Povratne neuronske mreže i primjene

Kabić, Stipe

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Stipe Kabić

POVRATNE NEURONSKE MREŽE I PRIMJENE

Diplomski rad

Voditelj rada:
izv. prof. dr. sc. Zvonimir Bujanović

Zagreb, Ožujak, 2023

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

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

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Obitelji bez koje do ovog puta ne bih ni došao i prijateljima bez kojih ga ne bih prešao.

Sadržaj

Sadržaj **iv**

Uvod **1**

1 Neuronske mreže **3**
1.1 Osnovni pojmovi 3
1.2 Gradijentni spust i propagacija unatrag 6

2 Povratne neuronske mreže **13**
2.1 Osnovne povratne neuronske mreže 13
2.2 Napredne arhitekture povratnih neuronskih mreža 19

3 Separacija instrumenata iz glazbe **25**
3.1 Uvod u problem 25
3.2 Fourierova transformacija i spektrogram 27

4 Primjena povratnih neuronskih mreža na separaciju glazbe **35**

Bibliografija **43**

Uvod

U zadnjem desetljeću tehnološki je napredak u razvoju modernih grafičkih kartica doveo do svojevrsne revolucije u području strojnog učenja. Treniranje dubokih neuronskih mreža, modela koji sadrže ogroman broj parametara, postalo je moguće. To je dovelo do mnogih uspjeha na područjima poput računalnog vida i obrade prirodnog jezika. Ovisno o primjeni i prirodi podataka, postoje razne podvrste neuronskih mreža koje se mogu koristiti. U ovom radu dajemo pregled jedne od tih vrsta, a to su povratne neuronske mreže čija je specijalnost modeliranje nizova. Najčešće se koriste nad tekstualnim podacima ili vremenskim nizovima, ali u ovom radu bavit ćemo se jednom drugom primjenom.

U glazbenoj industriji pjesme nastaju snimanjem audiozapisa individualnih instrumenata i vokala, a zatim spajanjem istih u jedan zapis. Ono što nije toliko jednostavno je inverzni proces, odnosno dobivanje individualnih instrumenata i vokala iz zapisa čitave pjesme. Model koji uspješno provodi taj proces bio bi izrazito koristan glazbenicima i stoga je ovaj problem godinama tema mnogih znanstvenih istraživanja. Originalno, pristupi su se temeljili na matričnim faktorizacijama i drugim metodama iz linearne algebre, uz korištenje metoda iz Fourierove analize za početnu obradu podataka.

U novije vrijeme, neuronske mreže su preuzele ulogu klasičnih pristupa i donijele značajan napredak. Za ovaj problem korištene su razne vrste mreža, a u ovom radu ćemo se fokusirati na arhitekture temeljene na povratnim neuronskim mrežama.

U prvom poglavlju rada izlažemo osnovni teorijski uvod u neuronske mreže i proces njihova treniranja. U drugom poglavlju u fokus ulaze povratne neuronske mreže i daje se detaljna analiza glavnih arhitektura. Treće poglavlje počinje se baviti konkretnom primjenom; u njemu izlažemo teorijske rezultate vezane za Fourierovu transformaciju koji će nam biti potrebni za obradu zvučnih podataka. Konačno, u četvrtom poglavlju predlažemo konkretnu arhitekturu neuronske mreže za problem separacije glazbe, treniramo taj model i iznosimo analizu dobivenih rezultata.

Poglavlje 1

Neuronske mreže

Neuronske mreže su širok skup modela koji se temelje na istim osnovnim principima i algoritmima za učenje. U ovom radu se fokusiramo na povratne neuronske mreže, ali za njihovo razumijevanje potrebno je poznavanje jedne jednostavnije vrste mreža, a to su unaprijedne neuronske mreže (eng. feedforward neural networks). Naziv dolazi iz načina na koji informacije prolaze kroz model, što će biti jasnije nakon formalne definicije.

1.1 Osnovni pojmovi

Neuronske mreže sastoje se od velikog broja instanci njihovih osnovnih elemenata koje nazivamo neuroni. Za dani ulazni vektor $x \in \mathbb{R}^n$ izlaz neurona je $y \in \mathbb{R}$ definiran s

$$y = f(\sum_{i=1}^{n} w_i x_i + b),$$

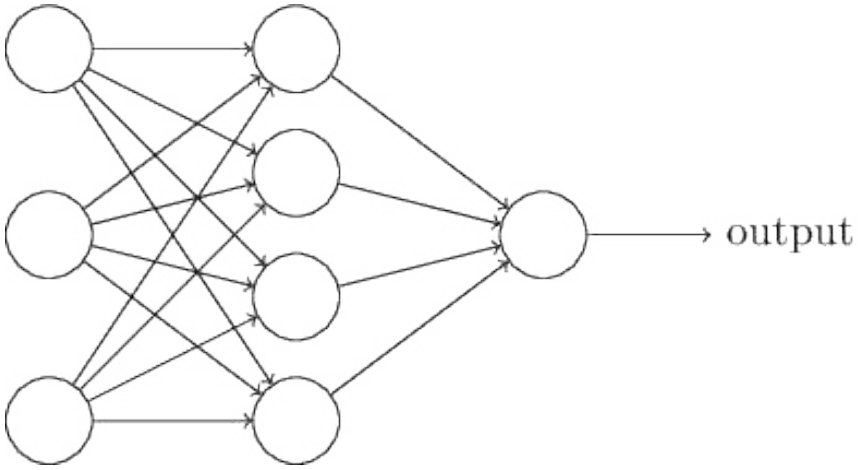
gdje su $(w_i)_{i=1}^n$ težine (eng. weights) neurona, $b \in \mathbb{R}$ pristranost (eng. bias) neurona, a $f\colon \mathbb{R} \to \mathbb{R}$ aktivacijska funkcija.

Intuitivno, težine predstavljaju važnost pripadnih elemenata ulaznog vektora za izlaz neurona, dok pristranost i aktivacijska funkcija translatiraju i skaliraju izlaz neurona. Cilj procesa učenja neuronske mreže je pronaći optimalne vrijednosti parametara $(w_i)_{i=1}^n$ i b koji za ulazne podatke $x \in \mathbb{R}^n$ daju dobru aproksimaciju neke funkcije $g\colon \mathbb{R} \to \mathbb{R}$ koja je zadana parovima $(x_j, g(x_j))$, $j = 1, ..., m, m \in \mathbb{N}$, odnosno postići da vrijedi $y \approx g(x)$.

S druge strane, aktivacijsku funkciju biramo pri konstrukciji arhitekture neuronske mreže, pri čemu se najčešće koristi nekoliko standardnih funkcija. Izbor ovisi o arhitekturi mreže i prirodi podataka, a u nastavku rada koristit ćemo sljedeće aktivacijske funkcije.

- Sigmoidalna aktivacijska funkcija koja je dana s $f(x) = \frac{1}{1+e^{-x}}$
- Tangens hiperbolni koji je dan s $f(x) = \frac{e^{2x}-1}{e^{2x}+1}$
- ReLU (eng. Rectified Linear Unit) koja je dana s $f(x) = max(0, x)$

Unaprijedne neuronske mreže dobivaju se slaganjem ovako definiranih neurona u slojeve kao na sljedećem dijagramu.



Slika 1.1: Unaprijedna neuronska mreža s tri sloja. Prvi (ulazni) sloj ima tri neurona, srednji (tzv. skriveni) sloj ima četiri neurona dok zadnji sloj ima jedan neuron.

Dakle, ulazni vektor za neurone u nekom sloju predstavlja konkatenacija izlaza nekih neurona iz prethodnog sloja. Na kraju mreže često imamo (kao na gornjoj slici) izlazni neuron čiji izlaz predstavlja konačnu predikciju našeg modela. Duboko učenje se sastoji od konstrukcije ovakvih mreža s velikim brojem slojeva, odnosno velikom dubinom, i procesa njihova učenja nad podacima. Često se koriste potpuno povezane neuronske mreže kod kojih svaki neuron nekog sloja kao ulaz dobiva izlaze svih neurona iz prethodnog sloja.

Preciznije, neka je s $w_{ij}^{(l)}$ dana i-ta težina j-tog neurona iz l-tog sloja mreže, a s $b_j^{(l)}$ pristranost j-tog neurona iz l-tog sloja mreže. Definiramo matricu

$$W^{(l)} = \begin{bmatrix} w_{11}^{(l)} & \dots & w_{1n}^{(l)} \\ \vdots & \ddots & \vdots \\ w_{m1}^{(l)} & \dots & w_{mn}^{(l)} \end{bmatrix}$$

koja sadrži sve težine l-tog sloja mreže i vektor

$$b^{(l)} = \begin{bmatrix} b_1^{(l)} \\ \vdots \\ b_n^{(l)} \end{bmatrix}$$

koji sadrži sve pristranosti l-tog sloja mreže. Tada je izlaz l-tog sloja mreže dan s

$$x^{(l)} = f(W^{(l)}x^{(l-1)} + b^{(l)})$$

gdje je $x^{(l-1)}$ izlaz $(l-1)$-og sloja, a $x^{(0)}$ vektor ulaznih podataka. Ovdje koristimo pokratu

$$f(x) = (f(x_1), ..., f(x_n)), f : \mathbb{R} \to \mathbb{R}, x \in \mathbb{R}^n.$$

Jednom kad smo definirali model, zanima nas kako pronaći optimalne težine i pristranosti za svaki neuron u mreži. U tu svrhu potrebno je definirati neku mjeru kvalitete modela, a za to koristimo funkciju gubitka (eng. loss function). Ona kao ulaz prima stvarnu vrijednost zavisne varijable koju pokušavamo aproksimirati te procjenu našeg modela i vraća mjeru njihove sličnosti. Stoga je cilj procesa učenja minimizirati ovu funkciju. Ovisno o prirodi zavisne varijable, postoje razne funkcije gubitka. Dva najčešća slučaja su regresija, gdje je zavisna varijabla realan broj, i klasifikacija, gdje je zavisna varijabla element nekog konačnog skupa.

U slučaju regresije koristi se srednjekvadratna greška MSE (eng.mean squared error) koja je dana s

$$MSE(\hat{y}, y) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

ili apsolutna greška MAE (eng. mean absolute error) koja je dana s

$$MAE(\hat{y}, y) = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|$$

dok se za klasifikaciju najčešće primjenjuje unakrsna entropija (eng. cross entropy) koja je dana s

$$CE(\hat{y}, y) = -\sum_{i=1}^{n} y_i log(\hat{y}_i)$$

U našem radu ćemo uglavnom raditi s varijacijama na prve dvije funkcije gubitka, uz određene prilagodbe specifičnim podacima na koje ćemo primjenjivati mreže.

1.2 Gradijentni spust i propagacija unatrag

Sada imamo definiranu neuronsku mrežu i funkciju gubitka koju želimo minimizirati. Parametri nad kojima imamo kontrolu su težine $(w_i)_{i=1}^n$ i pristranosti b svakog od neurona koji sačinjava našu mrežu. Poznato je da negativni gradijent funkcije u nekoj točki predstavlja smjer najbržeg pada te funkcije u toj točki, što motivira algoritam gradijentnog spusta kojim se provodi proces učenja neuronske mreže. Nadalje, zbog dubine modernih neuronskih mreža i ogromnog broja parametara, potreban je efikasan način za računanje gradijenata funkcije gubitka po svim težinama. Za tu svrhu koristi se algoritam propagacije unatrag (eng. backpropagation) kojeg ćemo precizno izvesti.

Neka je θ vektor koji sadrži sve parametre dane neuronske mreže, a $L = L(x; \theta)$ funkcija gubitka. Za dovoljno mali $\epsilon \in \mathbb{R}$ tada vrijedi (ako se već ne nalazimo u optimumu)

$$L(x; \theta - \epsilon \nabla_\theta L(x; \theta)) < L(x; \theta)$$

što motivira algoritam učenja koji iterativno radi pomake oblika

$$\theta_{new} = \theta_{old} - \epsilon \nabla_\theta L(x; \theta)$$

pri čemu se za ϵ uzima mala konstanta koju nazivamo faktor učenja (eng. learning rate). Faktor učenja predstavlja hiperparametar koji je potrebno eksperimentalno odrediti. Premale vrijednosti rezultirat će sporim učenjem, dok prevelike vrijednosti daju oscilacije u vrijednosti funkcije gubitka te čine trening modela nestabilnim.

Ovu metodu učenja nazivamo gradijentni spust. Ovisno o količini podataka koju koristimo za svaku iteraciju gradijentnog spusta, postoje tri verzije ovog algoritma. Prva verzija je gradijentni spust po grupama (eng. batch gradient descent) koji za svaku iteraciju koristi čitavi skup podataka za trening. Mana ovog pristupa je brzina jer je za svako ažuriranje parametara potrebno proći kroz čitavi, potencijalno vrlo veliki, skup podataka. Druga verzija je stohastički gradijentni spust koji svako ažuriranje radi na osnovu jednog slučajno (uniformno) odabranog podatka, oslanjajući se na činjenicu da je očekivana vrijednost ovako izračunatih gradijenata jednaka punom gradijentu. Ova metoda je puno brža, ali nestabilnija. Treća metoda, koja pokušava spojiti prednosti obaju prethodnih, naziva se gradijentni spust po podgrupama (eng. minibatch gradient descent). Ova metoda radi ažuriranje parametara pomoću gradijenta izračunatog nad manjim podskupom cijelog skupa podataka. Ona zapravo obuhvaća obje prethodne metode (za veličinu podskupa možemo uzeti 1 ili veličinu punog skupa). Na ovaj način dobivamo veću stabilnost i mogućnost korištenja paralelizacije, ali ne gubimo previše na brzini. Veličina ovog podskupa je hiperparametar koji biramo na osnovu dostupnih računalnih resursa i samih podataka.

