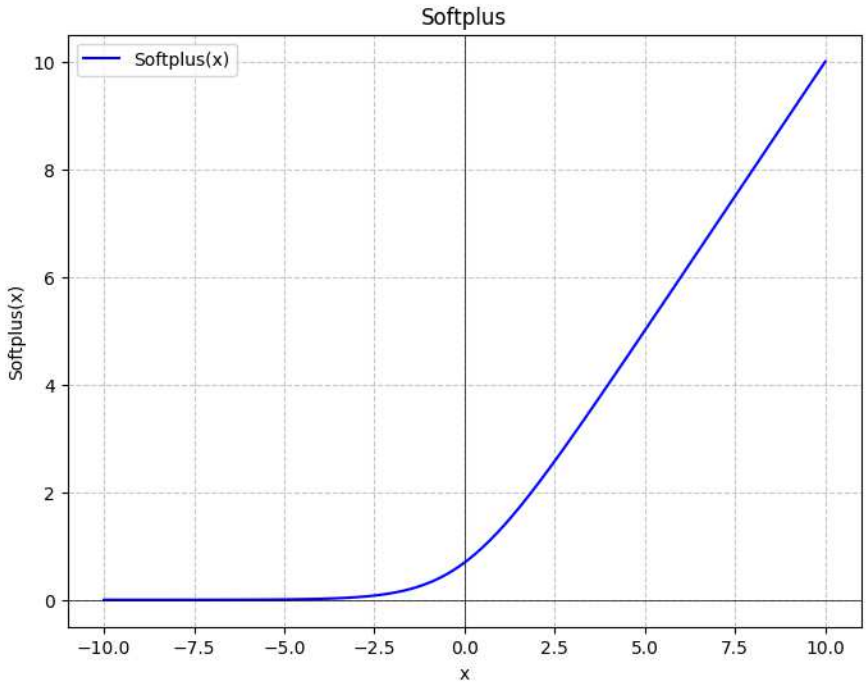
Pri čemu su s_B, s_C parametrizirane linearne funkcije s ciljem projekcije ulaza u d_h-dimenzionalni latentni prostor. Funkcija s_Δ također je parametrizirana linearna funkcija čija je zadaća projicirati ulaz u jednodimenzionalni prostor koraka diskretizacije.[1]

[1] Autori u radu navode kako stopa učenja ne mora nužno biti skalarna vrijednost te poopćavaju ovu projekciju na r-dimenzionalan prostor.

Autori koriste $softplus$ aktivacijsku funkciju definiranu kao:

$$softplus(x) = \log(1 + \exp(x)) \tag{5.2}$$

Graf aktivacijske funkcije $softplus$ prikazan je u 5.1.



Slika 5.1: Graf aktivacijske funkcije $softplus$.

Funkcija s_Δ ponavlja se za svaku dimenziju ulaza zasebno, rezultirajući s d koraka diskretizacije, odnosno d-dimenzionalnim vektorom Δ.

Označimo li nove parametre kao $\mathbf{B}_x, \mathbf{C}_x, \Delta_x$ i njihove pripadne diskretizirane vrijednosti s $\overline{\mathbf{B}}_x, \overline{\mathbf{C}}_x$, izraz za selektivni diskretni SSM postaje:

$$\begin{aligned} h^{(t)} &= \overline{A}h^{(t-1)} + \overline{\mathbf{B}}_x\mathbf{x}^{(t)} \\ y^{(t)} &= \overline{\mathbf{C}}_x h^{(t)} \end{aligned} \tag{5.3}$$

Parametar Δ_x regulira koliko pozornosti model treba staviti na trenutni ulaz $\mathbf{x}^{(t)}$. Ukratko, što je Δ_x veći, to model više pažnje posvećuje trenutnom ulazu, dok s vrlo malim vrijednostima Δ_x model zanemaruje trenutni ulaz. Intuitivno se to može shvatiti osvrtom na postupak diskretizacije. Pri pretvorbi diskretnog signala u kontinuirani, Δ predstavlja vremenski interval tijekom kojeg zadržavamo uzorak prije nego što stigne novi. S $\Delta \to 0$, trenutni uzorak zanemarujemo i prelazimo na sljedeći uzorak.

Parametar $\overline{\mathbf{B}}_x$ regulira količinu informacija iz trenutnog elementa ulaza koja se propušta u kontekst, odnosno skriveno stanje. S druge strane, parametar $\overline{\mathbf{C}}_x$ kontrolira

koliko informacija iz skrivenog stanja se koristi za izlaz, ovisno o trenutnom elementu.

Zanimljivo je napomenuti da se poveznica između modela SSM-a i povratnih modela nije izgubila dodatkom mehanizma selekcije. Autori originalnog rada ističu kako je mehanizam selekcije upravo poseban oblik mehanizma propusnica.

Iako mehanizam selekcije omogućava generiranje kvalitetnijeg konteksta, dopuštanjem parametrima $\overline{\mathrm{B}}, \overline{\mathrm{C}}, \Delta$ da variraju kroz vrijeme, selektivni modeli u prostoru stanja gube mogućnost konvolucijskog načina rada. Gubitak mogućnosti konvolucijskog načina rada proizlazi iz činjenice da je konvolucijska jezgra drugačija u svakom vremenskom koraku t. Konvolucijski način rada omogućio je modelima u prostoru stanja efikasno paralelno učenje i učinio ih privlačnijima u odnosu na klasične povratne modele. Kako bismo zadržali efikasnost učenja bez konvolucijskog načina rada, ključno je implementirati mehanizam selekcije na način koji će osigurati jednaku efikasnost kao i kod konvolucijskog načina rada. Tako će selektivni modeli u prostoru stanja zadržati efikasnost učenja i poboljšati efektivnost generiranjem kvalitetnijeg konteksta.

5.2. Učinkovita implementacija

Kako bi osigurali učinkovito izvođenje na računalima uz zadržavanje povećane ekspresivnosti koju donosi mehanizam selekcije, autori originalnog rada predlažu korištenje triju tehnika.

S pomoću paralelnog algoritma za izračun kumulativne sume, autori ipak uspijevaju učinkovito implementirati paralelnu obradu slijeda u selektivnim SSM-ovima. Dodatna ubrzanja autori postižu iskorištavanjem specifičnosti izvođenja operacija na GPU-u putem tehnike poznate kao stapanje jezgri. Konačno, za dodatno poboljšanje memorijskih svojstava implementacije, autori koriste tehniku ponovnog izračuna. Navedene tehnike objašnjenje su u ostatku poglavlja.

5.2.1. Paralelna kumulativna suma

Za postizanje veće računalne učinkovitosti, ključno je imati algoritam koji je moguće paralelno izvršavati. Takav paralelni algoritam omogućava optimalno korištenje grafičkih procesorskih jedinica, koje su specijalizirane za istodobnu obradu velikog broja operacija.

Prema [1], velik broj paralelnih algoritama zasniva se na operaciji kumulativne ili prefiksne sume (engl. *Cumulative Sums, All-prefix-sums Operation*).

Neka je $\oplus$ binarni operator koji zadovoljava svojstvo asocijativnosti, odnosno neka za bilo koja tri elementa a, b, c vrijedi:

$$(a \oplus b) \oplus c = a \oplus (b \oplus c). \tag{5.4}$$

Također, definira se I kao element identiteta za binarni asocijativni operator $\oplus$ takav da za bilo koji a vrijedi:

$$a \oplus I = I \oplus a = a \tag{5.5}$$

Uz tako definiran binarni asocijativni operator i zadani slijed $(z^{(1)}, \dots z^{(n)})$ kumulativna suma dana je s:

$$(I, z^{(1)}, (z^{(1)} \oplus z^{(2)}), (z^{(1)} \oplus z^{(2)} \oplus z^{(3)}), \dots (z^{(1)} \oplus z^{(2)} \dots z^{(n)})) \tag{5.6}$$

Slijedna implementacija prefiksne sume zahtijevala bi prolazak po svim elementima ulaznog slijeda. U svakom koraku i, uzima se trenutni element $z^{(i)}$ te rezultat iz prethodnog koraka i nad njima proveli operaciju $\oplus$ za dobivanje novog rezultata. Dakle, za ulazni slijed od n elemenata, potrebno je izvršiti $O(n)$ operacija.

Bitno je primijetiti kako je, za dani ulazni slijed elemenata, slijed skrivenih stanja selektivnog SSM-a upravo primjer prefiksne sume.

Prema Bllelochu [21], prefiksna suma dobar je primjer operacije za koju se čini da zahtijeva slijedno izvršavanje, ali za koju svejedno postoji efikasni paralelni algoritam. Jedan od takvih efikasnih paralelnih algoritama dao je upravo Blleloch te ga nazivamo Bllelochovim paralelnim *skenom* [21].

Bllelochov paralelni *sken*

Objašnjenje Blellochovog paralelnog *skena* dano je prema [1].

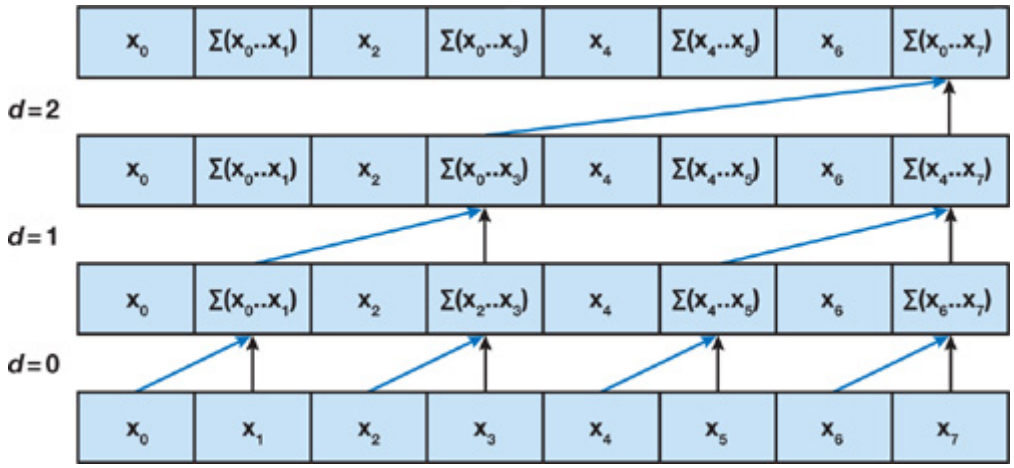
Bllelochov paralelni *sken* zasniva se na ideji izgradnje balansiranog binarnog stabla nad ulaznim slijedom te obilaženjem istog tog stabla za potrebe računanje prefiksne sume. Za ulazni slijed veličine n, izgrađeno binarno stablo ima $d = \log_2 n$ razina te svaka razina ima 2^d čvorova.

Stablo koje se gradi nije stvarna podatkovna struktura, već koncept koji se koristi kako bi se odredilo što pojedina dretva radi u svakom koraku obilaska.

Algoritam se provodi u dvije faze. Prva faza naziva se redukcijom (engl. *Reduction*) ili *Up-sweep*, dok se druga faza naziva *Down-sweep*.

Tijekom redukcije, stablo se obilazi kretanjem od listova prema korijenu usputno računajući parcijalne kumulativne sume unutarnjih čvorova stabla. Na kraju ove faze,

korijenski čvor sadrži sumu svih čvorova ulaznog slijeda. Slikovni prikaz redukcije dan je na slici 5.2, a pseudokod u Algoritmu 1.



Slika 5.2: Prva faza Bllelochov paralelnog skena (redukcija). U ovoj fazi cilj je izgraditi parcijalne vrijednosti kumulativnih suma u unutarnjim čvorovima binarnog stabla. Slika preuzeta s [1].

