Universitatea POLITEHNICA din București
Facultatea de Electronică, Telecomunicații și Tehnologia Informației

Sistem de transcriere automată a textului în vorbire pentru limba Română folosind tehnici de învățare adâncă

Lucrare de licență

Prezentată ca cerință parțială pentru obținerea
titlului de *Inginer*
în domeniul *Electronică, Telecomunicații și Tehnologia Informației*
programul de studii *Electronică Aplicată*

Conducător științific

Absolvent
Sergiu-Cristian Ionescu

Anul 2021

Universitatea "Politehnica" din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Program de studiu **ELA**

Anexa 1

TEMA PROIECTULUI DE DIPLOMĂ

a studentului IONESCU O.M. Sergiu- Cristian , 442B-ELA

1. Titlul temei: Sistem de transcriere automată a textului în vorbire pentru limba Română folosind tehnici de învăţare adâncă

2. Descrierea temei şi a contribuţiei personale a studentului (în afara părţii de documentare):
Proiectul presupune implementarea şi compararea mai multor arhitecturi de sistem de transcriere automată a textului ce folosesc tehnici de învăţare adâncă (Tacotron, DeepVoice3), urmată de propunerea propriei arhitecturi pentru texte în limba română. În acest scop se va dezvolta o bază de date proprie, formată din 4 ore de înregistrări de texte citite într-un mediu izolat fonic. Fiecare înregistrare va avea o durată cuprinsă între 5 si 15 secunde. Textele citite vor fi compuse din cărţi pentru copii şi articole de ziar, deoarece conţin toate propoziţiile fundamentale ale limbii române. După creearea bazei de date, înregistrările vor fi procesate, pentru eliminarea zgomotului ce poate apărea în semnal. Textul filtrat va fi normalizat (toate literele mari vor deveni litere mici, cifrele vor trebui transcrise cu litere etc). Fiecare înregistrare va avea asociată un text compus din una sau două propoziţii. Textul va fi prelucrat la nivel de caractere ce vor avea asociate un vector (conversie text → char2vec). Acesta va reprezenta intrarea în reţea. Reţeaua se va baza pe arhitectura Sistemelor de Traducere, compusă din 2 părţi: un codor şi un decodor. Codorul va fi responsabil cu codarea textului dat ca intrare în reţea, realizând într-o formă vectorială dependenţa analitică a literelor între ele (depedency parcing). Ca şi tehnică prinicipală, codarea se realizează utilizând reţele convoluţionale. Decodorul se implementează utilizând reţele neurale recurente, primind la intrare Spectogramele Mel ale înregistrărilor. Reţeaua va învăţa deci, să creeze o spectogramă întreagă Mel bazată pe codarea textului de intrare dată de prima reţea. La ieşire, spectograma Mel rezultată va fi convertită în Voce folosind algoritmul Griffin-Lim. De asemenea o altă tehnică ce va fi folosită pentru îmbunătăţirea rezultatelor va fi Învăţarea cu Atenţie. Sistemul va fi implementat în limbajul Python cu ajutorul librăriilor RegEx şi Tensorflow.

3. Discipline necesare pt. proiect:
DEPI, SS, RNSF

4. Data înregistrării temei: 2020-11-24 15:38:57

Conducător(i) lucrare,
As. drd. Ing. Ana-Antonia NEACŞU

Prof. Corneliu Burileanu

Student,
IONESCU O.M. Sergiu- Cristian

Director departament,
Ş.L. dr. ing Bogdan FLOREA

Decan,
Prof. dr. ing. Mihnea UDREA

Cod Validare: **4635d60206**

Anexa 5

Declarație de onestitate academică

Prin prezenta declar că lucrarea cu titlul *"Sistem de monitorizare a serelor"*, prezentată în cadrul Facultății de Electronică, Telecomunicații și Tehnologia Informației a Universității "Politehnica" din București ca cerință parțială pentru obținerea titlului de *Inginer* în domeniul *Electronică și Telecomunicații*, programul de studii *Electronică aplicată* este scrisă de mine și nu a mai fost prezentată niciodată la o facultate sau instituție de învățământ superior din țară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referințe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și fac referință la sursă. Reformularea în cuvinte proprii a textelor scrise de către alți autori face referință la sursă. Înțeleg că plagiatul constituie infracțiune și se sancționează conform legilor în vigoare.