Postoje razne osnovnog algoritma gradijentnog spusta koje u praksi daju bolje rezultate. Neke od njih su algoritmi Momentum, AdaGrad i RMSProp. Najpopularniji algoritam za učenje koji koristi neke od tih trikova je Adam. S obzirom da ćemo taj algoritam koristiti u praktičnom dijelu rada, ukratko ćemo ga objasniti. Glavne ideje na kojima se zasniva su praćenje prosjeka i varijance vrijednosti gradijenata kroz trening te njihovo normaliziranje zbog stabilnijeg procesa učenja. On radi u nekoliko koraka.

Prvo, izračunamo gradijent u novom podatku ili podgrupi

$$g_t = \nabla f(\theta_{t-1})$$

Taj gradijent dodamo varijabli m_t koju shvaćamo kao težinsku sredinu svih dosadašnjih gradijenata

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

Ažuriramo varijablu v_t koju shvaćamo kao varijancu svih dosadašnjih gradijenata

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Normaliziramo varijable m_t i v_t

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Izračunamo iduću vrijednost parametra:

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{[\sqrt{\hat{v}_t} + \epsilon]}$$

Iako i ovaj algoritam ima svoje nedostatke te nije nužno optimalan u svakoj situaciji, u praksi najčešće daje dobre rezultate bez potrebe za detaljnim traženjem optimalnih hiperparametara. Stoga je čest prvi izbor za optimizacijski algoritam u procesu učenja. Standardne vrijednosti hiperparametara su $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-8, \alpha = 0.001$.

Nakon izbora optimizacijskog algoritama preostaje odabrati optimalan faktor učenja. Spomenimo da uz odabir fiksnog faktora učenja postoje i pristupi koji definiraju raspored (eng. schedule) koji određuje različite vrijednosti faktora učenja u različitim fazama treninga.

Najčešća praksa je započeti trening s manjom vrijednosti, postupno ju povećavati u prvoj fazi, a zatim ju ponovno smanjiti na kraju treninga. Intuitivno, prvo dajemo algoritmu učenja malo vremena da se orijentira, zatim u početnoj fazi agresivno mijenjamo parametre s ciljem brzog učenja osnovnih uzoraka u podacima, a zatim na samom kraju radimo male i precizne pomake za učenje najtežih uzoraka. Alternativni pristup je periodički raspored gdje ovakav ciklus ponavljamo više puta kroz trening.

Kako bismo proveli postupak učenja potreban nam je efikasan način za računanje gradijenta. Naime, duboke neuronske mreže sadrže ogroman broj parametara. Ako bismo parcijalnu derivaciju po svakom parametru računali nezavisno, vrlo brzo bi mreže postale prevelike da bi takav proces bio izvediv u razumnoj količini vremena. Ipak, zbog slojevite strukture mreže, moguće je sve parcijalne derivacije izračunati samo jednim prolaskom kroz mrežu. Algoritam, odnosno skup jednadžbi, koji na taj način računa gradijent naziva se propagacija unatrag (eng. backpropagation).

Ideja iza algoritma je korištenje lančanog pravila za deriviranje kompozicije funkcija. Naime, neuronsku mrežu možemo promatrati kao kompoziciju svojih slojeva. Prisjetimo se, za $x \in \mathbb{R}$ i derivabilne funkcije $f\colon \mathbb{R} \to \mathbb{R}$, $g\colon \mathbb{R} \to \mathbb{R}$ neka je $y = f(x)$, $z = g(y) = g(f(x))$. Tada vrijedi

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}.$$

Analogno, za $x \in \mathbb{R}^n$ i derivabilne funkcije $f\colon \mathbb{R}^n \to \mathbb{R}^m$, $g\colon \mathbb{R}^m \to \mathbb{R}$ neka je $y = f(x)$, $z = g(y) = g(f(x))$. Tada vrijedi

$$\frac{\partial z}{\partial x_i} = \sum_{j=1}^{m} \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i}.$$

Koristeći ovaj rezultat možemo sekvencijalno računati parcijalne derivacije funkcije gubitka po parametrima mreže prolaskom unatrag po slojevima. U prvom koraku direktno izračunamo parcijalne derivacije po parametrima zadnjeg sloja. Koristeći dobivene vrijednosti u idućem koraku koristeći gornje pravilo dobijemo parcijalne derivacije po parametrima predzadnjeg sloja. Iterativnim postupkom prolazimo unatrag kroz mrežu i računamo parcijalne derivacije po svim slojevima višestrukom primjenom gornjeg lančanog pravila. Ovim postupkom dobivamo gradijent funkcije gubitka koji koristimo u gradijentnom spustu.

Pretpostavimo da imamo unaprijednu neuronsku mrežu s L slojeva. Prisjetimo se, izlaz l-tog sloja mreže dan je s

$$x^{(l)} = f(W^{(l)}x^{(l-1)} + b^{(l)})$$

gdje je $x^{(l-1)}$ izlaz $(l-1)$-og sloja, $x^{(0)}$ vektor ulaznih podataka, a $L(x, \theta)$ funkcija gubitka. Ono što želimo izračunati su vrijednosti $\frac{\partial L}{\partial w_{jk}^{(l)}}$ i $\frac{\partial L}{\partial b_j^{(l)}}$ za sve slojeve mreže. Za početak, uvedimo nekoliko korisnih oznaka. Neka je $z^{(l)} = W^{(l)}x^{(l-1)} + b^{(l)}$ i $\delta_j^{(l)} = \frac{\partial L}{\partial z_j^{(l)}}$. Sada ćemo prvo izvesti formulu za izraz $\delta_j^{(l)}$, a zatim ćemo pomoću $\delta_j^{(l)}$ dobiti jednostavne formule za računanje $\frac{\partial L}{\partial w_{jk}^{(l)}}$ i $\frac{\partial L}{\partial b_j^{(l)}}$.

Za početak, pokažimo da vrijedi

$$\delta_j^{(L)} = \frac{\partial L}{\partial y_j} f'(z_j^{(L)})$$

Po definiciji imamo

$$\delta_j^{(L)} = \frac{\partial L}{\partial z_j^{(L)}}.$$

Koristeći lančano pravilo dobivamo

$$\delta_j^{(L)} = \sum_{k=1}^{m} \frac{\partial L}{\partial y_k} \frac{\partial y_k}{\partial z_j^{(L)}}.$$

Naravno, y_i ovisi o $z_j^{(L)}$ samo kad je $i = j$ pa slijedi

$$\delta_j^{(L)} = \frac{\partial L}{\partial y_j} \frac{\partial y_j}{\partial z_j^{(L)}}.$$

S obzirom da je $y_j = f(z_j^{(L)})$ zapravo imamo

$$\delta_j^{(L)} = \frac{\partial L}{\partial y_j} f'(z_j^{(L)})$$

što smo i htjeli pokazati.

Za dva vektora $x = (x_1, ..., x_n) \in \mathbb{R}^n$ i $y = (y_1, ..., y_n) \in \mathbb{R}^n$ možemo definirati Hadamardov produkt kao

$$x \odot y = (x_1 y_1, ..., x_n y_n)$$

Sada ćemo pokazati da vrijedi

$$\delta^{(l)} = ((W^{(l+1)})^T \delta^{(l+1)}) \odot f'(z^{(l)})$$

Imamo

$$\begin{aligned}\delta_j^{(l)} &= \frac{\partial L}{\partial z_j^{(l)}} \\ &= \sum_{k=1}^{m} \frac{\partial L}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} \\ &= \sum_{k=1}^{m} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} \delta_k^{(l+1)}.\end{aligned}$$

Uočimo, vrijedi

$$z_k^{(l+1)} = \sum_j w_{kj}^{(l+1)} f(z_j^{(l)}) + b_k^{(l+1)}$$

pa diferenciranjem slijedi

$$\frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = w_{kj}^{(l+1)} f'(z_j^{(l)}).$$

Supstitucijom nazad dobivamo

$$\delta_j^{(l)} = \sum_{k=1}^{m} w_{kj}^{(l+1)} f'(z_j^{(l)}) \delta_k^{(l+1)}$$

što je upravo originalna tvrdnja zapisana po komponentama. Konačno, pokazat ćemo da za pristranosti mreže vrijedi

$$\frac{\partial L}{\partial b_j^{(l)}} = \delta_j^{(l)}$$

dok za težine vrijedi

$$\frac{\partial L}{\partial w_{jk}^{(l)}} = f(z_k^{(l-1)}) \delta_j^{(l)}.$$

Krenimo s prvom tvrdnjom. Imamo

$$\begin{aligned}\delta_j^{(l)} &= \frac{\partial L}{\partial z_j^{(l)}} \\ &= \sum_{k=1}^{m} \frac{\partial L}{\partial b_k^{(l)}} \frac{\partial b_k^{(l)}}{\partial z_j^{(l)}}.\end{aligned}$$

Kao i prije, ova suma se reducira na

$$\delta_j^{(l)} = \frac{\partial L}{\partial b_j^{(l)}} \frac{\partial b_j^{(l)}}{\partial z_j^{(l)}}.$$

Iz izraza

$$z_k^{(l)} = \sum_j w_{kj}^{(l)} f(z_j^{(l-1)}) + b_k^{(l)}$$

slijedi

$$b_k^{(l)} = z_k^{(l)} - \sum_j w_{kj}^{(l)} f(z_j^{(l-1)})$$

pa vrijedi

$$\frac{\partial b_k^{(l)}}{\partial z_k^{(l)}} = 1$$

što uvrštavanjem u gornju jednadžbu daje

$$\frac{\partial L}{\partial b_j^{(l)}} = \delta_j^{(l)}$$

a to smo htjeli pokazati. Za drugu tvrdnju analogno računamo

$$\frac{\partial L}{\partial w_{jk}^{(l)}} = \sum_{i=1}^{m} \frac{\partial L}{\partial z_i^{(l)}} \frac{\partial z_i^{(l)}}{\partial w_{jk}^{(l)}}$$

što se reducira na

$$\begin{aligned} \frac{\partial L}{\partial w_{jk}^{(l)}} &= \frac{\partial L}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}} \\ &= \delta_j^{(l)} \frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}}. \end{aligned}$$

Sada iz

$$z_j^{(l)} = \sum_i w_{ji}^{(l)} f(z_i^{(l-1)}) + b_j^{(l)}$$

slijedi

$$\frac{\partial z_j^{(l)}}{\partial w_{jk}^{(l)}} = f(z_k^{(l-1)})$$

pa uvrštavanjem imamo

$$\frac{\partial L}{\partial w_{jk}^{(l)}} = f(z_k^{(l-1)})\delta_j^{(l)}.$$

što je upravo tražena tvrdnja. Uočimo, točke u kojima je potrebno izvrijediti derivaciju od f su upravo vrijednosti izračunate propagacijom ulaznog podatka $x^{(0)}$ kroz mrežu, što znači da je za propagaciju unatrag potrebno prvo provesti propagaciju unaprijed (eng. forward pass).

Poglavlje 2

Povratne neuronske mreže

U prethodnom poglavlju predstavili smo klasične unaprijedne neuronske mreže. U brojnim primjenama ti modeli daju odlične rezultate. Ipak, za određene tipove ulaznih podataka postoje specijalizirane vrste neuronskih mreža koje su prilagođene njihovoj strukturi. Primjerice, jedna česta vrsta podataka su nizovi koji imaju određenu vremensku strukturu. Tekstualni podaci, zvučni podaci i klasični vremenski nizovi svi spadaju u tu kategoriju. S obzirom da jednostavne neuronske mreže ne poznaju koncept redoslijeda podataka, jasno je da one neće biti najprikladniji alat za modeliranje takvih procesa. Stoga se javlja potreba za bolje prilagođenom arhitekturom, a odgovor su upravo povratne neuronske mreže. Slike u ovom poglavlju preuzete su iz [11].

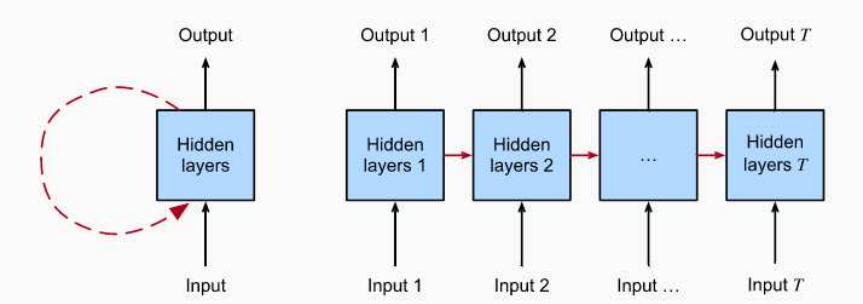
2.1 Osnovne povratne neuronske mreže

Povratne neuronske mreže (eng. recurrent neural network ili skraćeno RNN) dobivene su jednostavnom modifikacijom standardnih mreža koja im omogućava modeliranje nizova podataka. U klasičnom slučaju, podaci kroz mrežu putuju samo u jednom smjeru, od ulaza prema izlazu. Također, svaki ulazni podatak kroz mrežu prolazi neovisno o ostalim podacima. Kod povratnih mreža uz novi ulazni podatak mreža dobiva i izlaz koji pripada prethodnom podatku, što omogućava modeliranje niza. Prvo ćemo definirati povratni sloj u općenitijem obliku, a zatim prijeći na konkretnu verziju koja se koristi u praksi. Za niz ulaznih podataka $x_1, ..., x_t \in \mathbb{R}^n$ djelovanje sloja definiramo s

$$o_t = f(x_t, o_{t-1}, w_o)$$

gdje w_o označava kolekciju svih parametara mreže, a $f\colon \mathbb{R}^n \to \mathbb{R}^l$ je proizvoljna diferencijabilna funkcija.