Algoritam 1 Prva faza Bllelochovog paralelnog skena (redukcija)

1: **za** $d = 0$ do $\lfloor \log_2(n-1) \rfloor$:
2: **za sve** $k = 0$ do $n - 1$ paralelno radi:
3: $\mathbf{x}[k + 2^{d+1} - 1] = \mathbf{x}[k + 2^d - 1] + \mathbf{x}[k + 2^d + 1 - 1]$

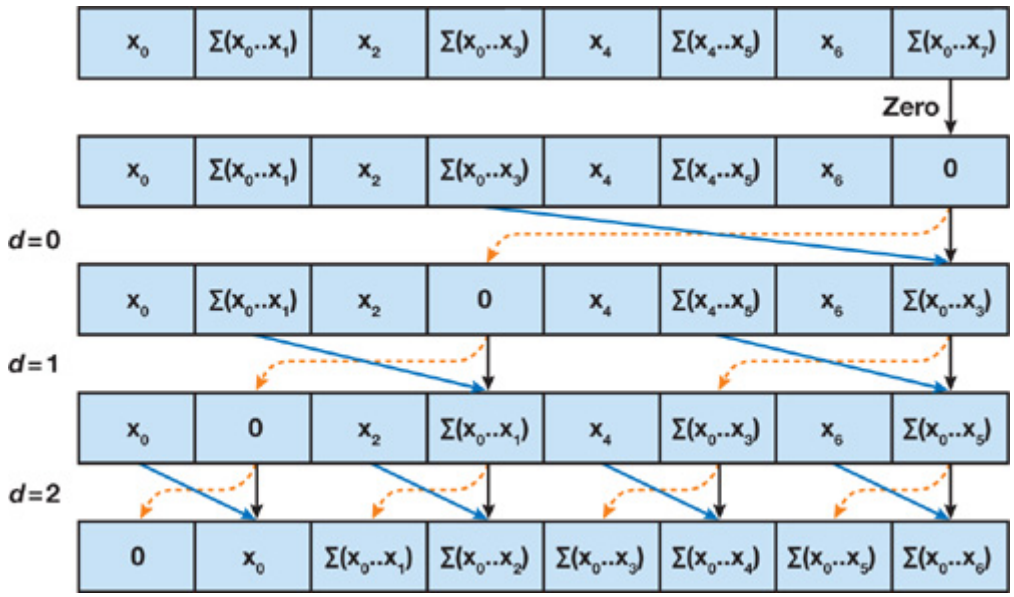
U pseudokodu 1, vanjska petlja prolazi kroz razine stabla. Počinje s razinom $d = 0$, koja odgovara listovima stabla, i nastavlja se do $\log_2(n-1)$, gdje je n broj elemenata u ulaznom nizu. Na svakoj razini d, unutarnja petlja računa sumu dva susjedna elementa na toj razini. Sume na istoj razini računaju se u paraleli. Drugim riječima, u ovoj fazi algoritam pažljivo dijeli elemente polja i računa parcijalne prefiksne sume dok ne dobijemo sumu svih elemenata polja, što odgovara završnom elementu kumulativne sume.

U fazi zvanoj *down-sweep*, ponovno se obilazi stablo, ali ovaj put kretanjem od korijena prema listovima. Pri tom obilasku koristimo parcijalne prefiksne sume unutarnjih čvorova stabla dobivenih u prethodnoj fazi kako bismo dobili konačnu kumulativnu sumu. Kretanjem po stablu, u svakom čvoru provodimo dva koraka:

1. Rezultat operacije $\oplus$ nad vrijednosti čvora i vrijednosti njegovog lijevog djeteta prenosimo u desno;

2. Vrijednost čvora prenosimo u njegovo lijevo dijete.

Obilaskom stabla i primjenom pravila ažuriranja vrijednosti čvorova, na kraju druge faze dobit ćemo kumulativnu sumu definiranu kao u (5.6). Slikovni prikaz faze zvane *down-sweep* dan je na slici 5.3, a pseudokod u Algoritmu 2.



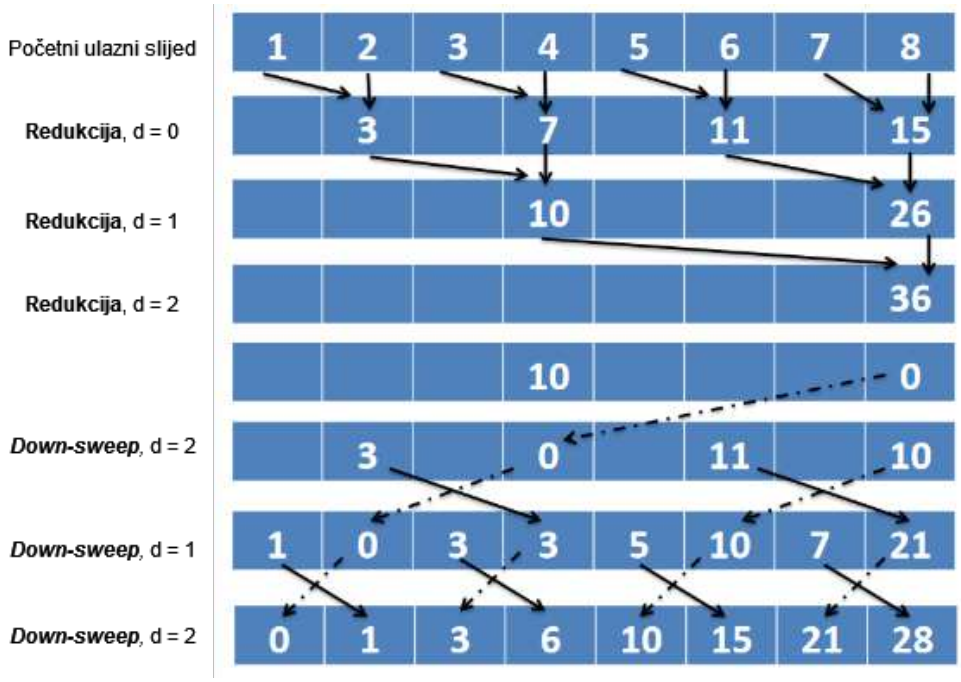
Slika 5.3: Druga faza Bllelochov paralelnog *skena* (*down-sweep*). U ovoj fazi cilj je na temelju rezultata prethodne faze izgraditi konačnu kumulativnu sumu. Konačna suma gradi se kretanjem po stablu od korijena prema listovima, prateći pravila ažuriranja vrijednosti čvorova danih u 2. Slika preuzeta s [1].

Algoritam 2 Druga faza Bllelochovog paralelnog *skena* (*down-sweep*)

$\mathbf{x}[n-1] \leftarrow 0$

1: **za** $d = \lfloor \log_2(n-1) \rfloor$ do 0:
2: **za sve** $k = 0$ do $n-1$ paralelno radi:
3: $t = \mathbf{x}[k + 2^d - 1]$
4: $\mathbf{x}[k + 2^d - 1] = \mathbf{x}[k + 2^d + 1 - 1]$
5: $\mathbf{x}[k + 2^d + 1 - 1] = t \oplus \mathbf{x}[k + 2^d + 1 - 1]$

Objasnimo Bllelochov paralelni sken na primjeru ulaznog slijeda duljine $n = 8$ te neka je taj slijed dan s $(1, 2, 3, 4, 5, 6, 7, 8)$. Neka je binaran asocijativan operator definiran kao operator zbrajanja. Konačna prefiksna suma tada je dana s $(0, 1, 3, 6, 10, 15, 21, 28)$ te je cijeli algoritam prikazan na slici 5.4.



Slika 5.4: Prikaz algoritma Bllelochovog paralelnog skena na primjeru ulaznog slijeda $(1, 2, 3, 4, 5, 6, 7, 8)$ za binaran asocijativan operator zbrajanja. Konačni rezultat algoritma iznosi $(0, 1, 3, 6, 10, 15, 21, 28)$

Budući da je $n = 8$, u prvoj fazi Bllelochovog paralelnog skena, vanjska suma redukcije polazi od $d = 0$ do $d = \lfloor log_2(7) \rfloor$, odnosno do $d = 2$.

U prvom koraku redukcije vrijedi $d = 0$. Unutarnja petlja počinje s $k = 0$ i ide do vrijednosti $k = 7$. U svakoj iteraciji unutarnje petlje, računaju se sume elemenata slijeda udaljenih za $2^d = 2^0$ pozicija, odnosno elemenata udaljenih za jednu poziciju. Rezultat prvog koraka redukcije tada je $(1, \mathbf{3}, 3, \mathbf{7}, 5, \mathbf{11}, 7, \mathbf{15})$.

U drugom koraku redukcije, $d = 1$, unutarnja petlja redukcije računa parcijalne sume susjednih elemenata na prvoj razini stabla, što odgovara računanju sume elemenata slijeda iz prethodnog koraka koji su udaljeni za $2^d = 2^1$ pozicija, odnosno elemenata s razmakom od dvije pozicije. Rezultat drugog koraka redukcije bit će $(1, 3, 3, \mathbf{10}, 5, 11, 7, \mathbf{26})$.

Konačni rezultat dobiva se u trećem koraku redukcije, $d = 2$, te glasi $(1, 3, 3, 10, 5, 11, 7, \mathbf{36})$. Konačni rezultat redukcije dobiven je računanjem sume elemenata iz prethodnog koraka koji su udaljeni za $2^d = 2^2 = 4$ pozicije.

Drugu fazu Bllelochovog paralelnog skena započinje postavljanjem zadnjeg elementa slijeda dobivenog u prethodnoj fazi na 0 što rezultira slijedom $(1, 3, 3, 10, 5, 11, 7, \mathbf{0})$. Nakon toga, kreće se u vanjsku petlju druge faze koja kreće s $d = \lfloor log_2(7) \rfloor$, tj. $d = 2$ te završava s $d = 0$.

U prvom koraku, $d = 2$, unutarnja petlja zamjenjuje vrijednost na poziciji $k + 2^2 - 1 = k + 3$ sa svojim desnim djetetom, tj. $k + 2^2 + 1 - 1 = k + 4$. Potom ažurira vrijednost na poziciji $k + 4$ sa zbrojem stare vrijednosti na $k + 3$ i trenutne vrijednosti na $k + 4$. Rezultat prvog koraka tada je $(1, 3, 3, \mathbf{0}, 5, 11, 7, \mathbf{10})$. Navedeno ažuriranje ponavlja se i preostale koraka. Konačni rezultat obiju faza Bllelochovog paralelnog skena dan je s $(0, 1, 3, 6, 10, 15, 21, 28)$.

Primjena Bllelochov *skena* za implementaciju selektivnog SSM-a

Kako bi bilo moguće primjeniti Bllelochov paralelni *sken* na jednadžbe selektivnog SSM-a dane u (5.3), potrebno je definirati binaran asocijativni operator $\oplus$ i pripadni element identiteta I.

Definirajmo $z^{(t)} = (a^{(t)}, b^{(t)})$ te neka su:

$$\begin{aligned} a^{(t)} &= \overline{\mathbf{A}} \\ b^{(t)} &= \overline{\mathbf{B}}_x \mathbf{x}^{(t)} \end{aligned} \tag{5.7}$$

Tada definiramo operator $\oplus$ na sljedeći način:

$$z^{(i)} \oplus z^{(j)} = [a^{(j)} a^{(i)}, a^{(j)} b^{(i)} + b^{(j)}] \tag{5.8}$$

Element identiteta pripadnog operatora I odgovora početnom stanju te ga definiramo kao $z^{(0)} = [\overline{\mathbf{A}}, \mathbf{h}^{(0)}]$.