Declar că toate rezultatele simulărilor, experimentelor și măsurătorilor pe care le prezint ca fiind făcute de mine, precum și metodele prin care au fost obținute, sunt reale și provin din respectivele simulări, experimente și măsurători. Înțeleg că falsificarea datelor și rezultatelor constituie fraudă și se sancționează conform regulamentelor în vigoare.

București, 30.06.2021

Absolvent *Sergiu-Cristian Ionescu*

(semnătura în original)

Cuprins

Lista figurilor

Lista tabelelor

Lista acronimelor

NLP = Natural Language Processing
RNN = Recurrent Neural Networks
TTS = Text to Speech
G2P = Grapheme to phoneme
GRU = Gate Recurrent Unit
CNN = Convolutional Neural Networks
CBHG = Convolutional Bank, Highway, biderectional GRU Network
RSC = Romanian Speech Corpus
RSS = Romanian Speech Synthesis
seq2seq = sequence to sequence
Adam = Adaptive Moment Estimation
STFT = Short Time Fourier Transform

Capitolul 1
Introducere

Acest proiect presupune realizarea unui sistem de transcriere a textului în limba română utilizând tehnici de învățare adâncă. Pentru proiectarea unui astfel de sistem este necesară înțelegerea limbajului uman și a modului în care acesta funcționează.

Limbajul este un sistem creat de oameni folosit pentru diferite scopuri și adaptat pentru tot felul de obiective. Niciodată nu putem ști în mod absolut ce înseamnă cuvintele pentru o persoană. O formă de exprimare poate jigni pe cineva, pe când pe altcineva poate amuza. Tot ce putem face este să încercăm să înțelegem cum afectează cuvintele oamenii.

Limbajul reprezintă de asemenea, un sistem ce este folosit nu doar cu scop informațional dar și cu scop social. Prin limbaj oamenii comunică, se apropie și învață unii de la ceilalți.

În ultima vreme, calculatoarele fac parte din viața noastră din ce în ce mai mult, devenind niște unelte indispensabile în îndeplinirea multor sarcini/activități. Astfel, comunicarea om-mașină devine o sarcina foarte importantă și trebuie îmbunătățită în permanență. Procesarea naturală a limbii încearcă tocmai acest lucru. Domeniul presupune proiectarea și construirea aplicațiilor și a sistemelor ce interacționează cu omul utilizând limbajul creat de noi.Prin urmare încercăm să punem sub o formă matematică, și ușor de procesat pentru o mașină, modul în care noi comunicăm unii cu ceilalți.

Scopul acestui proiect constă în transcrierea textului din limba română în vorbire pentru a permite comunicarea roboților și în limba noastră maternă. Există multe situații în care o mașină poate ajuta un om prin comunicare. Spre exemplu, pentru copiii cu autism, un robot ce reușește să se facă înțeles copiilor, poate deveni o unealtă importantă de captare a atenției dacă este utilizată corespunzător de către un profesor. Un alt exemplu este pentru persoanele ce nu pot să citească din anumite cauze. Un astfel de sistem ar ajuta la redarea pasajelor de text.