Iz definicije je jasno da izlaz u trenutku t ovisi o izlazu iz trenutka $t-1$ koji ovisi o ulazu iz trenutka $t-1$. Ako rekurzivno nastavimo dalje, vidimo da izlaz y_t ovisi o svim ulazima $x_1, ..., x_t$. Uobičajeno definiramo $y_0 = 0$ kako bi definicija imala smisla i za prvi podatak.

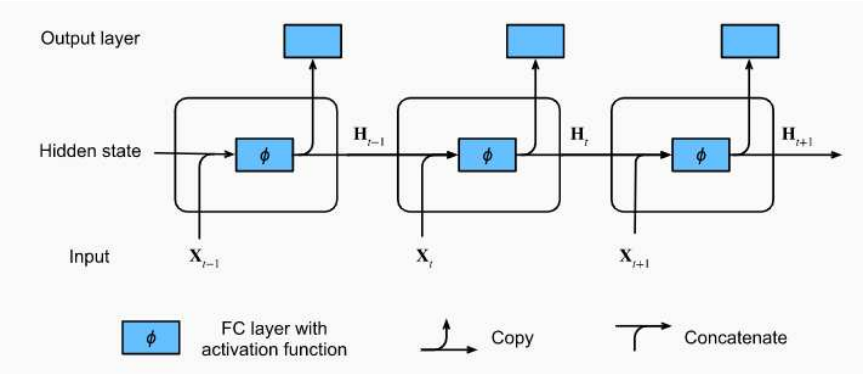


Slika 2.1: RNN sloj

U nekim situacijama, prikladno je na izlaz sloja primijeniti dodatnu aktivacijsku funkciju g, a idućem sloju proslijediti stanje prije primjene funkcije g. Tada izlaz mreže zovemo skriveno stanje (eng. hidden state) i označavamo ga s h pa imamo

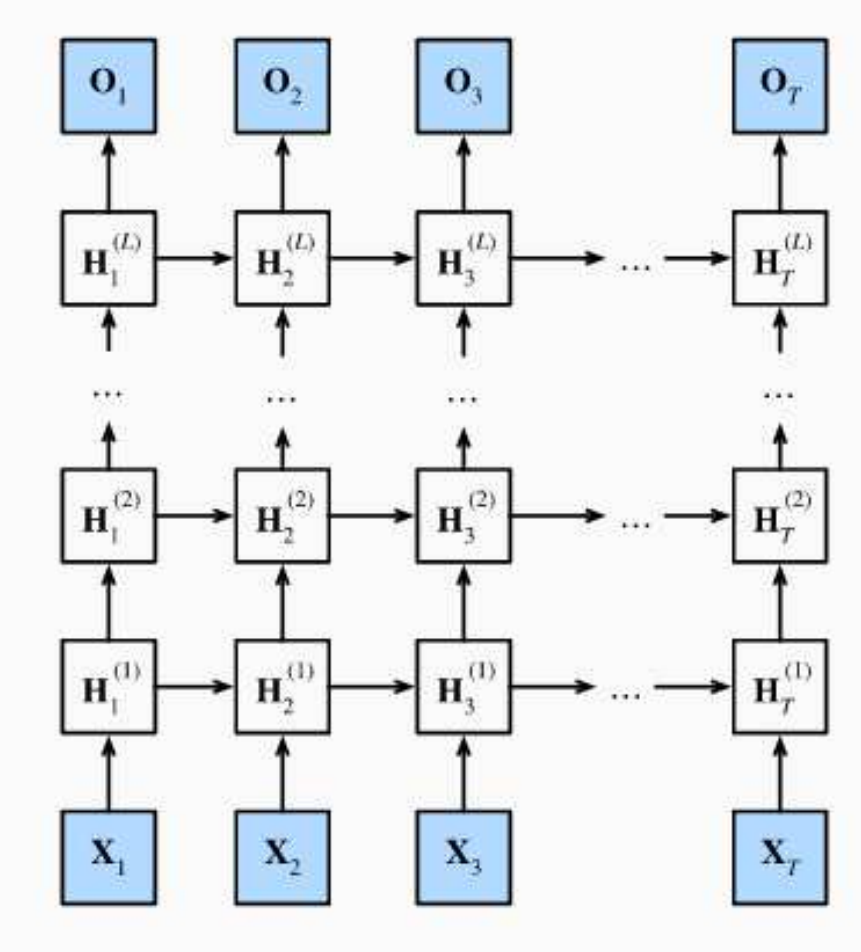
$$h_t = f(x_t, h_{t-1}, w_h) \tag{2.1}$$
$$o_t = g(h_t, w_o) \tag{2.2}$$

Ovdje w_h označava kolekciju svih parametara mreže o kojima ovisi računanje h_t dok w_o označava kolekciju svih parametara o kojima ovisi računanje o_t. Kao kod običnih ne-



Slika 2.2: Unutarnji mehanizam RNN sloja sa skrivenim stanjem

uronskih mreža, duboke povratne mreže nastaju slaganjem velikog broja ovako definiranih slojeva.



Slika 2.3: Duboka povratna neuronska mreža. Ulazni podatak x_i redom prolazi kroz L slojeva (na slici od dolje prema gore), a izlazna vrijednost u svakom sloju se izračunava pomoću one iz prethodnog sloja (dolje) i skrivenog stanja izračunatog u istom tom sloju za x_{i-1} (lijevo).

Po gornjoj definiciji, povratna neuronska mreža za svaki ulazni podatak x_i vraća pripadni izlazni podatak o_i, za $i = 1, ..., n$. Takav tip modela nazivamo "niz u niz" model. Jedna od primjena za ovakav model je predikcija vremenskih nizova.

S druge strane, mogli bismo zanemariti sve osim posljednjeg izlaznog podatka. Takav tip modela nazivamo "niz u vektor". Neke od primjena su klasifikacija vremenskih nizova ili teksta. Slično, mogli bismo za samo jedan ulazni podatak koji stalno ponavljamo tražiti niz kao izlaz, pa dobivamo mrežu tipa "vektor u niz". Jedna od primjena je generiranje

tekstualnog opisa za sliku. Konačno, mogli bismo imati dvije mreže - enkoder i dekoder. Enkoder bi bio mreža tipa "niz u vektor" dok bi dekoder bio mreža tipa "vektor u niz". Ovo nam opet daje model tipa "niz u niz", no on bi radio i u slučajevima kad ulazni i izlazni niz nisu nužno iste duljine. Neki od primjera problema koji koriste ovakve modele su prevođenje teksta i transkripcija govora.

Treniranje povratnih neuronskih mreža slično je treniranju klasičnih mreža. Naime, ako mrežu promatramo u "razmotanom" obliku imamo jednostavnu jednadžbu koja opisuje djelovanje jednog sloja mreže. Sada možemo primijeniti algoritam propagacije unatrag kroz vrijeme koji se temelji na već spomenutoj propagaciji unatrag i dobiti gradijent funkcije gubitka po svim parametrima mreže. Ostatak procesa učenja potpuno je jednak kao onaj u prošlom poglavlju.

Sada ćemo izvesti algoritam za propagaciju unatrag kroz vrijeme. Standardno, imamo funkciju gubitka

$$L(x_1, ..., x_t, y_1, ..., y_t, w_h, w_o) = \frac{1}{T}\sum_{t=1}^{T} l(y_t, o_t)$$

i zanimaju nas vrijednosti $\frac{\partial L}{\partial w_h}, \frac{\partial L}{\partial w_o}$. Drugi izraz se dobije relativno lako, dok za prvi izraz imamo

$$\begin{aligned}\frac{\partial L}{\partial w_h} &= \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l(y_t, o_t)}{\partial w_h} \\ &= \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l(y_t, o_t)}{\partial o_t}\frac{\partial g(h_t, w_o)}{\partial h_t}\frac{\partial h_t}{\partial w_h}\end{aligned}$$

Prva dva faktora u svakom od gornjih sumanada se lako izračunaju. Za treći faktor pak imamo

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial w_h}.$$

Za nastavak nam je potrebna jednostavna pomoćna tvrdnja. Za tri niza $(a_t)_t, (b_t)_t, (c_t)_t$ koji zadovoljavaju relacije

$$\begin{aligned}a_0 &= 0, \\ a_t &= b_t + c_t a_{t-1}, t \geq 1\end{aligned}$$

vrijedi

$$a_t = b_t + \sum_{i=1}^{t-1}(\prod_{j=i+1}^{t} c_j)b_i.$$

Koristeći se ovom relacijom imamo

$$\frac{\partial h_t}{\partial w_h} = \frac{\partial f(x_t, h_{t-1}, w_h)}{\partial w_h} + \sum_{i=1}^{t-1}(\prod_{j=i+1}^{t} \frac{\partial f(x_j, h_{j-1}, w_h)}{\partial h_{j-1}})\frac{\partial f(x_i, h_{i-1}, w_h)}{\partial w_h}$$

Promotrimo sada konkretnu implementaciju povratnog sloja. Neka je dan niz ulaznih podataka $x_1, ..., x_t \in \mathbb{R}^n$. Umjesto jedne matrice težina W, povratni sloj sastoji se od dvije matrice W_x i W_o, uz vektor pristranosti b. Djelovanje sloja na ulazni podatak dano je s

$$o_t = f(W_x x_t + W_o o_{t-1} + b)$$

gdje je f aktivacijska funkcija.

Verzija sloja koja ima i skriveno stanje je u ovom slučaju dana s

$$h_t = f(W_x x_t + W_h h_{t-1} + b) \tag{2.3}$$
$$o_t = g(W_o h_t) \tag{2.4}$$

Uočimo, ovako definiran sloj je specijalan slučaj sloja definiranog u (2.1) i (2.2).

Prikažimo sada propagaciju unatrag kroz vrijeme na konkretnijem primjeru. Promotrit ćemo mrežu koja nema pristranosti, a za aktivacijsku funkciju koristi identitetu. Ovime ćemo pojednostaviti račun i staviti fokus na ključne detalje, a lako se vidi da se izvod koji slijedi jednostavno generalizira i na druge slučajeve. Neka je model dan s

$$h_t = W_x x_t + W_h h_{t-1}$$
$$o_t = W_o h_t.$$

Uočimo, ovako definiran sloj je specijalan slučaj sloja definiranog u (2.3) i (2.4). Kao i prije, imamo

$$L = \frac{1}{T}\sum_{t=1}^{T} l(o_t, y_t)$$

Za početak, jednostavno je izračunati

$$\frac{\partial L}{\partial o_t} = \frac{1}{T}\frac{\partial l}{\partial o_t}$$

Koristeći se ovom relacijom lako dobivamo

$$\begin{aligned}\frac{\partial L}{\partial W_o} &= \sum_{t=1}^{T} \frac{\partial L}{\partial o_t}\frac{\partial o_t}{\partial W_o} \\ &= \frac{1}{T}\sum_{t=1}^{T} \frac{\partial l}{\partial o_t} h_t^T\end{aligned}$$

Nadalje, za posljednji član niza vrijedi

$$\begin{aligned}\frac{\partial L}{\partial h_T} &= \frac{\partial L}{\partial o_T}\frac{\partial o_T}{\partial h_T} \\ &= W_o^T \frac{\partial L}{\partial o_T}\end{aligned}$$

dok je za ranije članove niza račun malo kompliciraniji

$$\frac{\partial L}{\partial h_t} = \frac{\partial L}{\partial h_{t+1}}\frac{\partial h_{t+1}}{\partial h_t} + \frac{\partial L}{\partial o_t}\frac{\partial o_t}{\partial h_t} \tag{2.5}$$

$$= W_h^T \frac{\partial L}{\partial h_{t+1}} + W_o^T \frac{\partial L}{\partial o_t} \tag{2.6}$$

Ako "odmotamo" (2.4) do trenutka T, dobivamo

$$\frac{\partial L}{\partial h_t} = \sum_{i=t}^{T} (W_h^T)^{T-i} W_o^T \frac{\partial L}{\partial o_{t+T-i}} \tag{2.7}$$

Sada koristeći ovaj izraz lako možemo računati potrebne parcijalne derivacije

$$\begin{aligned}\frac{\partial L}{\partial W_x} &= \sum_{t=1}^{T} \frac{\partial L}{\partial h_t}\frac{\partial h_t}{\partial W_x} \\ &= \sum_{t=1}^{T} \frac{\partial L}{\partial h_t} x_t^T\end{aligned}$$

i

$$\begin{aligned}\frac{\partial L}{\partial W_h} &= \sum_{t=1}^{T} \frac{\partial L}{\partial h_t}\frac{\partial h_t}{\partial W_h} \\ &= \sum_{t=1}^{T} \frac{\partial L}{\partial h_t} h_{t-1}^T\end{aligned}$$

čime smo dobili sve potrebne izraze za računanje gradijenata u propagaciji unatrag kroz vrijeme.