Primjena Bllelochovog paralelnog algoritma nad ovako definiranim ulaznim slijedom $(z^1, \ldots, z^{(n)})$ rezultirat će traženim skupom skrivenih stanja $\mathbf{h}^{(t)}$. Na ovaj način efikasno možemo obraditi cijeli ulazni slijed paralelno, umjesto slijedno.

5.2.2. Stapanje jezgara

Vremenska i memorijska složenost drastično se smanjuje dodatnim iskorištavanjem memorijske hijerarhije GPU tehnikom stapanja jezgara (engl. *Kernel Fusion*) [33]. Tehnika *Kernel Fusion* podrazumijeva stapanje više operacija koje dijele iste ulaze u jednu operaciju.

Slijed operacija selektivnog modela u prostoru stanja s pogleda grafičke procesorske jedinice podrazumijeva velik broj prijenosa iz jedne vrste memorije u drugu vrstu memorije. Prvo, matrice $\mathbf{A}, \mathbf{B}, \mathbf{C}, \Delta$ se prenose iz HBM u bržu memoriju SRAM. Zatim se te matrice diskretiziraju unutar SRAM-a. Nakon toga, diskretizirane matrice $\overline{\mathbf{A}}$ i $\overline{\mathbf{B}}$ te $\overline{\mathbf{C}}$ se prenose natrag u HBM. Nakon toga slijedi ponovni prijenos matrica $\overline{\mathbf{A}}$ i $\overline{\mathbf{B}}$ iz HBM-a u SRAM. Zatim se one koriste u operaciji paralelnog skeniranja kako

bi se dobio skriveni sloj dimenzija koji se zatim prenosi iz SRAM-a natrag u HBM. Nakon toga, skriveni sloj i matrica $\overline{\mathbf{C}}$ prenose se iz HBM-a u SRAM. Konačno, izvodi se množenje skrivenog sloja s matricom izlaza $\overline{\mathbf{C}}$ kako bi se dobio konačni izlaz $\mathbf{y}$, koji se zatim prenosi iz SRAM-a u HBM.

Moguće je primijetiti da se ovdje radi o operaciji ograničenoj memorijom. Iako imamo paralelan algoritam koji iskorištava učinkovitost paralelne obrade na GPU-u, kako bismo smanjili vrijeme izvršavanja operacije, potrebno je smanjiti broj čitanja i pisanja u memoriju.

Da bi se to postiglo, koraci diskretizacije, paralelnog skeniranja i dobivanja konačnog izlaznog signala spajaju se u jednu operaciju. Prvo, matrice $\mathbf{A}, \mathbf{B}, \mathbf{C}$ i Δ prenose se iz HBM-a u SRAM. Zatim se te matrice diskretiziraju unutar SRAM-a, a paralelno skeniranje provodi se unutar iste memorije. Nakon toga, rezultati paralelnog skeniranja množe se s matricom $\overline{\mathbf{C}}$ kako bi se dobio konačni izlaz $\mathbf{y}$, koji ostaje u SRAM-u. Konačno, konačni rezultat $\mathbf{y}$ prenosi se iz SRAM-a natrag u HBM.

Ovaj pristup smanjuje ukupan broja prijenosa podataka iz HBM-a u SRAM i obratno, čime se poboljšava učinkovitost samog algoritma.

5.2.3. Ponovni izračun

Za smanjenje prostorne složenosti, autori predlažu tehniku zvanu *Recomputation* [33]. To znači da pri prolasku unaprijed međurezultati dimenzija (L, d, d_h) se ne pohranjuju u memoriju.[2] Kad su međurezultati potrebni, oni se ponovno izračunavaju.

[2]Točnije, riječ je međurezultatima veličine (B, L, d, d_h) budući da se radi nad grupama veličine B.

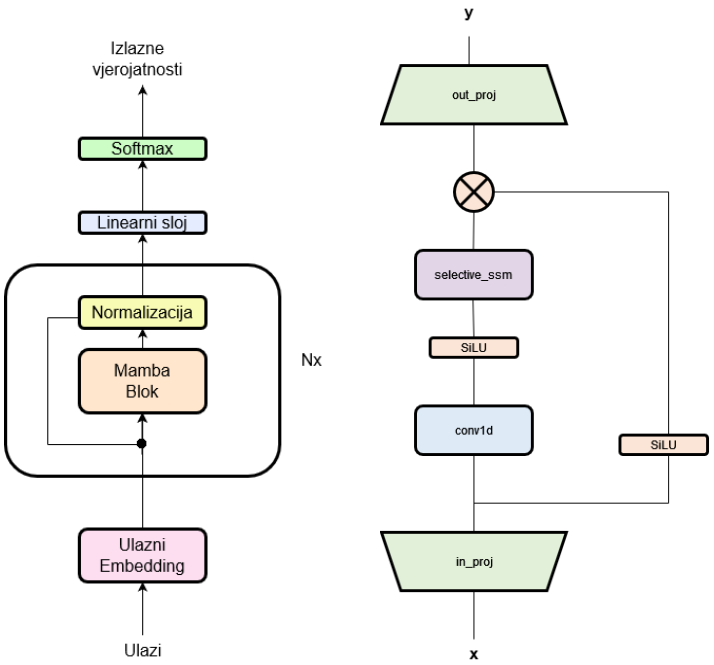
6. Mamba

Prethodno opisani selektivni modeli u prostoru stanja predstavljaju linearne transformacije sljedova te se kao takvi mogu koristiti kao zasebni slojevi unutar dubokih neuronskih mreža. Parametri tih slojeva uče se standardnim algoritmom propagacije pogreške unatrag, isto kao i parametri ostalih slojeva mreže.

6.1. Opis arhitekture

U ovom radu, neuronska mreža zasnovana na arhitekturi Mambe implementirana je u programskom jeziku Python koristeći programski okvir PyTorch [51].

Originalna arhitektura temelji se na ponavljajućem bloku Mambe, koji je povezan s povratnim vezama i standardnim normalizacijskim slojevima, kao što je prikazano na slici 6.1a. Blok Mambe integrira više operacija, prikazanih na slici 6.1b. Unaprijedni prolaz kroz blok Mambe dan je programskim isječkom 1.



(a) arhitektura Mambe **(b)** blok Mambe

Slika 6.1: Prikaz arhitekture Mambe i pripadnog bloka Mambe. Arhitektura Mambe sačinjena je od ponavljajućeg bloka omotanog povratnim vezama te normalizacijskim slojem (6.1a). Blok Mambe sastoji se od operacija linearne projekcije, konvolucije te rezidualne veze (6.1b).

Programski isječak 1 Unaprijedni prolaz bloka Mambe

```
1    def forward (self, \vectorbold{x}):
2
3        xz = self.in_proj (x)
4        x, z = xz.split (split_size=[self.config.d_expanded,
         ↪  self.config.d_expanded], dim=-1)
5
6        z = nn.SiLU (z)
7
8        x = rearrange (x, "b l d_ex -> b d_ex l")
9        x = self.conv1d (x)[:, :, :l]
10       x = rearrange (x, "b d_ex l -> b l d_ex")
11
12       x = nn.SiLU (x)
13       y = self.selective_ssm (x)
14
15       y = y * z
16       y = self.out_proj (y)
17
18       return y
```

Neka je B broj primjera u maloj grupi, L duljina ulaznog niza te d dimenzionalnost elemenata slijeda. Ulazi u blok $\mathbf{x}$ tada su dimenzija $(B \times L \times d)$. Ulazi se najprije projiciraju u prostor veće dimenzionalnosti te se potom razdvajaju u dva ogranka svaki konačne dimenzije `d_expanded`, što se vidi u retcima 3 – 5 programskog isječka 1. Projekcija u prostor veće dimenzionalnosti definirana je u programskom isječku 2.

Programski isječak 2 Projekcija ulaza u prostor veće dimenzije

```
1    self.in_proj = nn.Linear (
2        in_features=d,
3        out_features=2 * d_expanded,
4        bias=config.bias,
5    )
```

Zatim, ogranak projiciranog ulaza, `z`, prolazi kroz aktivacijsku funkciju $silu$, vidljivo u retku 6 programskog isječka 1.[1] Nad ogrankom projiciranog ulaza `x` provodi se operacija konvolucije kroz vremensku dimenziju. Konvolucija se obavlja nad svakim ulaznim kanalom (dimenzijom) zasebno, što se također naziva dubinskom konvolucijom (engl. *Depthwise Convolution*). Uz definirane hiperparametre kao što su veličina

[1] Aktivacijska funkcija $silu$ definira se kao $silu(\mathbf{x}) = \mathbf{x} \cdot sigmoid(\mathbf{x})$.

konvolucijske jezgre `d_conv` te pomak konvolucije `conv_bias`, konvolucijski sloj definiran je na način vidljiv u isječku 3. Postavljanjem hiperparametra `groups` na vrijednost `d_expanded` u retku 6 isječka 3, provodimo konvoluciju nad svakom dimenzijom zasebno. Ovaj sloj omogućava modelu da uči važne lokalne uzorke i značajke unutar vremenskog prozora svake dimenzije.

Programski isječak 3 Sloj za konvoluciju ulaznih elemenata slijeda

```
1     self.conv1d = nn.Conv1d (
2             in_channels=d_expanded,
3             out_channels=d_expanded,
4             kernel_size=d_conv,
5             bias=conv_bias,
6             groups=d_expanded,
7             padding=d_conv - 1,
8         )
```

Konačan rezultat konvolucije kroz vremensku dimenziju nad ogrankom `x`, matrica je iste dimenzije $(B \times L \times \texttt{d\_expanded})$. Rezultat konvolucije zatim prolazi kroz aktivacijsku funkciju $silu$, a potom kroz efikasnu implementaciju selektivnog SSM-a, vidljivo u retcima 12 – 13 programskog isječka 1. Konačno, rezultat selektivnog SSM-a spaja se s početnim ogrankom `z` pomoću Hadamardovog umnoška, odnosno umnoška elemenata po pozicijama. Na kraju, rezultat se projicira natrag u dimenziju ulaza, čime se dobiva konačni rezultat, dimenzija $(B \times L \times d)$. Konačni izlazni projekcijski sloj vidljiv je u programskom isječku 4.

Programski isječak 4 Izlazni projekcijski sloj

```
1     self.out_proj = nn.Linear (in_features=d_expanded,
      ↪  out_features=d)
```

6.2. Implementacijski detalji

Budući da PyTorch, kao i mnogi drugi okviri za duboko učenje, ne pruža potpunu slobodu programiranja uzimajući u obzir svojstva GPU-ova, korišten je Triton [55], programski jezik sličan Pythonu, za pisanje učinkovitijeg GPU koda. Triton se jednostavno integrira u postojeći Python kôd upotrebom posebnih dekoratora `@triton.jit` iznad funkcija napisanih u Pythonu. U ovom radu, mehanizam selekcije je implementiran pomoću Tritona, dok je ostatak koda napisan u PyTorchu. Ovaj pristup omogućuje

da se specifične računalno intenzivne funkcije izvršavaju na GPU-ovima putem Tritona, dok se PyTorch koristi za definiranje neuronskih mreža, upravljanje podacima te treniranje modela. Sav kôd u nastavku napisan je u PyTorchu. Razliku između funkcija napisanih u PyTorchu i Tritonu čitatelj može pronaći u dodatku A.