Proiectul își propune realizarea următoarelor obiective :

- *Procesarea textului* - presupune aducerea textului într-o formă ce poate fi prelucrată și înțeleasă ușor de calculator
- *Generarea de Spectrograme* - este modul de reprezentare sonora sub forma de spectrograme a textului procesat
- *Conversia spectrogramelor în semnal vocal* - eșantioanele de spectrogramă, specifice textului, ce sunt generate, trebuie convertite în semnal vocal cât mai aproape de naturalețea discursului uman

Capitolul 2
Noțiuni teoretice

2.1 Lanțul vorbirii

Știinta exprimării este studiul experimental al comunicarii orale, implicând apariția actului vorbirii si perceptia vorbirii, precum si analiza si procesarea semnalului acustic al vorbirii. Se referă la acea parte a comunicării verbale, în care limbajul capătă o formă mai degrabă fizică decât mentală.
Știința comunicării orale își are originile în fonetică, aceasta reprezentând ramura lingvistică ce studiază sunetele produse în cadrul vorbirii. În această definiție, „sunetele" nu se referă doar la zgomote, ci la piesele codului lingvistic, care sunt utilizate pentru a comunica sensul.
Fundamentul științei exprimării este reprezentat de elementele din cadrul lanțului de vorbire, elemente ce fac referire la următoarele aspecte:

- Elementele mesajului pronunțat sunt codificate ca articulații (execuția planificării articulare)
- Procese aeroacustice care generează sunete din articulație (acustica vorbirii)
- Transmiterea sunetului (acustică)
- Audierea sunetului (audiere)
- Interpretarea senzațiilor auditive în ceea ce privește elementele de pronunție (percepția vorbirii)

Să analizăm în detaliu etapele comunicării verbale:

- **Semnificația intenției** → Motivul consacrat al comunicării între oameni este acela că vorbitorul/emițătorul dorește să schimbe starea mentală a receptorului/ascultătorului. Comunicarea începe astfel întotdeauna cu intențiile vorbitorului. Pentru a atinge un anumit scop comunicativ prin limbă, vorbitorul trebuie să traducă această intenție printr-un enunț care are un anumit sens. Condiția este ca semnificația aleasă să aibă efectul dorit asupra ascultătorului, astfel enunțul nu trebuie să fie o codificare directă a intenției vorbitorului. De exemplu, dacă doriți ca partenerul dvs. să vă servească cu o ceașcă de ceai, s-ar putea să spuneți pur și simplu „Mi-e sete". Adesea, o codificare directă a intenției este considerată nepoliticoasă sau poate fi contraproductivă. Dacă aveți nevoie să știți ora, ați putea spune „Spuneți-mi ora", dar s-ar putea să nu fie o strategie la fel de reușită

ca întrebarea „Îmi puteți spune ora?". Când alegem să comunicăm, ascultătorul nu va lua în considerare doar semnificația enunțului, ci se va întreba „de ce s-a exprimat vorbitorul astfel?". În acest fel, ascultătorul recuperează intențiile vorbitorului. Această zonă de studiu lingvistic se numește Pragmatică; se ia în considerare modul în care este utilizat limbajul pentru comunicare și ceea ce poate presupune vorbitorul despre ascultător și invers.

- **Semnificația enunțului** → Având în vedere cerința unei enunțări, care să aibă semnificația dorită, vorbitorul construiește o enumerare adecvată de cuvinte. Înțelesurile unui enunț sunt compuse prin combinarea semnificațiilor cuvintelor componente, folosind Codul Gramaticii. Vorbitorul își poate explora propriul repertoriu de cuvinte disponibile în combinație, folosind regulile de structură a propozițiilor, pentru a obține astfel o enunțare adecvată, care ar codifica semnificația dorită. Domeniile studiului lingvistic, care leagă semnificațiile cuvintelor de semnificațiile enunțate, se numesc *sintaxă* și *semantică*.