Analizom gornjeg izvoda mogu se uočiti potencijalni problemi kod učenja povratnih neuronskih mreža nad nizovima podataka velike duljine. Jedan od problema je takozvana dugotrajna memorija. Naime, često vrijednost izlaza za neki član niza ovisi o članovima koji su se u nizu pojavili puno ranije. Iako je u teoriji moguće postići da povratna mreža

prepoznaje takve uzorke, u praksi je teško dobiti takav model. Nadalje, iz jednadžbe (2.7) se vidi problem s računanjem gradijenata u povratnim neuronskim mrežama. Naime, u izrazu s desne strane se javljaju potencijalno velike potencije matrice W_h^T. Stoga, ako je najveća svojstvena vrijednost te matrice po apsolutnoj vrijednosti veća od 1, javlja se tzv. problem eksplodirajućih gradijenata gdje vrijednost gradijenta postane prevelika pa gradijentni spust divergira zbog prevelikih koraka. S druge strane, ako je najveća svojstvena vrijednost te matrice po apsolutnoj vrijednosti manja od 1, javlja se tzv. problem nestajućih gradijenata gdje zbog premalih vrijednosti gradijenata proces učenja postaje prespor.

Postoje razni trikovi kojima se pokušava doskočiti ovim problemima. Jedna tehnika je tzv. rezanje gradijenata (eng. gradient clipping) gdje se ograničava maksimalna vrijednost norme gradijenata. Često se koristi i izbacivanje neurona (eng. dropout) gdje se izlaz svakog neurona u mreži s nekom vjerojatnosti zamijeni nulom u svakom prolazu. Korisne su i razne vrste normalizacije izlaza slojeva koje se i inače često koriste u dubokom učenju. Također, ispravan odabir aktivacijske funkcije i faktora učenja je važan za stabilizaciju treninga ovih modela.

Ipak, rješavanje ovih problema je kompliciranо i zahtijeva puno eksperimentiranja. Umjesto takvog načina borbe s običnim povratnim neuronskim mrežama, u praksi se puno češće koriste modificirane verzije povratnih slojeva koji su nastali upravo kao ispravak spomenutih mana. Najčešće korišteni slojevi su LSTM (eng. long short term memory) [3] i GRU (eng. gated recurrent unit) [1]. U nastavku ćemo definirati te slojeve i analizirati njihova svojstva koja ih čine pogodnijima za treniranje od osnovne verzije povratne neuronske mreže.

2.2 Napredne arhitekture povratnih neuronskih mreža

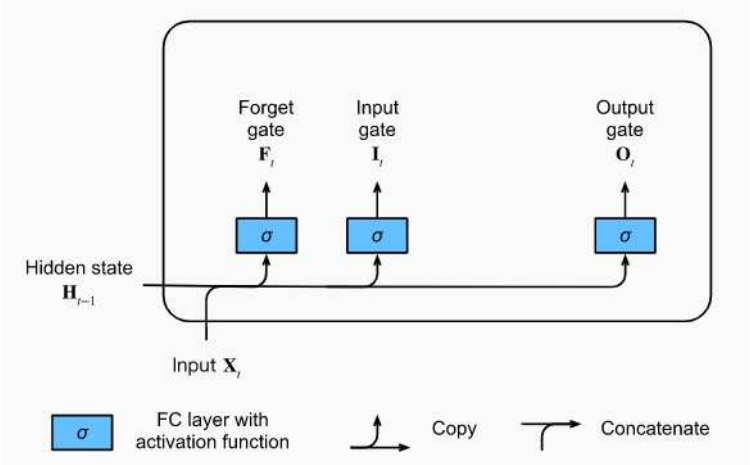
LSTM

Glavni razlog za probleme s modeliranjem dugih nizova pomoću povratnih neuronskih mreža je to što se sve informacije iz ranijih dijelova niza prenose preko jednog skrivenog stanja h_t. Glavna ideja iza LSTM slojeva je zamijeniti obični povratni sloj kompleksnijom strukturom koja sadrži specijalizirano interno stanje koje služi za prijenos informacije preko velikog broja koraka.

Za početak, kao i u običnom povratnom sloju, ulaz LSTM sloja sastoji se od ulaznog podatka x_t i skrivenog stanja prethodnog elementa h_{t-1}. Te dvije vrijednosti prolaze kroz tri različite komponente, a to su ulazna vrata I_t (eng. input gate), izlazna vrata O_t (eng. output gate) i vrata zaboravljanja F_t (eng. forget gate). Ta su vrata definirana s

$$I_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$
$$O_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$$
$$F_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$$

gdje su matrice težina i vektori pristranosti prikladnih dimenzija.



Slika 2.4: Vrata u LSTM sloju

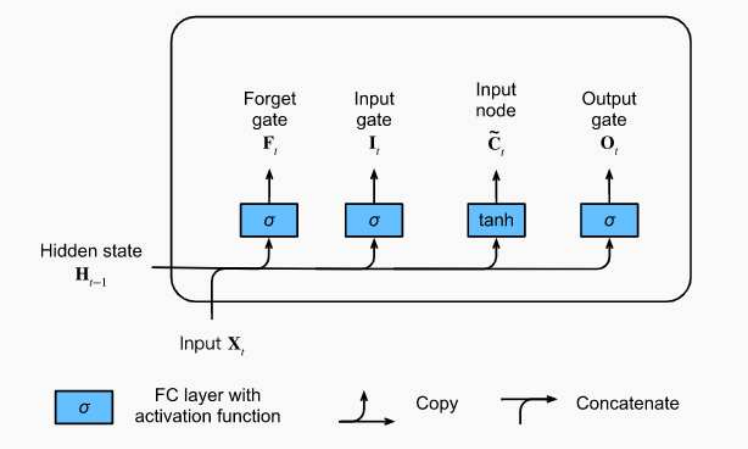
Uz ova tri vrata, dodatno definiramo i ulaznu ćeliju $\tilde{C}_t$ koja je dana s

$$\tilde{C}_t = tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$$

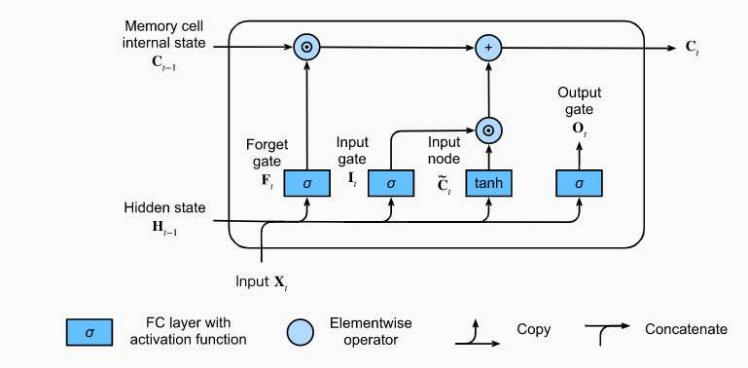
Uz skriveno stanje h_{t-1} i ulazni podatak x_t, LSTM ćelija kao ulaz prima i interno stanje c_{t-1}. Izlaz standardnog povratnog sloja je skriveno stanje h_t, a LSTM dodatno računa i interno stanje c_t. Da bi sloj bio definiran, potrebno je još definirati računanje tih dvaju vrijednosti.

Pri računanju internog stanja c_t, intuitivno shvaćamo prethodno interno stanje c_{t-1} kao dugoročno pamćenje mreže, a ulaznu ćeliju $\tilde{C}_t$ kao nove informacije. Ulazna vrata I_t odlučuju koje će nove informacije sloj pohraniti u interno stanje c_t, dok vrata zaboravljanja F_t kontroliraju koja će se od dugoročnih sjećanja zaboraviti. Preciznije, imamo

$$c_t = F_t \odot c_{t-1} + I_t \odot \tilde{C}_t$$



Slika 2.5: Ulazna ćelija LSTM sloja

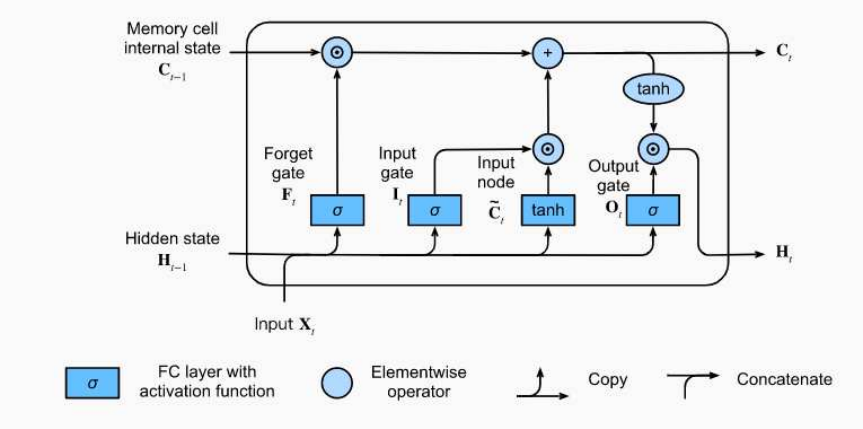


Slika 2.6: Računanje internog stanja LSTM sloja

Kod računanja skrivenog stanja h_t u igru ulaze izlazna vrata O_t koja kontroliraju što iz internog stanja c_t želimo proslijediti kao izlaz sloja. Preciznije, imamo

$$h_t = O_t \odot tanh(c_t).$$

Ovime smo definirali LSTM sloj. Uočimo, ako je vrijednost vrata zaboravljanja F_t velika, a vrijednost ulaznih vrata I_t mala, onda vrijednost internog stanja c_t ostaje gotovo nepromijenjena. Ovo omogućava prijenos informacija između udaljenih članova ulaznog niza te mreži daje svojevrsno dugoročno pamćenje koje olakšava modeliranje dugih nizova podataka. Ovakvim dizajnom sloja uklonjen je problem eksplodirajućih i nestajućih gradijenata što čini LSTM modele puno stabilnijima od običnih povratnih mreža.



Slika 2.7: Računanje skrivenog stanja LSTM sloja

GRU

GRU slojevi su modernija varijanta povratnog sloja koja je temeljena na istim principima kao i LSTM sloj, ali s ciljem poboljšanja efikasnosti korištenjem jednostavnije arhitekture. Umjesto troja vrata, GRU ima samo vrata resetiranja R_t (eng. reset gate) i vrata ažuriranja Z_t (eng. update gate). Intuitivno, R_t služi za zaboravljanje nepotrebnih starih informacija, dok Z_t služi za integraciju novih informacija. Formalno, slično kao kod LSTM sloja, imamo

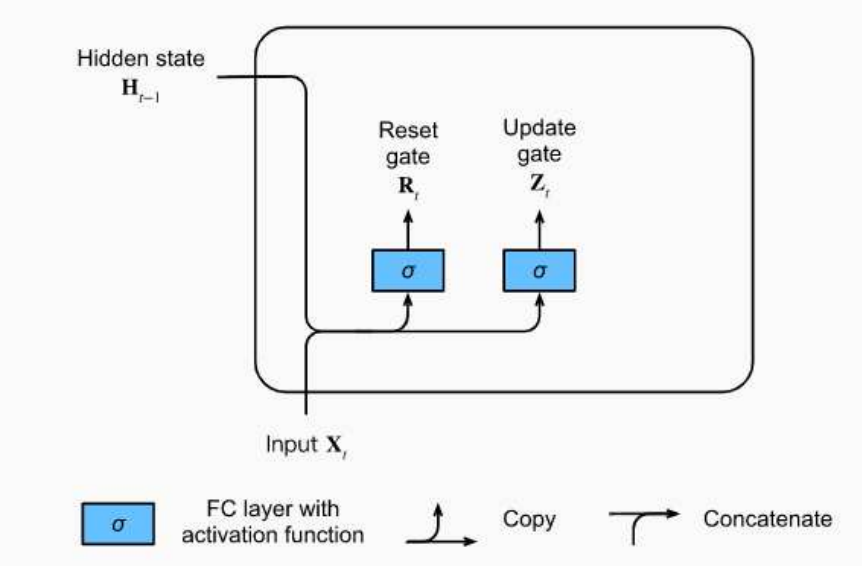
$$R_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$$
$$Z_t = \sigma(W_{xu}x_t + W_{hu}h_{t-1} + b_u).$$

Računanje novog skrivenog stanja h_t radimo u dva koraka. Prvo pomoću vrata R_t dobivamo kandidata za skriveno stanje

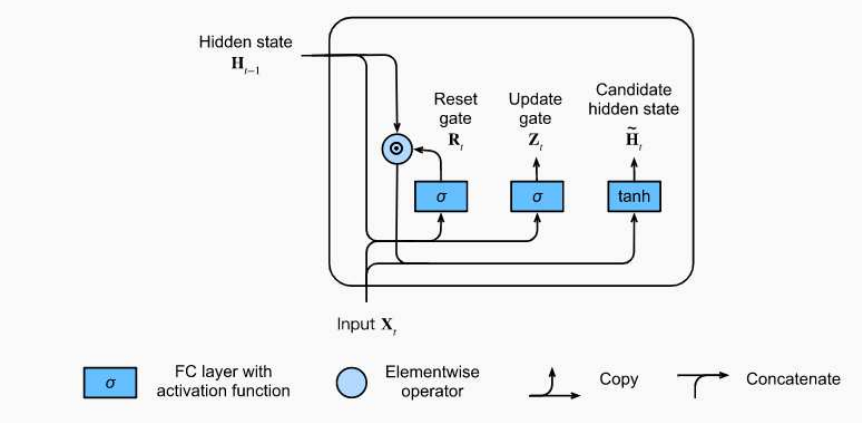
$$\tilde{h}_t = tanh(W_{xh}x_t + W_{hh}(h_{t-1} \odot R_t) + b_h).$$

Uočimo, za $R_t = 1$ imamo klasični RNN sloj, dok za $R_t = 0$ imamo obični unaprijedni neuron. Ovo omogućava efikasno kombiniranje informacija iz dugoročne memorije sa značajkama novog ulaznog podatka. Preostaje primijeniti vrata Z_t kako bismo integrirali nove i stare informacije. Imamo

$$h_t = Z_t \odot h_{t-1} + (1 - Z_t)\tilde{h}_t.$$

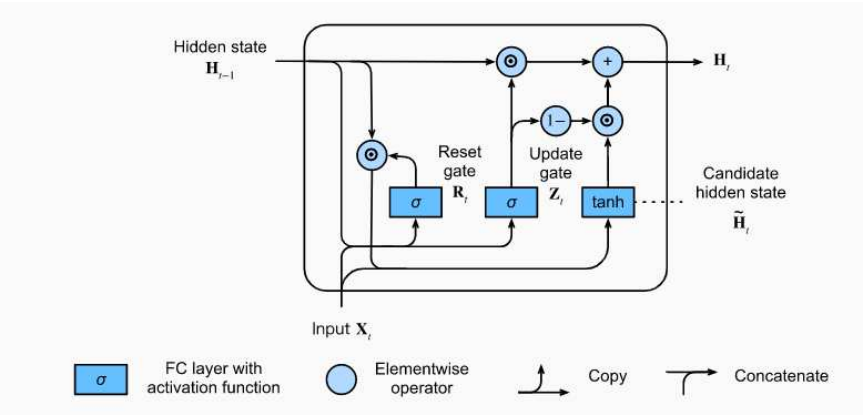


Slika 2.8: Vrata GRU sloja



Slika 2.9: Kandidat za skriveno stanje

Uočimo, za $Z_t = 1$ uzimamo staro prethodno stanje, odnosno u potpunosti izostavljamo informaciju koju donosi novi ulazni podatak. S druge strane, za $Z_t = 0$ novo skriveno stanje je upravo jednako kandidatu za skriveno stanje kojega smo dobili pomoću vrata R_t. Slično kao u LSTM sloju, ovaj mehanizam s vratima R_t i Z_t daje mreži kontrolu nad tokom informacija i omogućava jednostavno pamćenje i zaboravljanje dugoročnih korelacija među podacima.