Implementacija se temelji na originalnoj implementaciji Mambe, s time da su su autori izravno optimizirali rad GPU-ova koristeći programski jezik C++ [11], koji je memorijski učinkovitiji od programskog jezika Python korištenog u svrhu implementacije ovog rada. Iako je originalni rad isključio parametar $\mathbf{D}$ iz formulacije SSM-a, u originalnoj implementaciji je parametar zadržan te je ostavljen i u implementaciji za potrebe ovog rada. Dodatno, matrica stanja $\mathbf{A}$ inicijalizirana je korištenjem inicijalizacijske metode zvane *HiPPO*, po uzoru na originalnu implementaciju. Uz te napomene, unaprijedni prolaz kroz selektivni SSM dan je u programskom isječku 5.

Programski isječak 5 Implementacija prolaza unaprijed za selektivni SSM

```
    def selective_ssm (self, x):
        A = self.A.float ()
        D = self.D.float ()

        deltaBC = self.x_proj (x)

        delta, B, C = deltaBC.split (
            split_size=[self.config.r, self.config.d_h,
            ↪  self.config.d_h],
            dim=-1,
        )

        delta = F.softplus (self.dt_proj (delta))


        y = self.ssm (x, delta, A, B, C, D)

        return y
```

Mehanizam selekcije, odnosno dobivanje parametara $\mathbf{B}_x, \mathbf{C}_x, \Delta_x$ na temelju ulaza, vidljiv je u šestoj liniji isječka 5. Radi optimizacije, funkcije s_B, s_C, s_Δ objedinjene su u jedan linearni projekcijski sloj čija je definicija prikazana u isječku 6. Združeni rezultat na kraju se dijeli na odgovarajuće veličine kako bi dobili konačne matrice parametara $\mathbf{B}, \mathbf{C}, \Delta$ redom veličina $(B \times L \times \texttt{d\_h})$, $(B \times L \times \texttt{d\_h})$, $(B \times L \times r)$.

Programski isječak 6 Sloj za dobivanje kontekstno ovisnih parametara SSM-a, B, C, Δ

```
1     self.x_proj = nn.Linear (
2             in_features=d_expanded,
3             out_features=r + d_h * 2,
4             bias=False,
5         )
```

.

Međutim, kao što je definirano u izrazu za dobivanje kontekstno ovisnog koraka diskretizacije, parametar Δ potrebno je dobiti za svaku dimenziju ulaza. Iz tog razloga, Δ prolazi kroz dodatni projekcijski sloj čija je uloga projekcija Δ u `d_expanded`-dimenzionalni prostor te konačno kroz aktivacijsku funkciju $softplus$. To je vidljivo u liniji 13 isječka 5, a dodatni projekcijski sloj prikazan je u programskom isječku 7.

Programski isječak 7 Projekcijski sloj parametra Δ

```
1     self.dt_proj = nn.Linear (in_features=dt_rank,
      ↪  out_features=D_inner)
```

.

Funkcija ssm u liniji 16 programskog isječka 5 sadrži ključne korake rada modela SSM-a. Prvo, funkcija obavlja korak diskretizacije parametara $\mathbf{A}, \mathbf{B}$, a zatim operaciju prefiksne sume za dobivanje skrivenih stanja $\mathbf{h}$. Na temelju skrivenih stanja $\mathbf{h}$ te parametara $\mathbf{C}, \mathbf{D}$ dobivaju se konačni izlazi $\mathbf{y}$. Ova funkcija implementirana je na više načina za potrebe usporedbe izvedbi.

Prva implementacija je naivna slijedna implementacija operacije prefiksne sume. Kao što je prethodno objašnjeno, ovakva implementacija nije pogodna za izvođenje na GPU-ovima.

Druga implementacija temelji se na prethodno opisanom Blellochovom paralelnom algoritmu za dobivanje kumulativne sume te se pripisuje sljedećem izvoru [5].

Treća implementacija temelji se na nedavnom radu [39] koji nudi alternativnu formulaciju Blellochove prefiksne sume, pogodnu za učinkovito izvršavanje u modernim programskim okvirima poput PyTorcha.[2] Implementacija je napravljena po uzoru na

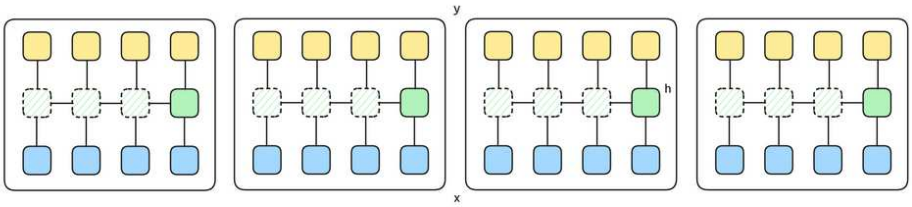
[2]Ukratko, implementacija se temelji na raspisivanju izraza kumulativne sume pomoću izraza za koje se zna da postoji njihova efikasna implementacija u PyTorch-u.

implementaciju autora rada [39]. Na ovu implementaciju će se u nastavku referirati nazivom *Heinsenov slijed*.

Četvrta implementacija napravljena je uz pomoć programskog jezika Triton. U njoj je obuhvaćen postupak stapanja jezgara, efikasnog paralelnog izračuna skrivenih stanja te izbjegavanje materijalizacije međurezultata kako bi se dodatno poboljšala vremenska i prostorna složenost. Triton nudi jednostavan način pisanja jezgara u obliku Python funkcija. Umjesto s varijablama, radi se s pokazivačima te izravno učitavamo ulaze i parametre iz memorije, izvršavamo operacije, te konačni rezultat pišemo na odgovarajuće mjesto u memoriju.

Unutar jezgre prvo se provodi diskretizacija parametara $\mathbf{A}, \mathbf{B}$ koji se potom koriste za računanje kumulativne sume s odgovarajućim asocijativnim operatorom kako je definirano u potpoglavlju 5.2.1.

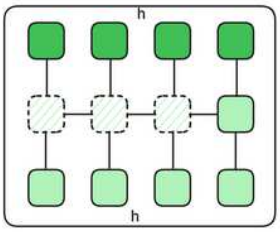
Kako bi implementacija radila i na vrlo dugim rečenicama koje se ne mogu učitati na GPU, rečenice se dodatno dijele u blokove koji se paralelno izvode. Nad svakim blokom rečenice izračunat će se kumulativna suma. Triton nudi učinkovitu implementaciju operacije kumulativne sume [14]. Dobivanje kumulativnih suma pojedinih blokova vidljivo je na slici 6.2.



Slika 6.2: Podjela ulaznog slijeda na blokove fiksne veličine te izračun slijeda skrivenih stanja za svaki od blokova zasebno. Slika preuzeta s [4].

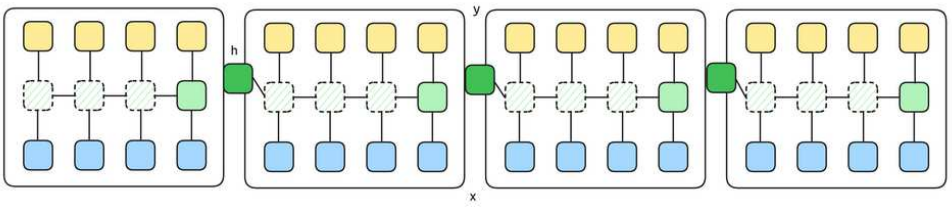
Jednom izračunate kumulativne sume pojedinih blokova, koristit ćemo za dobivanje konačne kumulativne sume u dva dodatna koraka.

U prvom koraku, nad završnim elementima kumulativnih suma pojedinih blokova provest ćemo operaciju kumulativne sume. To je vidljivo na slici 6.3.



Slika 6.3: Izračun kumulativne sume nad završnim skrivenim stanjima svakog od blokova početnog ulaznog slijeda. Dobivena kumulativna suma koristit će se za inicijalizaciju početnih skrivenih stanja pri dobivanju konačnih skrivenih stanja. Slika preuzeta s [4]

Zatim, u drugom koraku, svakom bloku bit će dano početno skriveno stanje. Početno skriveno stanje odgovarat će elementu kumulativne sume dobivenom u prvom koraku. Drugi korak vidljiv je na slici 6.4.



Slika 6.4: Izračun konačnih skrivenih stanja ulaznog slijeda podijeljenog na blokove. Slika preuzeta s [4]

Kao rezultat navedenih operacija, konačan rezultat su skrivena stanja $\mathbf{h}$. Ta skrivena stanja potom množimo parametrom $\mathbf{C}$ te im pridodajemo umnožak $\mathbf{Dx}$ kako bi dobili konačan rezultat.

Diskretizacija parametara, u svim implementacijama, napravljena je po uzoru na originalnu implementaciju u kojoj su se autori odlučili za pojednostavljenje metode zvane ZOH. U tom slučaju, diskretizacijsko pravilo dano je s:

$$\begin{aligned} \overline{\mathbf{A}} &= \exp(\Delta \cdot \mathbf{A}) \\ \overline{\mathbf{B}} &= \mathbf{B} \cdot \Delta \end{aligned} \tag{6.1}$$

Međusobna usporedba implementacija dana je u poglavlju s eksperimentima.

7. Eksperimenti

Nakon implementacije arhitekture Mamba, provedeno je nekoliko eksperimenata koji su detaljno opisani u narednim potpoglavljima.

Prvo, u potpoglavlju 7.1, dana je usporedba vremenske i prostorne složenosti mehanizma pozornosti, osnovne gradivne jedinice popularnih Transformatora, s različitim implementacijama mehanizma selekcije, gradivne jedinice Mambe.

Zatim, u poglavlju 7.2, prikazana je usporedba izvedbe Mambe i Transformera nad skupom sintetičkih zadataka koji ocrtavaju osnovne sposobnosti velikih jezičnih modela. Za svaki zadatak proveden je statistički test jednakosti srednjih vrijednosti kako bi se utvrdilo je li razlika u uspješnosti modela statistički značajna.[1]

Konačno, u potpoglavlju 7.3, dana je evaluacija računalne izvedbe Mambe nad podskupom popularnog ispitnog skupa GLUE, budući da autori originalnog rada nisu proveli takvu evaluaciju. Kao i za sintetičke zadatke, tako je i u zadatcima ispitnog skupa GLUE, model Mambe uspoređen s modelom Transformatora te je za potrebe ispitivanja statističke značajnosti rezultata korišten statistički test jednakosti srednjih vrijednosti.

7.1. Analiza vremenske i prostorne složenosti

Uspoređene su različite implementacije mehanizma selekcije, od naivne slijedne implementacije do varijanti paralelnog *skena*. Za potrebe analize, korištena je računalna arhitektura s Intel Core i5-9300H središnjim procesorom s 8 jezgri i brzinom takta 2.40 GHz, podržanog s 16 GB DDR4 RAM-a i NVIDIA GeForce GTX 1650 grafičkom karticom.

Od implementacija čije su izvedbe paralelne, uspoređeni su Bllelochov paralelni *sken*, *Heinsenov slijed* te paralelni *sken* implementiran u Tritonu, kraće naznačen kao *sken u Tritonu*. Osim navedenih, dana je usporedba i s visoko optimiziranim originalnim mehanizmom selekcije, naznačeno kao Originalni mehanizam selekcije.

[1] Korišten je Wilcoxonov test prednačenih rangova.