- **Enunț → Plan articulator**: Lexicul mental ne permite să cartografiem semnificațiile cuvintelor la pronunția lor (și invers). De exemplu „mamifer mic, cu blană, domestic, carnivor" = / kæt / (eng.). În plus, persoanele educate sunt capabile să cartografieze interacțiunile din lexicul mental prin ortografie: „mamifer mic, cu blană, domestic, carnivor" = „pisică". Adulții educați cunosc zeci de mii de forme de cuvinte. În general, pronunția cuvintelor este arbitrară și nu are legătură cu semnificația lor. Aceasta se numește Cod fonologic. În cadrul oricărei limbi, articulațiile vorbirii sunt utilizate pentru a codifica un număr mic de opțiuni de pronunție distincte. În timp ce articulațiile și sunetele sunt continue în timp și continue în calitate, aceste alegeri de pronunție sunt discrete. Diferențele de alegere schimbă adesea identitatea cuvintelor. De exemplu, cuvântul „pisică" – cat, are trei elemente, fiecare dintre ele putând fi schimbate pentru a crea cuvinte noi: „to pat"–a mângâia, „cot"–pat, „cap"–pălărie. Aceste alegeri sunt adesea descrise în termeni de selecție dintr-un mic inventar de unități numite foneme. Engleza are aproximativ 20 de foneme vocale și aproximativ 20 de sintagme consonante (în funcție de accent și de modul în care le numărați). Un dicționar vă va oferi, de obicei, pronunția unui cuvânt din ortografia sa, în termenii unei secvențe de foneme, e.g. "Pisică" – / k æ t /. Studiul lingvistic al alegerilor de pronunție utilizate într-o limbă se numește *Fonologie*.

- **Plan articulator → Articulare**: Sunetele vorbirii sunt generate de sistemul vocal, cuprinzând sistemul respirator și articulatorii vorbirii. Sistemul respirator furnizează aer la presiuni modeste, care pot fi utilizate pentru a crea surse de sunet. Articulatorii sunt laringele, maxilarul, buzele, limba și bolta palatină. Articulatorii sunt folosiți atât pentru a genera sunete, cât și pentru a modela calitatea sunetului care iese prin buze și nări. Fiecare unitate de pronunție (fonem) este asociată cu una sau mai multe mișcări sau gesturi articulare. Întrucât articulatorii au nevoie de timp pentru a se mișca, gestul articulator pentru un fonem, va depinde de ceea ce tocmai a fost articulat și de ceea ce urmează să fie articulat în continuare. Studiul lingvistic al articulației și sunetului se numește *Fonetică*.

- **Articulare → Sunet**: Mișcările articulatorilor, sincronizate cu fluxul de aer din plămâni, determină generarea și modelarea sunetului. Generarea sunetului este cauzată în principal de vibrațiile pliurilor vocale din laringe sau de turbulențele create atunci când aerul este forțat printr-o construcție îngustă. După ce un sunet a fost generat la un moment dat în tractul vocal, acesta poate fi modificat în continuare la trecerea acestuia din gură,

datorită dimensiunii și formei conductei tractului vocal prin care trece. Sunetul generat este radiat din gură și se deplasează din difuzor în toate direcțiile. Putem capta semnalul sonor și îl putem reprezenta folosind grafice precum o formă de undă, sau o spectrogramă. Studiul relației dintre articulație și sunet se numește *Acustico-Fonetică*.

- **Sunet → Răspuns auditiv**: la ascultător, sunetele vorbirii sunt convertite în activitate neuronală în sistemul auditiv, cuprinzând urechile exterioare și medii, cohleea și căile neuronale către cortexul auditiv. Sunetele vorbirii dau naștere senzațiilor care variază în intensitate, ton și timbru, dar care variază și în timp. Studiul relației dintre aspectele fizice și psihologice ale senzației sonore se numește *Psihoacustică*.