Slika 2.10: Skriveno stanje GRU sloja

Za LSTM i GRU slojeve smo opisali samo propagaciju podataka unaprijed kroz mrežu. Učenje tih mreža zasniva se na modifikacijama algoritma propagacije unatrag kroz vrijeme koji smo ranije opisali. S obzirom da su ključne ideje iste, a promjene uglavnom tehničke prirode, izostavljamo ih u ovom radu. Ti su algoritmi implementirani u svim popularnim bibliotekama za duboko strojno učenje.

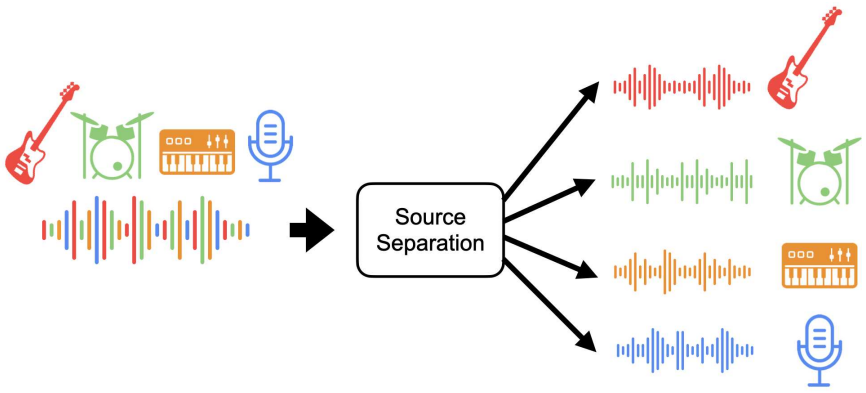
LSTM i GRU slojevi donijeli su rješenje svih velikih problema osnovne arhitekture povratne neuronske mreže te izrazito olakšali treniranje takvih modela. Kombinirajući osnovnu ideju modeliranja nizova te algoritam propagacije unatrag kroz vrijeme za učenje s ovim naprednijim slojevima dobili smo moćan alat za modeliranje nizova podataka proizvoljne duljine koji imaju proizvoljno dugoročne korelacije. U nastavku rada ćemo predstaviti jedan problem za koji su ovakve mreže prikladan alat i dizajnirati neke arhitekture temeljene na ovim mrežama koje će isti uspješno modelirati.

Poglavlje 3

Separacija instrumenata iz glazbe

3.1 Uvod u problem

U ovom poglavlju dajemo kratki uvod u odabranu primjenu povratnih neuronskih mreža, a to je separacija instrumenata iz glazbe. Slike u ovom poglavlju preuzete su s [4]. Glazba koju slušamo u digitalnom obliku nastaje snimanjem zapisa individualnih instrumenata, a zatim spajanjem tih zapisa u jednu audio datoteku koja sadrži čitavu pjesmu. Taj proces je relativno jednostavan za provesti, ali ono što nas zanima je inverzni proces. U situaciji gdje imamo samo audio zapis čitave pjesme, želimo izolirati zapis nekog individualnog instrumenta ili glas.



Slika 3.1: Ilustracija separacije glazbe

Ovaj problem, kao i drugi slični problemi u obradi zvuka (npr. separacija govora), tema je mnogih istraživanja i postoje brojni pristupi koji su korišteni u svrhu njegovog rješavanja. Neki od klasičnih pristupa koriste matrične faktorizacije (posebno nenegativnu

matričnu faktorizaciju) ili skrivene Markovljeve modele (eng. hidden Markov models), ali u novije vrijeme znatan napredak u ovom području donijele su neuronske mreže. I unutar te familije modela postoje razni pristupi, pri čemu su možda najčešće korištene konvolucijske neuronske mreže, ali i povratne neuronske mreže pokazale su se kao koristan alat. One se javljaju kao prirodan izbor s obzirom da je zvuk u fizikalnom smislu zapravo samo val određene valne duljine, a digitalni audio zapis predstavlja samo diskretizaciju vrijednosti tog vala kroz vrijeme.

Slika 3.2: Valni oblik

Takav osnovni zapis zvuka u računalu nazivamo valni oblik (eng. waveform). U suštini, imamo konačan niz vrijednosti neke duljine t. Razlikujemo monofoni zapis gdje se taj niz sastoji od jednog kanala pa ga možemo reprezentirati matricom $x \in \mathbb{R}^{tx1}$ i stereofoni zapis gdje u svakom trenutku imamo mjerenja iz dva kanala pa nam je potrebna matrica $x \in \mathbb{R}^{tx2}$.

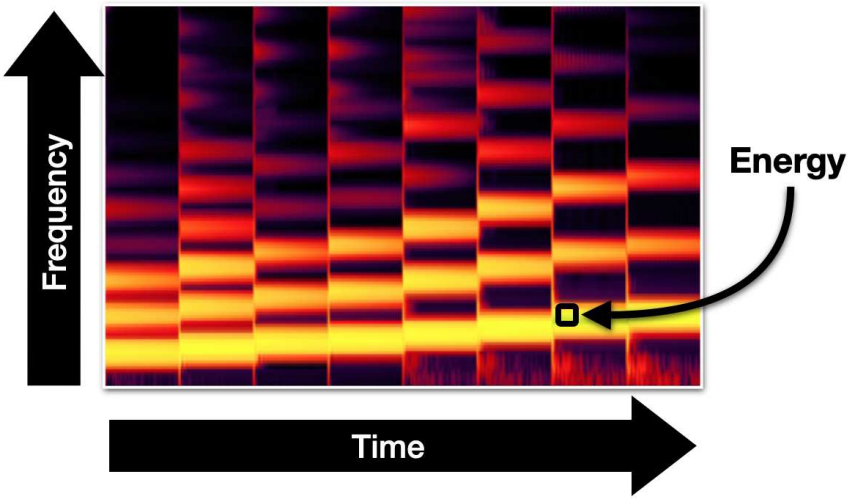
Još jedan važan pojam kod ovakvog zapisa zvuka je frekvencija uzorkovanja (eng. sampling rate) koja nam govori koliko elemenata niza, odnosno redaka matrice, imamo za jednu sekundu zvuka. Za glazbu se najčešće koristi vrijednost $sr = 44100Hz$ što znači da jednu sekundu zvuka predstavlja 44100 vrijednosti. S obzirom da moderne pjesme najčešće traju 3-5 minuta, a ponekad i znatno duže, zvuk u ovakvom obliku teško je modelirati. Smanjivanje frekvencije uzorkovanja također nije poželjno jer poznati Nyquistov teorem kaže da maksimalna frekvencija valova koje možemo reprezentirati odgovara polovici te frekvencije, odnosno imamo

$$f_N = \frac{sr}{2}.$$

S obzirom da su visoke frekvencije bitne za kvalitetu zvuka, u praksi bi smanjivanje frekvencije uzorkovanja rezultiralo neupotrebljivim audio zapisima. Nadalje, čak i kada bismo imali dovoljne računalne resurse za korištenje neuronskih mreža koje mogu raditi nad tako

dugim nizovima, valni oblik nije najprikladniji način za reprezentiranje zvuka za separaciju. Iako ljudsko uho relativno dobro može vršiti taj zadatak, ni ljudi ne mogu inspekcijom valnog oblika dobiti nikakvu informaciju o instrumentima.

Zbog navedenih problema javlja se potreba za prikladnijim načinom prikazivanja zvuka. Odgovor na to pitanje daje nam spektrogram koji predstavlja reprezentaciju zvuka pomoću matrice gdje jedna os odgovara vremenu, a druga os frekvenciji. Vizualno spektrogram možemo prikazati kao sliku iz koje se lako može isčitati puno informacija o zvuku koji je na njoj prikazan.



Slika 3.3: Spektrogram

U nastavku ovog poglavlja dajemo kratki uvod u matematički alat koji se koristi za dobivanje spektrograma, a to je dobro poznata Fourierova transformacija. Nakon dobivanja boljeg razumijevanja rada samog algoritma, dat ćemo dodatnu motivaciju za korištenje ovog prikaza za separaciju glazbe.

3.2 Fourierova transformacija i spektrogram

Konstrukcija spektrograma

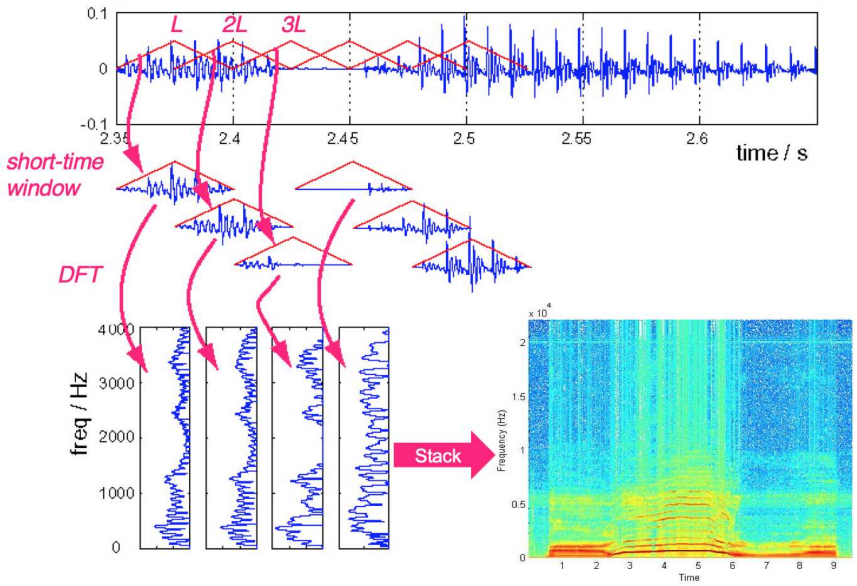
Poznato je da svaku periodičku funkciju koja zadovoljava određene uvjete možemo zapisati kao Fourierov red, odnosno beskonačnu sumu sinusa i kosinusa različitih frekvencija. Ta teorija primjenjiva je i na zvučne valove. Fourierovu transformaciju možemo shvatiti kao funkciju koja za određenu frekvenciju i neki val daje energiju koju taj val ima na

toj frekvenciji, odnosno udio pripadnog sinusa i kosinusa u njegovom Fourierovom redu. Ako frekvencijski raspon diskretiziramo, možemo zvučnom valu $x \in \mathbb{R}^{tx1}$ pridružiti vektor $z \in \mathbb{C}^F$.

Sada ćemo prvo prikazati način na koji se ovakva transformacija koristi za konstrukciju spektrograma. Zatim ćemo formalno definirati Fourierovu transformaciju, njenu diskretiziranu varijantu DFT (diskretna Fourierova transformacija) i algoritam za njeno efikasno računanje FFT (eng. fast Fourier transform).

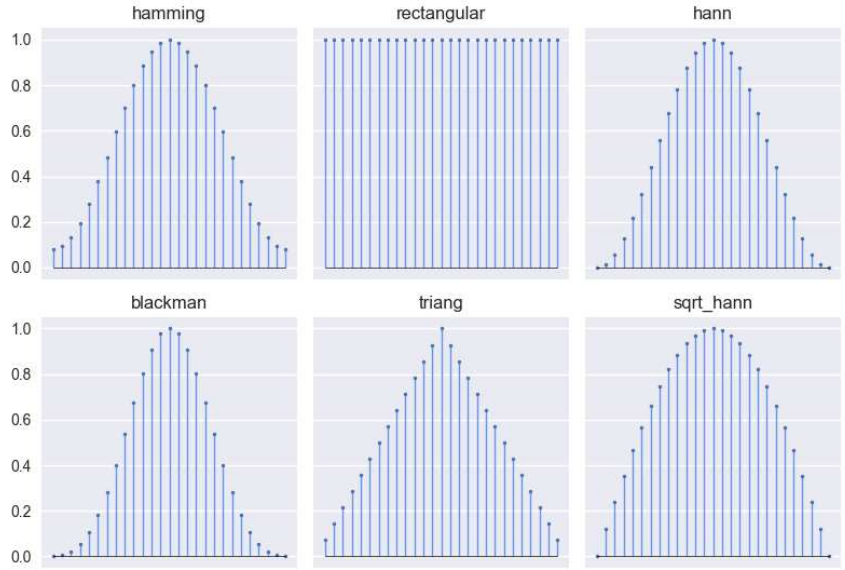
Ako primijenimo DFT na čitavi audio zapis, dobivamo jedan vektor $z \in \mathbb{C}^F$ koji predstavlja čitavi val i iz njega imamo potpunu informaciju o frekvencijama koje tvore taj val. S druge strane, potpuno smo izgubili pojam o vremenu. Zbog fizikalnih svojstava nemoguće je imati reprezentaciju vala koja nam istovremeno daje potpunu informaciju i o vremenskoj i o frekvencijskoj komponenti, a spektrogram predstavlja hibridni prikaz gdje o obje komponente imamo parcijalno znanje.