Prvo, usporedba vremena izvođenja paralelnih implementacije i slijedne implementacije nad nizovima duljine 256 dana je tablicom 7.1.[2]

Implementacija	μ **(ms)**	σ **(ms)**
Slijedna	51.7	2.9
Bllelochov *sken*	2.71	0.105
Heinsenov slijed	1.47	0.106
sken u Tritonu	1.23	0.444
Originalni mehanizam selekcije	0.123	0.125

Tablica 7.1: U tablici je moguće vidjeti srednju vrijednost μ i standardnu devijaciju σ vremena potrebnog za unaprijedni prolaz različitih implementacija mehanizma selekcije nad sljedovima duljine 256 u 1000 ponavljanja.

Iz tablice 7.1, vidljiva je prednost paralelne implementacije u vidu poboljšanja računalne izvedbe. Zanimljivo je primijetiti kako je implementacija uz pomoć *Heinsenovog slijeda* postigla najbolje vrijeme na nizovima duljine 256. Mogući razlog je njezina posebna prilagođenost programskom okviru PyTorch. Naime, budući da je napisana uz pomoć operacija koje su temelj Pytorcha, njezina efikasnost nije začuđujuća.

Različite implementacije mehanizma selekcije uspoređene su i po svojoj potrošnji memorije nad nizovima duljine 128, što je dano tablicom 7.2. Memorijska potrošnja izmjerena je uz pomoć Pytorcheve klase zvane *profiler*.

Implementacija	**CPU memorija (Kb)**	**GPU memorija (Kb)**
Slijedna	0.00	16310.00
Bllelochov *sken*	4.00	250.00
Heinsenov slijed	24.12	306.00
sken u Tritonu	12.12	52.50
Originalni mehanizam selekcije	16.12	45.50

Tablica 7.2: Memorija potrebna za unaprijedni prolaz različitih implementacija mehanizma selekcije nad sljedovima duljine 128.

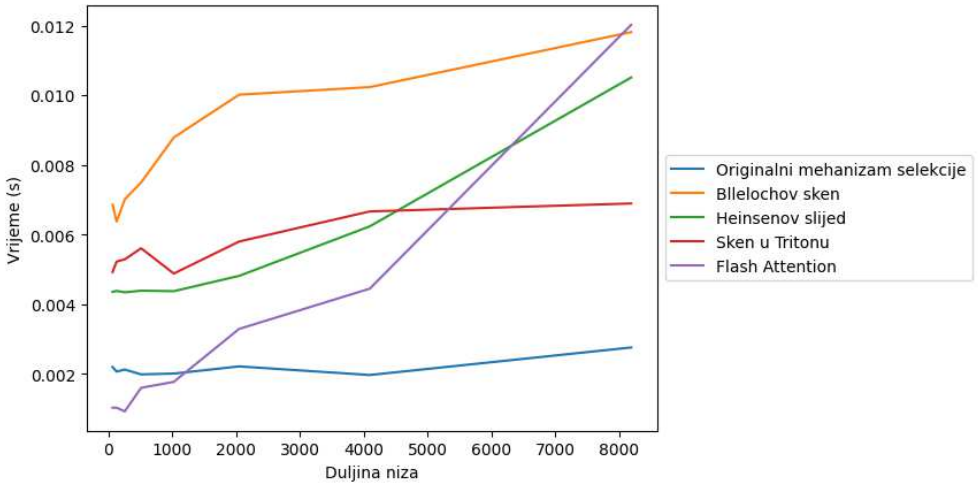
Iz tablice 7.2, do izražaja ponovno dolazi visoka optimiziranost originalne implementacije koja je najbolja i po pitanju memorijske potrošnje. Međutim, ovdje do izražaja dolazi i snaga tehnika stapanja jezgara i ponovnog izračuna međurezultata. Naime, iz drastične razlike koja postoji između implementacije scana u Tritonu i drugih

[2]Riječ je o nizovima slučajno inicijaliziranih 4-dimenzionalnih vektora.

implementacija vidljiv je problem operacija ograničenih memorijom. Budući da mehanizam selekcije zahtijeva vrlo mnogo prijenosa iz HBM-a u SRAM, bez tehnike stapanja jezgara, ostale implementacije troše drastično puno memorije. Također, izostanak spremanja međurezultata, karakterističan za implementaciju u Tritonu te slijednu implementaciju, znatno doprinosi boljoj iskorištenosti memorije.

Osim međusobno, po uzoru na originalni rad [33], učinkovitije paralelne implementacije uspoređene su i s linearnim mehanizmom pozornosti. Konkretno, usporedba je dana s trenutno najbržom varijantom linearnog mehanizma pozornosti, tzv. *FlashAttention* [25]. Mehanizam *FlashAttention* postiže linearnu računalnu složenost minimiziranjem broja prijenosa iz jednog tipa memorije u drugi. Za detalje postizanja linearne složenosti čitatelja se uputuje na rad [25].

Mehanizmi su uspoređeni na temelju mogućnosti skaliranja s obzirom na duljinu ulaznog niza. Uzete su sve duljine niza potencije broja 2 od 2^6 do 2^{14} te je izmjereno vrijeme potrebno za obradu ulaznog niza. Eksperiment je ponavljan 1000 puta za svaku duljinu niza te je zabilježeno prosječno vrijeme izvođenja. Usporedba tako izmjerenih prosječnih vremena potrebnih za obradu nizova različitih duljina dana je grafom na slici 7.1.



Slika 7.1: Usporedba srednjih vremena potrebnih za obradu nizova različitih duljina između različitih implementacija mehanizme selekcije te linearnog mehanizma pozornosti.

Iz grafa 7.1, može se zamijetiti više toga. Prvo, originalna implementacija mehanizma selekcije i implementacija scana u Tritonu pokazuju da dobro skaliraju s obzirom na duljinu niza. Povećanjem duljine niza, vrijeme potrebno za obradu putem ovih mehanizama ne raste značajno. S druge strane, ostale implementacije skaliraju lošije s obzirom na duljinu ulaznog niza, pri čemu *FlashAttention* ima najbrži rast. Među-

tim, bitno je primijetiti da niti jedna od ovih implementacija ne pokazuje kvadratno skaliranje s obzirom na duljinu ulaznog niza, što je napredak u odnosu na originalni mehanizam pozornosti.

7.2. Sintetički zadatci

Autori originalnog rada predlažu dva sintetička zadatka za koja smatraju da najbolje ispituju ključna svojstva temeljnih modela [33]. Na temelju tih zadataka dana je usporedba modela Mambe s Transformatorom približno jednake veličine. Oba modela imaju svega 50 tisuća parametara. Model Mambe ima 2 sloja sa dimenzionalnošću modela 64 i dimenzionalnošću skrivenog stanja 16. Model Transformatora također ima 2 sloja sa dimezionalnošću modela 64, dimenzionalnošću potpuno povezanog sloja 128 te 8 glava višestruke pozornosti.

Kao ključna svojstva modela koji bi trebali biti osnova velikih jezičnih modela, autori navode sposobnosti pamćenja modela (engl. *memorization abilities*) te sposobnost učenja unutar konteksta (engl. *in-context learning abilities*). Dobrim pokazateljem tih ključnih sposobnosti predlažu rezultate nad dva sintetička zadataka.

Prvi zadatak naziva se se zadatkom selektivnog kopiranja. On je inačica zadatka kopiranja u kojem se zadatak modela svodi na preslikavanje ulaznog slijeda na svoj izlaz [16]. Prema [16], u originalnom zadatku kopiranja, ulaz u model sastoji se od $T + 20$ elemenata pri čemu vrijedi da su elementi ulaza oznake iz skupa $\{a_0, ..., a_9\}$.

Ulazni slijed od $T + 20$ elemenata organiziran je na sljedeći način. Prvih 10 elemenata ulaza predstavljaju uzorak koji model mora zapamtiti pri čemu su elementi uzorka dobiveni slučajnim uzorkovanjem iz skupa $\{a_0, ..., a_7\}$. Nakon uzorka koji model mora zapamtiti, sljedećih $T - 1$ elemenata čine praznine ili tzv. prazni element (*blank element*), koji je uvijek a_8. Nakon $T - 1$ praznina, model dobiva signal za kraj, označen kao a_9. Nakon signala za kraj, zadnjih 10 elemenata ulaza ponovno su popunjeni prazninama. Željeni izlaz modela sastoji se od $T + 10$ praznih elemenata kojih slijede elementi uzorka u istom redoslijedu pojavljivanja kao u ulazu. Autori originalnog zadatka kopiranja eksperimentiraju s vrijednostima
$T \in \{100, 200, 300, 500\}$.

Jedan primjer ulaza modela $\mathbf{x}$ te očekivanog izlaza y dan je s:

$$\begin{aligned} \mathbf{x} &= [\mathbf{a_2}, \mathbf{a_7}, \ldots, \mathbf{a_1}, a_8, a_8, \ldots, a_8, a_9, a_8, \ldots, a_8] \\ \mathbf{y} &= [a_8, a_8, a_8, \ldots, a_8, a_8, a_8, a_8, \mathbf{a_2}, \mathbf{a_7}, \ldots, \mathbf{a_1}] \end{aligned} \tag{7.1}$$

Pri čemu je uzorak za pamćenje dan s $\mathbf{a_2}, \mathbf{a_7}, \ldots, \mathbf{a_1}$.

Na tako opisan način, generiran je skup podatka za učenje od 10000 ulaznih sljedova te skup podataka za ispitivanje od 3000 ulaznih slijedova. Nad tako generiranim skupovima podataka, trenirani su i testirani jednostavni Mamba model te jednostavni Transformator model.

Metrika korištena u originalnom radu je točnost (engl. *Accuracy*), definirana kao točan broj izlaznih elemenata modela podijeljen s ukupnim brojem izlaznih elemenata.

Rezultati su dani u tablici 7.3.

Model	**Točnost**	σ
Mamba	0.900	$8.02 \cdot 10^{-4}$
Transformator	0.900	$3.10 \cdot 10^{-4}$

Tablica 7.3: Usporedba Mambe i Transformatora na jednostavnom zadatku kopiranja

Iz rezultata 7.3, vidi se da niti jedan model nema problema u učenju ovog zadatka te razlika u njihovim rezultatima nije statistički značajna.

Iako originalni zadatak kopiranja testira sposobnost pamćenja, on ne zahtijeva nikakvo zaključivanje na temelju konteksta. Naime, struktura ulaza je uvijek ista. Prvih 10 elemenata je uvijek uzorak kojeg slijede praznine i signal kraja koje model ne mora zapamtiti. Pokazano je da ovakav zadatak mogu naučiti i modeli s parametrima koji ne ovise o ulazu, npr. obični strukturirani modeli u prostoru stanja.

Iz tog razloga, autori eksperimentiraju s težom inačicom ovog zadatka u kojem položaji praznina variraju. Taj zadatak naziva se selektivnim zadatkom kopiranja [33]. U njemu, model se ne može oslanjati na uvijek istu strukturu ulaza, nego mora naučiti filtrirati nebitne informacije, odnosno praznine. Iz toga je očito da ovakav zadatak nije rješiv uz pomoć običnih modela s linearno vremenski nepromjenjivim parametrima.

Zadatak autori definiraju na sljedeći način. Definira se V kao veličina vokabulara podatkovnih elemenata. Unutar vokabulara, definira se jedan element kao prazni element ili praznina. Ulazni uzorak ukupne je veličine L, dok je uzorak za pamćenje veličine N. U ulaznom uzorku slučajno se odabire $L - N$ pozicija na kojima će biti praznine, dok se ostalih N elemenata popuni sa slučajno odabranim elementima vokabulara. Očekivani izlaz sastoji se od L elemenata gdje prvih $L - N$ čine praznine. Nakon praznina, zadnjih N elementa čini uzorak za pamćenje, dan u istom redoslijedu kao i u ulaznom nizu. Na primjer, za veličinu vokabulara $V = 10$ pri čemu je praznina dana kao 9, veličinu uzorka za pamćenje $N = 3$ te duljinu ulaznog niza $L = 10$,

mogući ulaz $\mathbf{x}$ te očekivani izlaz y bili bi dani s:

$$\begin{aligned} \mathbf{x} &= [9, 9, \mathbf{7}, 9, 9, 9, 9, 9, \mathbf{7}, \mathbf{1}] \\ \mathbf{y} &= [9, 9, 9, 9, 9, 9, 9, \mathbf{7}, \mathbf{7}, \mathbf{1}] \end{aligned} \tag{7.2}$$

Ovaj zadatak je puno bolji pokazatelj sposobnosti pamćenja modela. Model mora naučiti selektivno propustiti informaciju ovisno o trenutnom elementu slijeda.

Slično kao i u običnom zadatku kopiranja, tako su za zadatak selektivnog kopiranja generirani pripadni skupovi podataka za učenje i treniranje jednake veličine kao u zadatku kopiranja te je provedeno učenje i evaluacija Mambe i Transformatora. Ponovno, po uzoru na originalni rad, korištena je metrika točnosti. Rezultati usporedbe dani su tablicom 7.4.

Model	**Točnost**	σ
Mamba	0.916	$8.12 \cdot 10^{-4}$
Transformator	0.875	$2.80 \cdot 10^{-5}$

Tablica 7.4: Usporedba Mambe i Transformatora na selektivnom zadatku kopiranja

Iz rezultata je vidljivo da model Mambe uspješnije uči zadatak selektivnog kopiranja te je razlika između modela statistički značajna na temelju statističkog testa jednakosti srednjih vrijednosti, $p < 0.05$.

Sljedeći bitni zadatak za ispitivanje sposobnosti modela je zadatak indukcije (engl. *Induction Heads Task*) [49]. Postoji hipoteza da taj zadatak objašnjava većinu sposobnosti današnjih velikih jezičnih modela da uče iz konteksta [49]. Primjer je to asocijativnog pamćenja koji zahtijeva pronalaženje odgovora iz konteksta, ključnu sposobnost za velike jezične modele.

Neka je V veličina vokabulara te neka je definiran poseban element separatora. Unutar ulaznog slijeda duljine L slučajno se odabire pozicija separatora. Osim na tu poziciju, separator se stavlja i na kraj niza. Ostalih $L-2$ pozicija popunjeno je slučajno odabranim elementima vokabulara. Očekivani izlaz modela je element koji se nalazi neposredno nakon prvog znaka separatora. Tako za duljinu ulaznog niza $L = 10$, veličinu vokabulara $V = 16$, recimo da je separator jednak 17. Tada primjer mogućeg ulaza $\mathbf{x}$ te očekivanog izlaza mogli bi biti:

$$\begin{aligned} \mathbf{x} &= [3, 0, 9, 0, 10, 3, \mathbf{17}, \mathbf{1}, 14, \mathbf{17}] \\ y &= \mathbf{1} \end{aligned} \tag{7.3}$$

Generiran je ponovno skup od 10000 sljedova za učenje te 3000 sljedova za ispitivanje. Po uzoru na autore originalnog rada, za veličinu vokabulara uzeto je $V = 30$

te su modeli učeni na nizovima duljina 256. Modeli su ponovno evaluirani koristeći točnost kao metriku.

Usporedba Mambe i Transformatora dana je u tablici 7.5.

Model	**Točnost**	σ
Mamba	0.070	0.017
Transformator	0.067	0.015

Tablica 7.5: Usporedba Mambe i Transformatora na zadatku indukcije

Ovaj zadatak pokazao se izrazito težak za oba modela te rezultati dani tablicom 7.5 pokazuju kako nijedan model nije bio u stanju naučiti ga. Potencijalan razlog je nedovoljna veličina modela. Budući da je riječ o modelima sa svega 50 tisuća parametara, moguće je da im je kapacitet nedovoljan za ovaj zadatak. Razlika u modelima nije statistički značajna.

7.3. Ispitni skup GLUE

Ispitni skup GLUE (engl. *General Language Understanding Evaluation*) važan je alat u području obrade prirodnog jezika koji služi za evaluaciju i usporedbu izvedbe različitih modela [57]. Ovaj ispitni skup sastoji se od devet zadataka koji obuhvaćaju širok spektar jezičnih razumijevanja, kao što su klasifikacija teksta, prevođenje, razumijevanje parafraziranja i slično. Razvijen je kako bi pružio standardiziranu metriku za ocjenu jezičnih modela, olakšavajući tako usporedbu između različitih pristupa i modela.

Svaki zadatak ima pridružen skup podataka za učenje, ispitivanje te, opcionalno, skup za provjeru. Budući da skup podataka za ispitivanje nije javno dostupan, u svrhu testiranja bit će korišten skup za provjeru. Za treniranje će biti korišten dostupni skup za treniranje. Također, zbog ograničenih računalnih resursa, evaluacija je napravljena na podskupu ispitnog skupa GLUE koji uključuje šest od originalnih devet zadataka.

Za različite zadatke koriste se različite metrike. Svaki zadatak unutar ispitnog skupa GLUE propisuje svoj skup metrika. Osim spomenute točnosti, u nastavku se koriste još i metrike poput Matthewsov koeficijent korelacije, Pearsonovog koeficijenta korelacije, mjere F1 te Spearmanovog korelacijskog ranga.

Za potrebe ovog eksperimenta, uzet je predtrenirani model Mambe s 130 milijuna parametara te je fino prilagođen na svakom od narednih zadataka [10]. Predtrenirani model ima 24 sloja te je dimenzionalnost modela 768. Model je predtreniran od strane

autora originalnog rada koristeći skup podataka Pile [28, 20]. Fino prilagođeni model Mambe uspoređen je s fino prilagođenim modelom Transformatora slične veličine. Za potrebe ove usporedbe, uzet je predtrenirani model BERT-a (engl. *Bidirectional Encoder Representations from Transformers*), varijacija arhitekture Transformatora, s 110 milijuna parametara, 12 slojeva, dimenzionalnosti modela 768 te 12 glava višestruke pozornosti. Predtrenirani model Model BERT-a fino je prilagođen za svaki od izabranih zadataka ispitnog skupa GLUE, na isti način kao i model Mambe.

Pojašnjenja zadataka kao i rezultati na njima, dani su u potpoglavljima što slijede.

CoLA

CoLA (engl. *Corpus of Linguistic Acceptability*) zadatak je procjene gramatičke točnosti rečenica [2].

Riječ je o zadatku binarne klasifikacije. Svakoj rečenici pridružena je oznaka klase ovisno o njezinoj ispravnosti. Gramatički ispravnim rečenicama pridružena je oznaka klase 1, dok je neispravnim rečenicama pridružena oznaka klase 0. Primjeri rečenica s pripadnim oznakama klase dani su tablicom 7.6.

Rečenica	**Oznaka klase**
They made him angry	1
They caused him to become angry by making him	0
We're dancing the night away.	1
You get angrier, the more we eat, don't we.	0

Tablica 7.6: Primjeri rečenica s pripadnim oznakama klase u CoLA skupu podataka. Oznaka klase 0 označava gramatički neprihvatljive rečenice, dok klasa 1 predstavlja gramatički ispravne rečenice.

Skup podataka sastoji se od 7696 rečenica u skupu za učenje te 855 rečenica u skupu za ispitivanje. Za potrebe vrednovanja rezultata, koristi se Matthewsov koeficijent korelacije. Rezultati su dani tablicom 7.7. Osim same točnosti, u tablici 7.7, dana je i standardna devijacija točnosti. Također, proveden je statistički test jednakosti srednjih vrijednosti, koji je pokazao da razlika između modela Mamba i BERT nije statistički značajna.

Model	Matthewsov koeficijent korelacije	σ
Mamba	0.498	0.142
BERT	0.432	0.139

Tablica 7.7: Usporedba točnosti i standardne devijacije točnosti σ modela Mambe i BERT-a na zadatku CoLa-e. Razlika modela nije statistički značajna.

MRPC

MRPC (engl. *Microsoft Research Paraphrase Corpus*) zadatak je u kojemu je potrebno odrediti jesu li dvije rečenice parafraze [3].

Riječ je o zadatku binarne klasifikacije. Svakom paru rečenica pridružena je oznaka klase iz skupa $\{0, 1\}$. Ako par rečenica predstavlja rečenice koje su parafraze, onda im je dodijeljena oznaka klase 1, u suprotnom 0. Primjeri parova rečenica s pripadnim oznakama klase dani su tablicom 7.8.

Rečenice	Oznaka klase
On July 3, Troy is expected to be sentenced to life in prison without parole. Troy faces life in prison without parole at his July 30 sentencing.	1
And they think the protein probably is involved in the spread of other forms of cancer. They researchers say the research could be relevant to other forms of cancer.	0

Tablica 7.8: Primjeri parova rečenica s pripadnim oznakama klase u MRPC skupu podataka. Oznaka klase 0 označava da rečenice nisu parafraze, dok klasa 1 predstavlja par parafraziranih rečenica.

Skup podataka sastoji se od 3301 parova rečenica u skupu za treniranje te 367 parova rečenica u skupu za ispitivanje. U ovom zadataku za vrednovanje koriste se točnost i F1 mjera. Rezultati usporedbe modela Mambe i modela BERT-a dani su tablicom 7.9. Svakoj metrici pridružena je i standardna devijacija. Provedeni statistički test jednakosti srednjih vrijednosti pokazao je da razlika između modela nije statistički značajna.

Model	Točnost	F1	σ (točnost)	σ (F1)
Mamba	0.799	0.864	0.062	0.038
BERT	0.803	0.859	0.039	0.029

Tablica 7.9: Usporedba modela Mambe i BERT-a na zadatku MRPC-a. Razlika modela nije statistički značajna.

RTE

RTE (engl.*Recognizing Textual Entailment*) zadatak je koji se fokusira na prepoznavanje implikacija između rečenica [9].

Skup podataka sastoji se od parova rečenica. Zadatak je ovo binarne klasifikacije u kojem svaki par rečenica nosi oznaku klase 0 ako je druga rečenica implikacija prvoj te 1 suprotno. Primjeri parova rečenica i pripadnih klasa dani su tablicom 7.10.

Rečenice	Oznaka klase
The Dutch, who ruled Indonesia until 1949, called the city of Jakarta Batavia. Formerly (until 1949) Batavia, Jakarta is largest city and capital of Indonesia.	1
Oil prices fall back as Yukos oil threat lifted. Oil prices rise.	0

Tablica 7.10: Primjeri parova rečenica s pripadnim oznakama klase u skupu podataka RTE. Oznaka klase 0 označava da je druga rečenica implikacija prve, dok oznaka klase 1 označava suprotno.

Skup za treniranje sastoji se od 2241 parova rečenica, dok se skup za ispitivanje sastoji od svega 249 parova rečenica. Za vrednovanje se koristi točnost. Rezultati su dani tablicom 7.11. Prema rezultatima, vidljivo je da Mamba ostvaruje bolje rezultate od modela BERT-a te je ta razlika statistički značajna na temelju statističkog testa jednakosti srednjih vrijednosti, $(p < 0.05)$.

Model	Točnost	σ
Mamba	0.631	0.069
BERT	0.526	0.104

Tablica 7.11: Usporedba Mambe i BERT-a na RTE. Razlika u rezultatima modela je statistički značajna.