Spektrogram dobivamo primjenom diskretne Fourierove transformacije na T manjih podsegmenata zvučnog vala. Time za svaki segment dobivamo vektor $z \in \mathbb{C}^F$, a njihovim slaganjem u matricu $Z \in \mathbb{C}^{FxT}$ nastaje spektrogram.



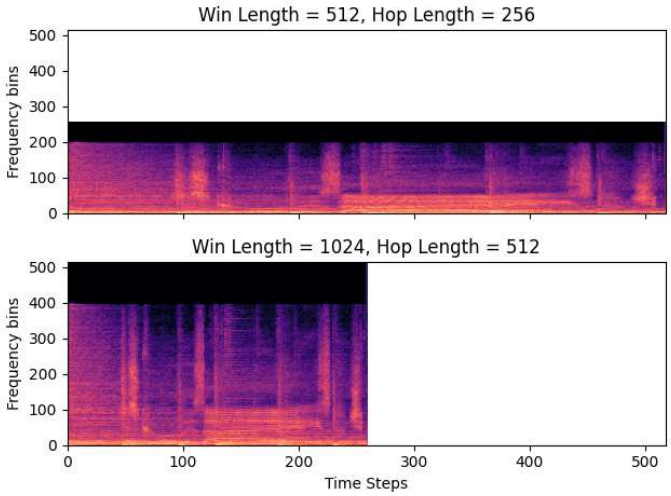
Slika 3.4: Konstrukcija spektrograma

Postoje tri ključna parametra koja određuju oblik spektrograma. Prvi je funkcija prozora (eng. window function) koja predstavlja filter koji se konvolvira sa segmentom audio zapisa prije primjene DFT-a. Oblik tog filtera utječe na izlaz DFT-a i bira se ovisno o frekvencijskim svojstvima pripadnog vala. Preostala dva parametra su duljina prozora i



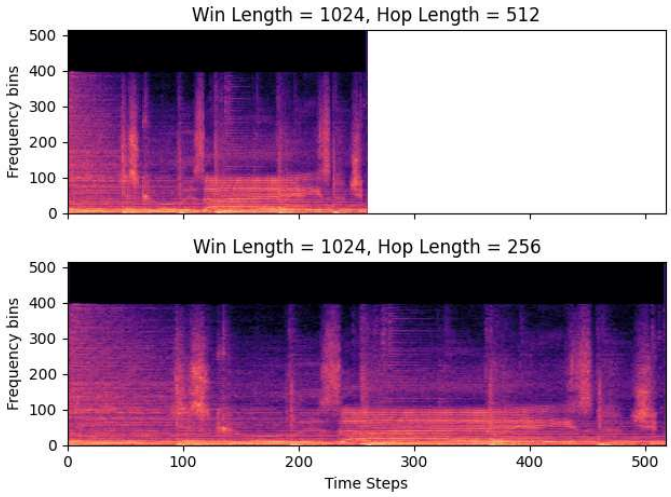
Slika 3.5: Funkcije prozora

duljina koraka. Duljina prozora, kao što i ime sugerira, daje duljinu segmenta koji se predaje DFT-u, dok duljina koraka određuje udaljenost između početnih točaka dva susjedna prozora. Ako su te dvije vrijednosti jednake, dobivamo subdiviziju audio zapisa i prozori se ne presijecaju. U slučaju gdje je duljina koraka veća, neki segmenti audio zapisa se preskaču, a najčešće je duljina koraka manja od duljine prozora što rezultira prozorima koji se presijecaju.



Slika 3.6: Utjecaj duljine prozora na spektrogram

S druge strane, duljina koraka obrnuto je proporcionalna vremenskoj rezoluciji spektrograma. Još je važno spomenuti da je potrebno oprezno odabrati ova tri parametra kako bi



Slika 3.7: Utjecaj duljine koraka na spektrogram

se spektrogram mogao invertirati i savršeno rekonstruirati originalni valni oblik. Za kombinaciju parametara koja zadovoljava ovo svojstvo kažemo da zadovoljava COLA uvjete (eng. constant overlap-add). Ovime smo opisali način konstrukcije spektrograma koristeći

DFT kao crnu kutiju, a sada ćemo definirati tu transformaciju i opisati efikasan način za njenu implementaciju.

Fourierova transformacija

Za apsolutno integrabilnu funkciju f definiramo Fourierovu transformaciju kao funkciju $F : \mathbb{R} \to \mathbb{R}$ danu s

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-i\omega t}\, dt.$$

Uočimo, za opisanu primjenu Fourierove transformacije izrazito je važno svojstvo invertibilnosti, odnosno postojanje jednostavnog načina za dobivanje originalne funkcije iz njene Fourierove tranformacije. Stoga nam je korisno pokazati da za funkciju f i njenu Fourierovu transformaciju F, u slučaju kad je i F apsolutno integrabilna, vrijedi

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega)e^{i\omega t}\, d\omega.$$

Naime, zbog činjenice da je $F \in L^1(\mathbb{R})$ lako se pokaže da vrijedi

$$\begin{aligned} \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega)e^{i\omega t}\, d\omega &= \lim_{\epsilon \to 0+} \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega)e^{-\epsilon\omega^2} e^{i\omega t}\, d\omega \\ &= \lim_{\epsilon \to 0+} \frac{1}{2\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x)e^{-\epsilon\omega^2} e^{i\omega(t-x)}\, dx d\omega. \end{aligned}$$

Odavde zamjenom poretka integracije i korištenjem izraza za Fourierovu transformaciju funkcije gustoće normalne slučajne varijable čiji se dokaz može naći u poglavlju 2.2.2 u [6] slijedi

$$\begin{aligned} \lim_{\epsilon \to 0+} \frac{1}{2\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x)e^{-\epsilon\omega^2} e^{i\omega(t-x)}\, dx d\omega &= \lim_{\epsilon \to 0+} \frac{1}{2\sqrt{\epsilon\pi}} \int_{-\infty}^{\infty} f(x)e^{\frac{-(t-x)^2}{4\epsilon}}\, dx \\ &= \lim_{\epsilon \to 0+} \frac{1}{2\sqrt{\pi}} \int_{-\infty}^{\infty} f(t - 2\sqrt{\epsilon}y)e^{-y^2}\, dy. \end{aligned}$$

Kako je f neprekidna i integrabilna, vrijedi

$$\begin{aligned} \lim_{\epsilon \to 0+} \frac{1}{2\sqrt{\pi}} \int_{-\infty}^{\infty} f(t - 2\sqrt{\epsilon}y)e^{-y^2}\, dy &= \frac{f(t)}{\sqrt{\pi}} \int_{-\infty}^{\infty} e^{-y^2}\, dy \\ &= f(t) \end{aligned}$$

pri čemu koristimo poznati rezultat

$$\int_{-\infty}^{\infty} e^{-y^2}\, dy = \sqrt{\pi}.$$

Ovime smo dobili formulu za invertiranje Fourierove transformacije koja nam omogućuje izradu modela za separaciju glazbe koja radi nad spektrogramima. Jednom kada dobijemo spektrograme koji odgovaraju separiranim izvorima zvuka, korištenjem ove formule, odnosno njenog diskretnog analogona, možemo dobiti separirane izvore u valnom obliku.

U praksi, s obzirom da radimo s diskretiziranim valovima, radit ćemo s diskretnom Fourierovom transformacijom koja je za diskretizirani val $x \in \mathbb{R}^N$ dana s

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i\frac{2\pi}{N}kn}, k = 0, ..., N-1.$$

I za ovu transformaciju htjeli bismo dobiti jednostavnu formulu za invertiranje. Pokazat ćemo da u ovom slučaju vrijedi

$$x_n = \frac{1}{N}\sum_{k=0}^{N-1} X_k e^{i\frac{2\pi}{N}kn}, n = 0, ..., N-1.$$

Prvo pokažimo jednostavnu pomoćnu tvrdnju. Za $n \neq m \in \mathbb{N}$ vrijedi

$$\frac{1}{N}\sum_{k=0}^{N-1} e^{\frac{2\pi ik(n-m)}{N}} = \frac{1}{N}\frac{1-e^{2\pi i(n-m)}}{1-e^{\frac{2\pi i(n-m)}{N}}} = 0$$

dok za $n = m$ vrijedi

$$\frac{1}{N}\sum_{k=0}^{N-1} e^{\frac{2\pi ik(n-m)}{N}} = \frac{1}{N}\sum_{k=0}^{N-1} 1 = 1.$$

Sada računamo

$$\begin{aligned}
\frac{1}{N}\sum_{k=0}^{N-1} X_k e^{i\frac{2\pi}{N}kn} &= \frac{1}{N}\sum_{k=0}^{N-1}(\sum_{m=0}^{N-1} x_m e^{-i\frac{2\pi}{N}km})e^{i\frac{2\pi}{N}kn} \\
&= \frac{1}{N}\sum_{k=0}^{N-1}\sum_{m=0}^{N-1} x_m e^{\frac{2\pi ik(n-m)}{N}} \\
&= \frac{1}{N}\sum_{m=0}^{N-1}\sum_{k=0}^{N-1} x_m e^{\frac{2\pi ik(n-m)}{N}} \\
&= \sum_{m=0}^{N-1} x_m(\frac{1}{N}\sum_{k=0}^{N-1} e^{\frac{2\pi ik(n-m)}{N}}) \\
&= \sum_{m=0}^{N-1} x_m \delta_{n,m} \\
&= x_n
\end{aligned}$$

čime je formula za inverznu diskretnu Fourierovu transformaciju dokazana. Ovo nam daje jednostavan i praktičan način za invertiranje spektrograma koji možemo koristiti u primjeni.

Ipak, za korištenje ovih transformacija u kontekstu treniranja neuronskih mreža bitno je uzeti u obzir i brzinu računanja same transformacije. Tu je od velike koristi algoritam za brzo računanje diskretne Fourierove transformacije FFT. Dok direktna primjena gornjih formula ima složenost $O(N^2)$, taj algoritam radi u složenosti $O(N \log N)$ što donosi znatno ubrzanje.

Glavna ideja ovog algoritma je korištenje simetrije za uštedu u računanju. Naime, iz definicije diskretne Fourierove transformacije imamo

$$\begin{aligned}
X_k &= \sum_{n=0}^{N-1} x_n e^{-i\frac{2\pi}{N}kn} \\
&= \sum_{m=0}^{N/2-1} x_{2m} e^{-i\frac{2\pi}{N}k(2m)} + \sum_{m=0}^{N/2-1} x_{2m+1} e^{-i\frac{2\pi}{N}k(2m+1)} \\
&= \sum_{m=0}^{N/2-1} x_{2m} e^{-i\frac{2\pi}{N/2}km} + e^{-i\frac{2\pi}{N}k}\sum_{m=0}^{N/2-1} x_{2m+1} e^{-i\frac{2\pi}{N/2}km} \\
&= E_k + e^{-i\frac{2\pi}{N}k} O_k.
\end{aligned}$$

Zbog periodičnosti kompleksne eksponencijalne funkcije, izraze E_k i O_k dovoljno je izračunati za $k = 1, ..., \frac{N}{2} - 1$, Naime, vrijedi

$$\begin{aligned}
X_{k+\frac{N}{2}} &= \sum_{m=0}^{N/2-1} x_{2m} e^{-i\frac{2\pi}{N/2}(k+\frac{N}{2})m} + e^{-i\frac{2\pi}{N}(k+\frac{N}{2})} \sum_{m=0}^{N/2-1} x_{2m+1} e^{-i\frac{2\pi}{N/2}(k+\frac{N}{2})m} \\
&= \sum_{m=0}^{N/2-1} x_{2m} e^{-i\frac{2\pi}{N/2}km} e^{-i2\pi m} + e^{-i\frac{2\pi}{N}k} e^{-i\pi} \sum_{m=0}^{N/2-1} x_{2m+1} e^{-i\frac{2\pi}{N/2}km} e^{-i2\pi m} \\
&= \sum_{m=0}^{N/2-1} x_{2m} e^{-i\frac{2\pi}{N/2}km} - e^{-i\frac{2\pi}{N}k} \sum_{m=0}^{N/2-1} x_{2m+1} e^{-i\frac{2\pi}{N/2}km} \\
&= E_k - e^{-i\frac{2\pi}{N}k} O_k
\end{aligned}$$

Ovime smo računanje diskretne Fourierove transformacije sveli na računanje dva izraza koja također predstavljaju istu transformaciju, ali su pripadne sume upola kraće. Dok god je $M = \frac{N}{2}$ djeljiv s 2 možemo nastaviti rekurzivno dijeliti ove sume i lako se vidi da je vremenska složenost ove rekurzije upravo $O(N \log N)$ što rezultira željenim ubrzanjem. Zbog mogućnosti korištenja FFT algoritma se u praksi uglavnom koriste vrijednosti $N = 2^i$ za neki $i \in \mathbb{N}$. Sljedeći pseudokod prikazuje djelovanje FFT algoritma.

function FFT(x)
  $n \leftarrow$ length(x)
  if $n = 1$ **then**
    return x
  end if
  $even \leftarrow$ FFT(even-indexed elements of x)
  $odd \leftarrow$ FFT(odd-indexed elements of x)
  $W_n \leftarrow \exp(-2\pi i/n)$
  for $k = 0$ to $\frac{n}{2} - 1$ **do**
    $t \leftarrow W_n^k odd_k$
    $even_k \leftarrow even_k + t$
    $odd_k \leftarrow even_k - t$
  end for
  return $[even_0, \ldots even_{n/2}, odd_0, \ldots, , odd_{n/2}]$
end function

Ovime smo potpuno opisali proces pretvorbe zvučnog vala u spektrogram i invertiranje te operacije. U narednom poglavlju ćemo definirati arhitekturu neuronske mreže za separaciju glazbe iz spektrograma temeljenu na LSTM slojevima definiranima u drugom poglavlju. Zatim ćemo na konkretnom skupu podataka istrenirati tu mrežu te pomoću nje i algoritma iz ovog poglavlja dobiti model koji radi separaciju glazbe nad zvučnim valovima. Na kraju ćemo prikazati i analizirati dobivene rezultate.