SST-2

SST-2 (engl. *Stanford Sentiment Treebank*) zadatak je klasifikacije sentimenta rečenica [61].

Skup podataka sastoji se od rečenica i pripadnih oznaka klasa. Budući da je riječ o zadatku binarne klasifikacije, oznake klase su 0 i 1. Pri čemu, ako je rečenica pozitivnog sentimenta, oznaka klase bit će 1. Ako je rečenica negativnog sentimenta, oznaka klase bit će 0. Primjeri rečenica s pripadnim oznakama sentimenta dani su tablicom 7.12.

Rečenica	**Oznaka klase**
gorgeous and deceptively minimalist	1
excruciatingly unfunny and pitifully unromantic	0
equals the original and in some ways even betters it	1
a depressed fifteen-year-old 's suicidal poetry	0

Tablica 7.12: Primjeri rečenica s pripadnim oznakama klase u SST-2 skupu podataka. Oznaka klase 0 označava rečenicu negativnog sentimenta, dok klasa 1 sadrži rečenice pozitivnog sentimenta.

Skup podataka za treniranje sadrži 60614 rečenica, dok skup za ispitivanje sadrži njih 6735. U ovom zadataku za vrednovanje koristi se točnost. Rezultati su dani tablicom 7.13. Prema rezultatima, vidljivo je da Mamba ostvaruje bolje rezultate od modela BERT-a te je ta razlika statistički značajna na temelju statističkog testa jednakosti srednjih vrijednosti, $(p < 0.05)$.

Model	**Točnost**	σ
Mamba	0.920	0.026
BERT	0.804	0.039

Tablica 7.13: Usporedba Mambe i BERT-a na SST-2. Razlika u rezultatima modela je statistički značajna.

STS-B

STS-B (engl. *Semantic Textual Similarity*) zadatak koji se bavi procjenom sličnosti između parova rečenica [7].

Skup podataka sastoji se od parova rečenica. Svakom paru pridružena je vrijednost od 0 do 5 u ovisnosti o njihovoj sličnosti. Što su rečenice bliže u značenju, to je njihova

vrijednost bliža 5. Suprotno, što su manje semantički slične, to je vrijednost manja i bliža 0. Primjeri parova rečenica s pripadnim ocjenama sličnosti dani su tablicom 7.14.

Rečenice	**Oznaka klase**
A plane is taking off.	
An air plane is taking off.	5
A woman is dancing.	
A man is talking.	0
A man is cutting up a potato.	
A man is cutting up carrots.	2.375

Tablica 7.14: Primjeri parova rečenica s pripadnim ocjenama semantičke sličnosti u skupu podataka STS-B. Sličnije rečenice imaju ocijenu bližu 5, dok one manje slične imaju manju ocijenu. Potpuno iste rečenice imat će ocijenu 5, dok će one potpuno različite imati 0.

Skup podataka sastoji se 5174 parova rečenica u skupu za treniranje te njih 575 u skupu za ispitivanje. U ovom zadataku za vrednovanje koriste se Pearsonov korelacijski koeficijent te Spearmanov korelacijski rang. Prema rezultatima, vidljivo je da u ovom zadatku model BERT-a ostvaruje bolje rezultate od modela Mambe te je ta razlika statistički značajna na temelju statističkog testa jednakosti srednjih vrijednosti, $(p < 0.05)$.

Model	**Pearsonov koeficijent**	**Spearmanov koeficijent**	σ **(Pearson)**	σ **(Spearman)**
Mamba	0.811	0.799	0.090	0.095
BERT	0.855	0.849	0.056	0.062

Tablica 7.15: Usporedba Mambe i BERT-a na STS-B. Razlika u modelima je statistički značajna.

WNLI

WNLI (engl. *Winograd NLI*) zadatak je prepoznavanja implikacija, sličan zadatku RTE [6].

Skup podataka sastoji se od parova rečenica. Zadatak je ovo binarne klasifikacije u kojem svaki par rečenica nosi oznaku klase 1 ako je druga rečenica implikacija prvoj te 0 suprotno. Primjere parova rečenica i pripadnih oznaka klase može se vidjeti u tablici 7.16.

Rečenice	Oznaka klase
I stuck a pin through a carrot. When I pulled the pin out, it had a hole.	
The carrot had a hole.	1
The table won't fit through the doorway because it is too narrow.	
The table is too narrow.	0

Tablica 7.16: Primjeri parova rečenica s pripadnim oznakama klase u WNLI skupu podataka. Oznaka klase 1 označava da je druga rečenica implikacija prve, dok oznaka klase 0 označava suprotno.

Skup podataka sastoji se 571 parova rečenica u skupu za treniranje te njih 64 u skupu za ispitivanje. Za vrednovanje se koristi točnost. Rezulati usporedbe modela Mambe i BERT-a dani su tablicom 7.17. Prema rezultatima danima u tablici 7.17, vidljivo je da model BERT-a ostvaruje veću točnost od modela Mambe. Međutim, ta razlika nije statistički značajna. Moguće objašnjenje lošijih rezultata oba modela je da je riječ o teškom zadatku s vrlo malo primjera u skupu podataka za učenje. Da je objašnjenje loših rezultata mala količina podataka, govori i činjenica da je na RTE-u Mamba uspijela ostvariti značajno bolje rezulate.

Model	Točnost	σ
Mamba	0.434	0.217
BERT	0.566	0.185

Tablica 7.17: Usporedba Mambe i BERT-a na WNLI. Razlika u rezultatima modela nije statistički značajna.

8. Zaključak

U posljednje vrijeme, sve veći resursi ulažu se u učenje velikih jezičnih modela. Uz obilje dostupnih podataka, glavni ograničavajući faktor postaje računalna snaga, a posebno zbog visoke računalne složenosti Transformatora, dominantne arhitekture suvremenih jezičnih modela. Trud istraživačke zajednice dugo je bio usmjeren na pokušaje poboljšanja izvedbe Transformatora, stavljajući ostale arhitekture u drugi plan.

U ovom radu cilj je bio prezentirati arhitekturu Mambe zajedno sa strukturiranim modelima u prostoru stanja kao njezinim temeljnim komponentama. Iskorištavanje detalja izvedbe modela na grafičkim procesorima omogućila je arhitekturi Mambe ostvarivanje boljih računalnih izvedbi u odnosu na popularne Transformatore. Prezentirane su optimizacijske tehnike čija je implementacija omogućila ostvarivanje boljih računalnih izvedbi u odnosu na Transformatore. Uspoređena je vremenska i prostorna složenost Mambe i Transformatora, te su provedene evaluacije na stvarnim i sintetičkim skupovima podataka. Rezultati pokazuju da model Mambe ostvaruje zadovoljavajuće izvedbe s iznimno povoljnom vremenskom i prostornom složenošću.

Unatoč impresivnim rezultatima prikazanim u originalnom radu, potrebne su dodatne evaluacije arhitekture Mambe u brojnim drugim zadatcima prije nego što ju se proglasi zamjenicom dominantnih Transformatora. Čak i ako ne zamijeni Transformatore, arhitektura Mambe predstavlja korak naprijed za istraživačku zajednicu, potičući istraživanje i razmatranje novih arhitektura.

LITERATURA

[1] Chapter 39. Parallel Prefix Sum (Scan) with CUDA, . URL https://developer.nvidia.com/gpugems/gpugems3/part-vi-gpu-computing/chapter-39-parallel-prefix-sum-scan-cuda.

[2] The Corpus of Linguistic Acceptability (CoLA), . URL https://nyu-mll.github.io/CoLA/.

[3] Download Microsoft Research Paraphrase Corpus from Official Microsoft Download Center, . URL https://www.microsoft.com/en-us/download/details.aspx?id=52398.

[4] Mamba: The Hard Way, . URL https://srush.github.io/annotated-mamba/hard.html.

[5] mamba.py/pscan.py at main · alxndrTL/mamba.py, . URL https://github.com/alxndrTL/mamba.py/blob/main/pscan.py.

[6] Papers with Code - WNLI Dataset, . URL https://paperswithcode.com/dataset/wnli.

[7] Papers with Code - STS Benchmark Dataset, . URL https://paperswithcode.com/dataset/sts-benchmark.

[8] PyTorch, . URL https://pytorch.org/.

[9] Recognizing Textual Entailment - ACL Wiki, . URL https://aclweb.org/aclwiki/Recognizing_Textual_Entailment.

[10] state-spaces/mamba-130m · Hugging Face, . URL https://huggingface.co/state-spaces/mamba-130m.

[11] state-spaces/mamba, . URL https://github.com/state-spaces/mamba.

[12] Structured State Spaces: Combining Continuous-Time, Recurrent, and Convolutional Models, . URL https://hazyresearch.stanford.edu/blog/2022-01-14-s4-3.

[13] Tracking Progress in Natural Language Processing, . URL http://nlpprogress.com/.

[14] triton.language.associative_scan — Triton documentation, . URL https://triton-lang.org/main/python-api/generated/triton.language.associative_scan.html.

[15] Sai Saketh Aluru, Binny Mathew, Punyajoy Saha, i Animesh Mukherjee. Deep Learning Models for Multilingual Hate Speech Detection, Prosinac 2020. URL http://arxiv.org/abs/2004.06465. arXiv:2004.06465 [cs].

[16] Martin Arjovsky, Amar Shah, i Yoshua Bengio. Unitary Evolution Recurrent Neural Networks, Svibanj 2016. URL http://arxiv.org/abs/1511.06464. arXiv:1511.06464 [cs, stat].

[17] Dzmitry Bahdanau, Kyunghyun Cho, i Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate, Svibanj 2016. URL http://arxiv.org/abs/1409.0473. arXiv:1409.0473 [cs, stat].

[18] Iz Beltagy, Matthew E. Peters, i Arman Cohan. Longformer: The Long-Document Transformer, Prosinac 2020. URL http://arxiv.org/abs/2004.05150. arXiv:2004.05150 [cs].

[19] Y. Bengio, P. Simard, i P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, Ožujak 1994. ISSN 1941-0093. doi: 10.1109/72.279181. URL https://ieeexplore.ieee.org/document/279181. Conference Name: IEEE Transactions on Neural Networks.

[20] Stella Biderman, Kieran Bicheno, i Leo Gao. Datasheet for the pile. *arXiv preprint arXiv:2201.07311*, 2022.

[21] G.E. Blelloch. Scans as primitive parallel operations. *IEEE Transactions on Computers*, 38(11):1526–1538, Studeni 1989. ISSN 00189340. doi: 10.1109/12.42122. URL http://ieeexplore.ieee.org/document/42122/.

[22] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, i Adrian Weller. Rethinking Attention with Performers, Studeni 2022. URL http://arxiv.org/abs/2009.14794. arXiv:2009.14794 [cs, stat].

[23] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, i Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, Prosinac 2014. URL http://arxiv.org/abs/1412.3555. arXiv:1412.3555 [cs].

[24] Andrew M. Dai i Quoc V. Le. Semi-supervised Sequence Learning, Studeni 2015. URL http://arxiv.org/abs/1511.01432. arXiv:1511.01432 [cs] version: 1.

[25] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, i Christopher Ré. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness, Lipanj 2022. URL http://arxiv.org/abs/2205.14135. arXiv:2205.14135 [cs].

[26] Thomas Davidson, Dana Warmsley, Michael Macy, i Ingmar Weber. Automated Hate Speech Detection and the Problem of Offensive Language, Ožujak 2017. URL http://arxiv.org/abs/1703.04009. arXiv:1703.04009 [cs].

[27] Jacob Devlin, Ming-Wei Chang, Kenton Lee, i Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Svibanj 2019. URL http://arxiv.org/abs/1810.04805. arXiv:1810.04805 [cs] version: 2.

[28] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The Pile: An 800GB dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

[29] Yoav Goldberg. *Neural Network Methods for Natural Language Processing*. Synthesis Lectures on Human Language Technologies. Springer International

Publishing, Cham, 2017. ISBN 978-3-031-01037-8 978-3-031-02165-7. doi: 10.1007/978-3-031-02165-7. URL https://link.springer.com/10.1007/978-3-031-02165-7.

[30] Ian Goodfellow, Yoshua Bengio, i Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[31] Alex Graves. Generating Sequences With Recurrent Neural Networks, Lipanj 2014. URL http://arxiv.org/abs/1308.0850. arXiv:1308.0850 [cs] version: 5.

[32] Albert Gu. *MODELING SEQUENCES WITH STRUCTURED STATE SPACES*. Doktorska disertacija, Stanford University, 2023.

[33] Albert Gu i Tri Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces, Prosinac 2023. URL http://arxiv.org/abs/2312.00752. arXiv:2312.00752 [cs].

[34] Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, i Christopher Ré. Combining recurrent, convolutional, and continuous-time models with linear state-space layers, 2021. URL https://arxiv.org/abs/2110.13985.

[35] Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, i Christopher Ré. Combining Recurrent, Convolutional, and Continuous-time Models with Linear State-Space Layers, Listopad 2021. URL http://arxiv.org/abs/2110.13985. arXiv:2110.13985 [cs].

[36] Albert Gu, Karan Goel, i Christopher Ré. Efficiently Modeling Long Sequences with Structured State Spaces, Kolovoz 2022. URL http://arxiv.org/abs/2111.00396. arXiv:2111.00396 [cs].

[37] Albert Gu, Ankit Gupta, Karan Goel, i Christopher Ré. On the Parameterization and Initialization of Diagonal State Space Models, Kolovoz 2022. URL http://arxiv.org/abs/2206.11893. arXiv:2206.11893 [cs].

[38] Albert Gu, Isys Johnson, Aman Timalsina, Atri Rudra, i Christopher Ré. How to Train Your HiPPO: State Space Models with Generalized Orthogonal Basis Projections, Kolovoz 2022. URL http://arxiv.org/abs/2206.12037. arXiv:2206.12037 [cs].

[39] Franz A. Heinsen. Efficient Parallelization of a Ubiquitous Sequential Computation, Prosinac 2023. URL http://arxiv.org/abs/2311.06281. arXiv:2311.06281 [cs].

[40] Sepp Hochreiter i Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, Studeni 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

[41] Daniel Jurafsky i James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall PTR, USA, 1st izdanju, 2000. ISBN 0130950696.

[42] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, i François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention, 2020. URL https://arxiv.org/abs/2006.16236.

[43] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, i François Fleuret. Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention, Kolovoz 2020. URL http://arxiv.org/abs/2006.16236. arXiv:2006.16236 [cs, stat].

[44] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, i Ping Tak Peter Tang. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima, Veljača 2017. URL http://arxiv.org/abs/1609.04836. arXiv:1609.04836 [cs, math].

[45] Xin Li, Lidong Bing, Wai Lam, i Bei Shi. Transformation Networks for Target-Oriented Sentiment Classification. U Iryna Gurevych i Yusuke Miyao, urednici, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, stranice 946–956, Melbourne, Australia, Srpanj 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1087. URL https://aclanthology.org/P18-1087.

[46] Yang Liu i Mirella Lapata. Text Summarization with Pretrained Encoders, Rujan 2019. URL http://arxiv.org/abs/1908.08345. arXiv:1908.08345 [cs].

[47] Minh-Thang Luong, Hieu Pham, i Christopher D. Manning. Effective Approaches to Attention-based Neural Machine Translation, Rujan 2015. URL http://arxiv.org/abs/1508.04025. arXiv:1508.04025 [cs].

[48] Eric Martin i Chris Cundy. Parallelizing Linear Recurrent Neural Nets Over Sequence Length, Veljača 2018. URL http://arxiv.org/abs/1709.04057. arXiv:1709.04057 [cs].

[49] Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, i Chris Olah. In-context Learning and Induction Heads, Rujan 2022. URL http://arxiv.org/abs/2209.11895. arXiv:2209.11895 [cs].

[50] Razvan Pascanu, Tomas Mikolov, i Yoshua Bengio. On the difficulty of training Recurrent Neural Networks, Veljača 2013. URL http://arxiv.org/abs/1211.5063. arXiv:1211.5063 [cs].

[51] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, i Adam Lerer. Automatic differentiation in pytorch. U *NIPS-W*, 2017.

[52] Abigail See, Peter J. Liu, i Christopher D. Manning. Get To The Point: Summarization with Pointer-Generator Networks, Travanj 2017. URL http://arxiv.org/abs/1704.04368. arXiv:1704.04368 [cs].

[53] Ilya Sutskever, Oriol Vinyals, i Quoc V. Le. Sequence to Sequence Learning with Neural Networks, Prosinac 2014. URL http://arxiv.org/abs/1409.3215. arXiv:1409.3215 [cs].

[54] Duyu Tang, Bing Qin, Xiaocheng Feng, i Ting Liu. Effective LSTMs for Target-Dependent Sentiment Classification. U Yuji Matsumoto i Rashmi Prasad, urednici, *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, stranice 3298–3307, Osaka, Japan, Prosinac 2016. The COLING 2016 Organizing Committee. URL https://aclanthology.org/C16-1311.

[55] Philippe Tillet, H. T. Kung, i David Cox. Triton: an intermediate language and compiler for tiled neural network computations. U *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, MAPL 2019, stranice 10–19, New York, NY, USA, Lipanj 2019. Association for Computing Machinery. ISBN 978-1-4503-6719-6. doi: 10.1145/3315508.3329973. URL https://doi.org/10.1145/3315508.3329973.

[56] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, i Illia Polosukhin. Attention Is All You Need, Prosinac 2017. URL http://arxiv.org/abs/1706.03762. Number: arXiv:1706.03762 arXiv:1706.03762 [cs].

[57] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, i Samuel Bowman. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. U *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, stranice 353–355, Brussels, Belgium, 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-5446. URL http://aclweb.org/anthology/W18-5446.

[58] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, i Hao Ma. Linformer: Self-Attention with Linear Complexity, Lipanj 2020. URL http://arxiv.org/abs/2006.04768. arXiv:2006.04768 [cs, stat].

[59] Aston Zhang, Zachary C. Lipton, Mu Li, i Alexander J. Smola. *Dive into Deep Learning*. Cambridge University Press, 2023. https://D2L.ai.

[60] Jingqing Zhang, Yao Zhao, Mohammad Saleh, i Peter J. Liu. PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization, Srpanj 2020. URL http://arxiv.org/abs/1912.08777. arXiv:1912.08777 [cs].

[61] Yuxin Jiang . YJiangcm/SST-2-sentiment-analysis, Svibanj 2024. URL https://github.com/YJiangcm/SST-2-sentiment-analysis. original-date: 2020-11-17T08:16:27Z.

Dodatak A
Razlika PyTorch i Triton funkcija

Za razliku od funkcija napisanih u PyTorchu, gdje argumenti funkcija predstavljaju varijable s kojima se može dalje raditi, u Tritonu argumenti funkcija su pokazivači na memorijske adrese gdje su te varijable pohranjene. Argument funkcije može biti i varijabla koja sadrži konstantni izraz, a tada je taj argument funkcije označen s `tl.constexpr`.

Kako bismo razumjeli način rada Tritona, razmotrit ćemo primjer računanja jednostavne kumulativne sume, kao u primjeru u poglavlju 5. Podsjetimo se, radi se o računanju kumulativne ili prefiksne sume za ulazni niz $(1, 2, 3, 4, 5, 6, 7, 8)$. Implementacija takve funkcije je prilično jednostavna u PyTorchu i prikazana je u programskom isječku 8.

Programski isječak 8 Implementacija prefiksne sume u PyTorchu.

```
1 def prefix_sum(x):
2     y = torch.cumsum(x, dim=0)
3     return y
```

PyTorch nudi gotovu implementaciju funkcije `torch.cumsum(x, dim=0)`, te ćemo funkciju `prefix_sum` jednostavno pozvati s argumentom `x = [1, 2, 3, 4, 5, 6, 7, 8]`.

Implementacija iste funkcije `prefix_sum` u Tritonu prikazana je u isječku 9.

Programski isječak 9 Implementacija prefiksne sume u Tritonu.

```
@triton.jit
def prefix_sum(X, Y, L: tl.constexpr):
    Ls = tl.arange(0, L)

    x = tl.load(X + Ls)

    y = tl.associative_scan(x, 0, plus_fn)

    tl.store(Y + Ls, y)
```

U Tritonu, funkcija `prefix_sum` umjesto polja `x = [1, 2, 3, 4, 5, 6, 7, 8]` prima pokazivač na memorijsku adresu gdje se nalazi prvi element polja, označen s X. Funkcija u Tritonu ne vraća izravno vrijednost pomoću naredbe `return`, nego izračunatu kumulativnu sumu sprema na memorijsku adresu na koju pokazuje argument funkcije Y. Funkcija također prima argument `L` koji predstavlja duljinu polja. Taj argument je potreban kako bi se unutar funkcije ispravno učitalo polje `x` na kojem će se računati kumulativna suma te ispravno spremilo rezultantno polje `y`. Funkcija `tl.associative_scan` je učinkovita implementacija skena čiji argumenti su redom: polje za koje se računa kumulativna suma, element identiteta te binaran asocijativan operator, što je u ovom slučaju obični operator zbrajanja.

Prednost implementacije u Tritonu nad implementacijom u PyTorchu je što u Tritonu imamo točnu kontrolu nad operacijama čitanja i pisanja u memoriju. Smanjivanjem broja operacija pisanja i čitanja u memoriju poboljšavamo svojstva programa koji se izvršava.

Implementacija i analiza arhitekture Mamba za neuronske jezične modele

Sažetak

Duboko učenje proteklih godina ostvaruje iznimne rezultate, pogotovo u području obrade prirodnog jezika kroz uporabu velikih jezičnih modela. Fokus istraživačke zajednice većinski je bio usmjeren na arhitekturu Transformatora kao temelj velikih jezičnih modela. Međutim, nedavna inovacija, arhitektura Mamba, pokazala je potencijal za postizanje usporedivih rezultata s boljom vremenskom i prostornom složenošću. U ovom radu, detaljno su razmotreni matematički temelji Mambe te optimizacijske tehnike za izvedbu na grafičkim procesorima. Za potrebe rada, arhitektura Mambe implementirana je u programskom okviru PyTorch te je nad tom implementacijom provedena evaluacija na različitim zadatcima obrade prirodnog jezika.

Ključne riječi: Veliki jezični modeli, Transformatori, Mamba, Duboko učenje, Obrada prirodnog jezika

Implementation and Analysis of the Mamba Architecture for Neural Language Models

Abstract

In recent years, deep learning has achieved remarkable results, especially in the field of natural language processing through the use of large language models. The research community has primarily focused on the Transformer architecture as the foundation of large language models. However, a recent innovation, the Mamba architecture, has shown the potential to achieve comparable results with better temporal and spatial complexity. This thesis provides a detailed examination of the mathematical foundations of Mamba and optimization techniques for implementation on graphics processors. For the purposes of this thesis, the Mamba architecture was implemented in the PyTorch framework, and evaluations were conducted on various natural language processing tasks.

Keywords: Large Language Models, Transformers, Mamba, Deep Learning, Natural Language Processing