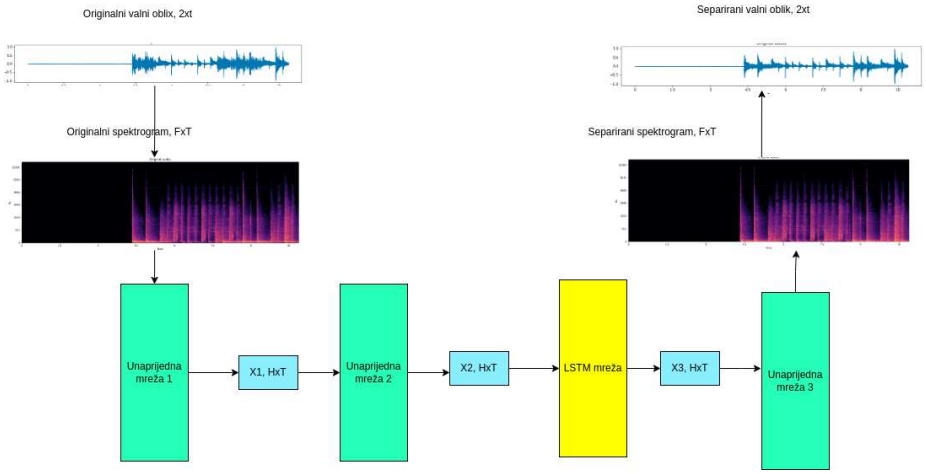
Poglavlje 4

Primjena povratnih neuronskih mreža na separaciju glazbe

U prethodnim poglavljima izgradili smo teoriju potrebnu za konstrukciju neuronske mreže koja rješava problem separacije glazbe. U ovom poglavlju definirat ćemo jednu konkretnu takvu mrežu koja će biti sastavljena od osnovnih elemenata s kojima smo se upoznali u prva dva poglavlja. Ulaz i izlaz te mreže biti će spektrogrami, a teorija iz trećeg poglavlja će nam omogućiti transformaciju audio zapisa u spektrogram i obratno. Prvo ćemo izložiti arhitekturu mreže, zatim predstaviti skup podataka nad kojim ćemo tu mrežu trenirati, opisati konfiguraciju treninga i korištene metode u procesu učenja te definirati metrike koje ćemo koristiti za evaluaciju modela na validacijskom i testnom skupu podataka. Konačno, prikazat ćemo rezultate na testnom skupu podataka i analizirati iste.

Predložena neuronska mreža zapravo se sastoji od četiri različite manje mreže koje djeluju sekvencionalno. Prva mreža je jednostavna unaprijedna neuronska mreža s jednim slojem na koju je dodan sloj za normalizaciju. Sljedeća i najvažnija mreža je LSTM mreža s 3 sloja. Nakon nje slijedi unaprijedna neuronska mreža s jednim slojem na koju je opet dodan sloj za normalizaciju te ReLU aktivacijska funkcija, a zadnja mreža je ponovno unaprijedna sa slojem za normalizaciju.

U ovoj mreži spektrogram $X \in \mathbb{R}^{FxT}$ tretiramo kao niz vektora frekvencija $X_t \in \mathbb{R}^F, t = 1, ..., T$. Unaprijedne mreže svaki od tih vektora procesiraju posebno, dok se LSTM mreža prirodno primjenjuje na niz. Konačni model kao ulaz prima audio zapis u valnom obliku, pretvara ga u spektrogram, provlači ga kroz mrežu kako bi dobio spektrogram separiranog izvora zvuka te invertira taj spektrogram nazad u valni oblik. Arhitektura je prikazana na sljedećem dijagramu.



Slika 4.1: Arhitektura modela

Prvo, model prima valni oblik $x \in \mathbb{R}^{2\times t}$ koji predstavlja pjesmu sa svim instrumentima. Taj se valni oblik transformira u spektrogram $X \in \mathbb{R}^{F\times T}$. Zatim na ranije opisani način taj spektrogram šaljemo u niz od četiri neuronske mreže, pri čemu nastaju međurezultati $X_1, X_2, X_3 \in \mathbb{R}^{H\times T}$, gdje je H dimenzija izlaza svake od mreža. Konačno, zadnja mreža kao izlaz daje spektrogram $Y \in \mathbb{R}^{F\times T}$. Taj se spektrogram zatim invertira u valni oblik $y \in \mathbb{R}^{2\times t}$ koji predstavlja separirani instrument. U eksperimentima koristimo vrijednosti $t = 132300, T = 271, F = 1024, H = 512$. Vrijednost za t predstavlja tri sekunde vremena jer su audio zapisi u skupu podataka koji koristimo uzorkovani na frekvenciji $44100Hz$.

Ovaj model ćemo trenirati i testirati na poznatom MUSDB18 skupu podataka [7] koji je standardni izbor za usporedbu ovakvih modela. Taj se skup podataka sastoji od 150 pjesama kod kojih je zvuk podijeljen u 4 izvora - vokale, bubnjeve, bas i ostalo (svi drugi instrumenti i zvukovi u pjesmi). Stoga ćemo istrenirati tri različita modela, po jedan za separaciju vokala, bubnjeva i basa. Modeli će kao ulaz primati valni oblik dobiven spajanjem sva četiri izvora zvuka te kao izlaz vraćati separirani izvor zvuka. Skup podataka podijeljen je na skup podataka za trening koji sadrži 100 pjesama i skup podataka za testiranje koji sadrži preostalih 50 pjesama. Iz skupa podataka za trening dodatno ćemo izdvojiti dio pjesama od kojih ćemo dobiti skup podataka za validaciju.

Model ćemo implementirati koristeći programski jezik Python i pripadnu biblioteku za duboko strojno učenje PyTorch [5]. U sklopu te biblioteke implementirane su i funkcije za računanje i invertiranje spektrograma, a dodatnu bibilioteku TorchAudio koristit ćemo za dohvaćanje podataka iz MUSDB skupa podataka i njihovu obradu. Metrike za evaluaciju modela, koje ćemo kasnije definirati, računat ćemo koristeći biblioteke Torchme-

trics i fast_bss_eval [9]. Kod za treniranje i evaluaciju modela nalazi se na Github stranici `https://github.com/StipeKabic/RNN_master_thesis`, a priložen je i na CD-u uz diplomski rad.

Za evaluaciju koristimo standardne metrike u ovom području, a to su SNR (signal to noise ratio), SDR (signal to distortion ratio) te SAR (signal to artifact ratio) [10]. Ove metrike detektiraju različite vrste grešaka koje mogu nastati prilikom separacije. Za stvarnu vrijednost izvora zvuka $s \in \mathbb{R}^t$ i procijenjenu vrijednost izvora zvuka $\hat{s} \in \mathbb{R}^t$ SNR se definira kao

$$SNR = 10log_{10}(\frac{||s||^2}{||s - \hat{s}||^2}).$$

Ostale dvije metrike temelje se na dekompoziciji procijenjenog izvora zvuka $\hat{s_j} \in \mathbb{R}^t$ danoj s

$$\hat{s}_j = s_{target} + e_{interf} + e_{noise} + e_{artif}$$

gdje je s_j stvarna vrijednost, a $e_{interf}, e_{noise}, e_{artif}$ redom greške nastale zbog interferencije s drugim izvorima zvuka, pozadinskog šuma i umjetno nastalih zvukova, dok je s_{target} modificirana verzija procijenjenog izvora zvuka. Ova dekompozicija dobije se na sljedeći način. Prvo, sa $\Pi\{y_1, ..., y_n\}$ označimo ortogonalni projektor na potprostor generiran vektorima $y_1, ..., y_n$. Općenito, neka je dan zvučni zapis s dobiven kao zbroj izvora zvuka $s_1, ..., s_n$ (u našem su slučaju to različiti instrumenti) te izvora šuma $n_1, ..., n_m$. Definiramo tri ortogonalna projektora

$$\begin{aligned} P_{s_j} &= \Pi\{s_j\} \\ P_s &= \Pi\{s_1, ..., s_n\} \\ P_{s,n} &= \Pi\{s_1, ..., s_n, n_1, ..., n_m\} \end{aligned}$$

i pomoću njih definiramo izvore grešaka kao

$$\begin{aligned} s_{target} &= P_{s_j}\hat{s}_j \\ e_{interf} &= P_s\hat{s}_j - P_{s_j}\hat{s}_j \\ e_{noise} &= P_{s,n}\hat{s}_j - P_s\hat{s}_j \\ e_{artif} &= \hat{s}_j - P_{s,n}\hat{s}_j. \end{aligned}$$

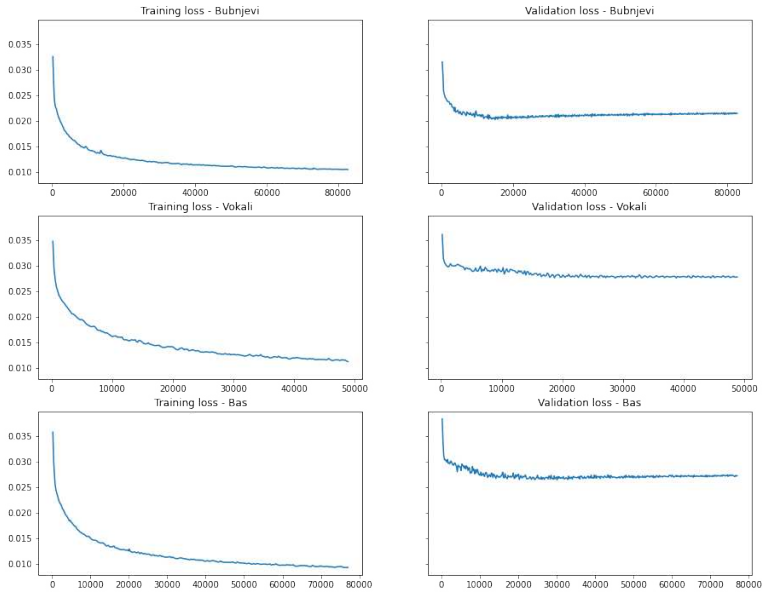
Tada su vrijednosti metrika dane s

$$\begin{aligned} SAR &= 10log_{10}\left(\frac{||s + e_{interf} + e_{noise}||^2}{||e_{artif}||^2}\right) \\ SDR &= 10log_{10}\left(\frac{||s||^2}{||e_{interf} + e_{noise} + e_{artif}||^2}\right). \end{aligned}$$

Što su vrijednosti ovih metrika veće, to je procjena stvarnog izvora zvuka bolja. Nakon što treniranje završi izračunat ćemo ih na testnom skupu podataka.

Za svaki od tri modela koja treniramo koristimo istu konfiguraciju treninga. Za funkciju gubitka koristimo L1 normu primijenjenu na valni oblik, a učenje provodimo pomoću Adam algoritma s faktorom učenja 0.0003, dok za ostale parametre tog algoritma uzimamo standardne vrijednosti navedene ranije u radu. Audio zapise dijelimo na segmente od 6 sekundi te ih kroz mrežu provlačimo segment po segment jer bi obrađivanje čitave pjesme bilo računalno preskupo. Kroz mrežu podaci prolaze u podgrupama od 64 elementa (batch size = 64). Veličina prozora za spektrogram je 4096, a duljina koraka je 512. Modele treniramo 500 epoha, gdje jedna epoha znači jedan prolazak kroz skup podataka za trening. Za treniranje koristimo NVIDIA RTX 3090 grafičku karticu na kojoj trening traje približno 24 sata.

Za svaki od modela tijekom treninga pratimo vrijednosti funkcije gubitka na skupovima podataka za trening i validaciju.



Slika 4.2: Vrijednosti funkcije gubitka kroz trening na skupovima podataka za treniranje i validaciju za sva tri instrumenta. Vrijednosti se računaju na kraju svake epohe. Na početku treninga vrijednost funkcije gubitka je oko 0.08.

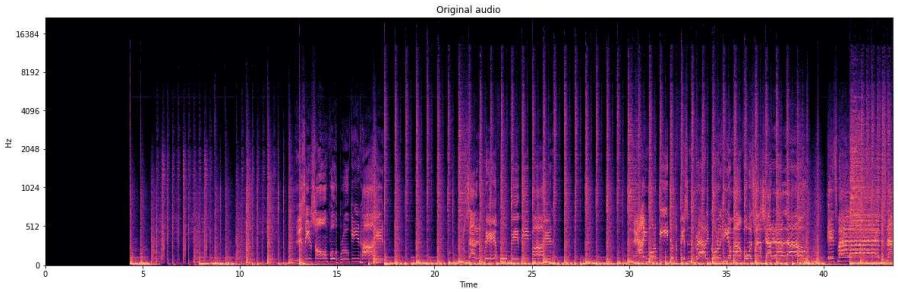
Uočavamo da funkcija gubitka na skupu podataka za treniranje nastavlja padati, dok se na skupu podataka za validaciju zaustavlja i naposlijetku počinje blago rasti. Ovo je klasična pojava prenaučenosti modela, gdje se zbog premalog skupa podataka za treniranje model uspije previše prilagoditi tom skupu i naučiti neke značajke tog skupa koje se ne prenose na skup podataka za validaciju. S obzirom na to, zaključujemo da je modele dovoljno trenirati i puno kraće nego što je provedeno u ovom eksperimentu. Postoje razni pristupi rješavanju problema prenaučenosti koji se uglavnom temelje na umjetnom proširivanju skupa podataka (augmentacija podataka) ili modifikacijama modela. S obzirom na to da je fokus rada arhitektura neuronske mreže i matematičko modeliranje problema, ne primjenjujemo takve metode. Nakon završetka treninga spremamo konačni model i evaluiramo ga na skupu podataka za testiranje, pri čemu računamo vrijednosti gore definiranih metrika SDR, SNR i SAR.

	SDR	SNR	SAR
bas	5.23	5.31	5.16
bubnjevi	5.33	5.52	5.21
vokali	5.84	5.71	5.47

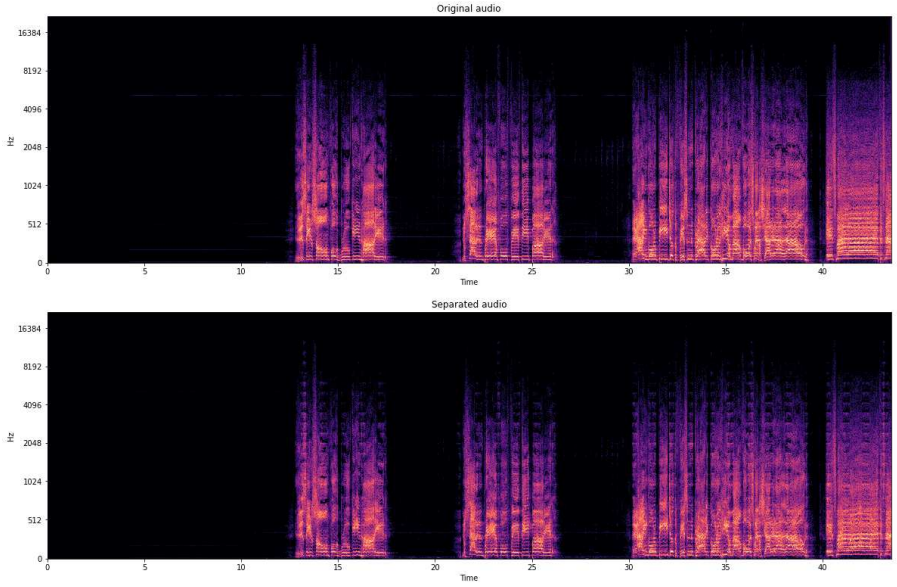
Trenutno najbolji javno objavljeni model za separaciju glazbe koji je dan u [8] za bas, bubnjeve i vokale postiže vrijednosti metrike SDR redom: 9.78, 10.08, 9.20. Taj model koristi dodatne podatke za trening te moderniju arhitekturu. Usporedbe radi, starija verzija istog modela, dana na [2], postiže rezultate za SDR redom: 8.76, 8.24, 8.13. Vrijednosti SNR i SAR metrika se uglavnom ne objavljuju u radovima u ovom području pa tu nemamo ogledne primjere. Dakle, dobiveni rezultati su nešto lošiji od najboljih modela razvijenih za ovaj problem koji koriste značajno naprednije i kompleksnije arhitekture mreža te bolje načine obrade podataka. Ipak, ovaj jednostavan model temeljen na LSTM mreži daje upotrebljive rezultate, a moguće ga je istrenirati brzo i korištenjem relativno malo resursa, što ga ipak čini korisnim.

Osim samih metrika, za vizualizaciju kvalitete rada modela možemo se poslužiti i spektrogramima. Na sljedećoj slici prikazan je spektrogram jednog segmenta jedne pjesme iz skupa podataka za testiranje. Zatim slijede tri slike na kojima je gore prikazan stvarni spektrogram instrumenta, a ispod se nalazi izlaz našeg modela. Te slike redom prikazuju vokale, bubnjeve i bas.

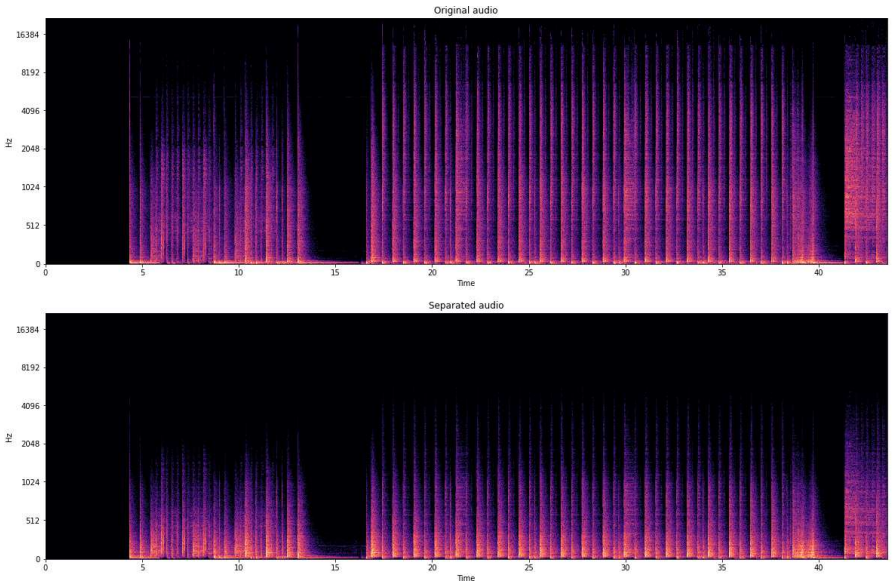
Uočavamo da model prilično dobro separira instrumente, no separirani vokali su blago oštećeni dok su u separiranim bubnjevima izostavljene visoke frekvencije.



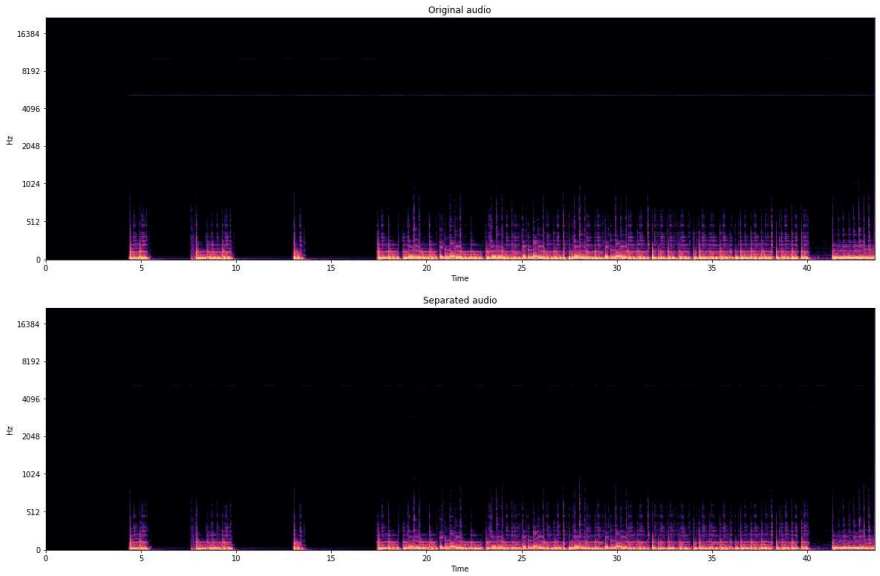
Slika 4.3: Spektrogram segmenta pjesme



Slika 4.4: Stvarni i separirani spektrogram vokala



Slika 4.5: Stvarni i separirani spektrogram bubnjeva



Slika 4.6: Stvarni i separirani spektrogram basa

Bibliografija

[1] KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau i Yoshua Bengio, *On the Properties of Neural Machine Translation: Encoder-Decoder Approaches*, CoRR **abs/1409.1259** (2014), http://arxiv.org/abs/1409.1259.

[2] Alexandre Défossez, *Hybrid Spectrogram and Waveform Source Separation*, 2021, https://arxiv.org/abs/2111.03600.

[3] Sepp Hochreiter i Jürgen Schmidhuber, *Long short-term memory*, Neural computation **9** (1997), br. 8, 1735–1780.

[4] Ethan Manilow, Prem Seetharman i Justin Salamon, *Open Source Tools & Data for Music Source Separation*, https://source-separation.github.io/tutorial, 2020, https://source-separation.github.io/tutorial, posjećeno u veljači 2023.

[5] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai i Soumith Chintala, *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, str. 8024–8035, http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[6] Stanford University Prof. Brad Osgood, Electrical Engineering Department, *The Fourier Transform and its Applications*, https://see.stanford.edu/materials/lsoftaee261/book-fall-07.pdf, preuzeto 6.2.2023.

[7] Zafar Rafii, Antoine Liutkus, Fabian Robert Stöter, Stylianos Ioannis Mimilakis i Rachel Bittner, *MUSDB18-HQ - an uncompressed version of MUSDB18*, kolovoz 2019, https://doi.org/10.5281/zenodo.3338373.

[8] Simon Rouard, Francisco Massa i Alexandre Défossez, *Hybrid Transformers for Music Source Separation*, 2022, https://arxiv.org/abs/2211.08553.

[9] Robin Scheibler, *SDR — Medium Rare with Fast Computations*, 2021.

[10] E. Vincent, R. Gribonval i C. Fevotte, *Performance measurement in blind audio source separation*, IEEE Transactions on Audio, Speech, and Language Processing **14** (2006), br. 4, 1462–1469.

[11] Aston Zhang, Zachary C. Lipton, Mu Li i Alexander J. Smola, *Dive into Deep Learning*, arXiv preprint arXiv:2106.11342 (2021).

Sažetak

U ovom radu dali smo osnovni teorijski uvod u neuronske mreže i motivirali njihovo korištenje. Nakon toga smo precizno izveli algoritam za njihovo treniranje nad podacima. Zatim smo detaljnije obradili povratne neuronske mreže, analizirali glavne probleme koji se javljaju prilikom njihovog treniranja te predstavili naprednije verzije takvih mreža koje rješavaju te probleme, a to su LSTM i GRU mreže. Nakon toga, predstavili smo problem separacije instrumenata iz glazbe te ga matematički opisali. Nadalje, prikazali smo osnovne rezultate o Fourierovim transformacijama koji su nam omogućili jednostavnije modeliranje tog problema. Također smo izveli algoritam za brzo računanje diskretne Fourierove transformacije koji nam je olakšao praktičnu primjenu iste. Ti su nas rezultati doveli do pojma spektrograma koji predstavlja prikladan način zapisa zvučnih podataka s ciljem primjene neuronskih mreža na njih. Konačno, predložili smo arhitekturu neuronske mreže koja je temeljena na povratnim mrežama i primijenili ju na problem separacije. Na samom kraju prikazali smo rezultate ovog modela i ukratko ih analizirali.

Summary

In this thesis we gave a basic theoretical introduction to neural networks and motivated their practical use. After that, we precisely worked out an alogorithm for training these networks on data. Next, we analyzed recurrent neural networks in more detail, worked out the main problems with their training and introduced more advanced versions of these networks which are the LSTM and GRU networks. Afterwards, we introduced the problem of music source separation and described it mathematically. We introduced basic results about Fourier transformations which allowed us to model this problem more easily. We also derived an algorithm for faster calculation of the discrete Fourier transform which made practical use of this transformation easier. These results led us to the definition of the spectrogram as an appropriate way of describing audio data with the goal of training neural networks on it. Finally, we defined a neural network architecture based on recurrent networks and applied it to the problem of music source separation. At the very end, we described the results of this model and briefly analyzed them.

Životopis

Rođen sam u Kninu 23.6.1998. Tamo sam završio Osnovnu školu Domovinske zahvalnosti te jezičnu gimnaziju u Srednjoj školi Lovre Montija. Prilikom izbora studija ipak je prevladao interes za matematiku pa sam 2017. godine upisao preddiplomski studij matematike na Prirodoslovno matematičkom fakultetu u Zagrebu. Taj studij sam završio 2020. godine i odmah nakon upisao diplomski studij matematičke statistike na istom fakultetu. Tijekom studija sudjelovao sam na dva Lumen Data Science natjecanja s kolegama sa studija te osvojio drugo i prvo mjesto, a nakon upisa diplomskog studija počeo sam raditi kao inženjer za strojno učenje i istraživač u domaćoj tvrtci Atomic Intelligence d.o.o u kojoj sam još uvijek. Ta iskustva motivirala su me u izboru teme ovog diplomskog rada, a znanje matematike sa studija pokazalo se kao dragocjen alat u karijeri.