- **Răspuns auditiv → Secvență de cuvinte**: sistemul de percepție a vorbirii este reglat pentru a asculta diferențele de sunet care semnalează diferite categorii de foneme. Acestea sunt doar acele diferente, care sunt importante pentru diferențierea cuvintelor. De exemplu, suntem mult mai pricepuți să înțelegem distincții în propria limbă, decât distincțiile care sunt folosite în altă limbă, dar nu și în a noastră. Având în vedere senzația auditivă de vorbire (și opțional o senzație vizuală), ascultătorul găsește o secvență de cuvinte care explică cel mai bine cauza senzațiilor. Pentru a determina cea mai probabilă secvență de cuvinte, ascultătorul va apela la o arie largă de cunoaștere: a formei probabile a fonemelor și a variației lor tipice în context, cunoașterea vorbitorului și a mediului acustic, cunoașterea cuvintelor în limba respectivă, cunoașterea accentului și a variației dialectale, cunoașterea secvențelor de cuvinte probabile, cunoașterea semnificațiilor probabile de enunț și cunoașterea subiectelor probabile de conversație.

- **Secvență de cuvinte → Înțelesul**: Din secvența de cuvinte, ascultătorul recuperează semnificațiile posibile pentru fiecare cuvânt și semnificațiile posibile pentru enunț. Există întotdeauna multă ambiguitate în aceste interpretări, dar ascultătorul le poate alege pe cele mai probabile având în vedere contextul.

- **Înțeles → Înțelegerea**: În cele din urmă, ascultătorul ghicește scopul comunicativ al înțelesului enunțului - de ce ar fi vorbit vorbitorul astfel? – pentru a înțelege intențiile vorbitorului.
  [2]

2.1.1 Fonetica si fonologia

Un sistem TTS presupune în prima parte o analiză riguroasă a textului. Textul nu trebuie înțeles doar sintactic ci și fonetic (cum se alcătuiesc cuvintele, cum se pronunță și cum depind unele de celelalte). Este deci foarte important să înțelegem și caracteristicile limbii române prin analiza elementelor ce alcătuiesc limba noastră și cum depind unele de celelalte. Pentru aceasta va fi nevoie să discutăm despre fonetica și fonologia limbii române.

Fonetica este știința care studiază sunetele vorbirii, cele mai mici unități ale limbii, din perspectiva fizică (acustică).[3]

Fonologia pe de altă parte studiază sunete din punct de vedere funcțional.[4]

Sunetele reprezintă rădăcinile oricărei limbi. Acestea alcătuiesc silabe, iar silabele se unesc în cuvinte. Este important să înțelegem cum sunt create și cum relaționează între ele sunetele

pentru a pricepe cum este alcătuită o limbă, dar și cum se diferențiază între ele limbile și caracteristicile lor.
O limbă are 3 componente: fonetica, lexicologia (ce reprezintă studiul vocabularului) și gramatica.

Limbile vorbite de comunitățile umane diferă din punct de vedere *fonologic* și nu fonetic. Asta înseamnă că toate limbile folosesc de fapt același set de sunete, diferența apare doar în modul lor de scriere, și în înțelesul lor (analiza din punct de vedere funcțional).

Sunetul reprezintă cea mai mică unitate a limbii și cea mai mică emisie sonoră articulată (cel mai scurt produs al vocii omenești).[5]
Din punct de vedere fonologic, *sunetele sunt grupate în foneme diferite.*

Fonemul este unitatea sonoră care nu poate fi analizată în unități mai mici și succesive, care ajută la diferențierea unui cuvânt de altul. Fonemele sunt grupate în vocale și consoane.

În limba română avem 7 vocale, e.g. *a e i o u ă â* și 22 de consoane

În limba engleză avem 12 vocale și 24 de consoane.

Putem observa astfel ca diferența între limbi apare în modul de grupare al fonemelor. Diferența dintre sunet și fonem, este faptul că fonemul reprezintă o *clasă de sunete* care îndeplinesc un rol identic în vorbire. Diferența este deci la *modul funcțional*, nu la nivel fizic de creare a sunetului.

2.1.1.1 Fonetica

Pentru analiza fonetică vom considera următoarele definiții:

Vocalele sunt elementele sonore ce sunt rostite fără ca aerul să întâlnească obstacol, pe când *consoanele* sunt elementele sonore ce întâlnesc obstacol [6][7].

Vocalele sunt cele care pot forma singure silabe, iar consoanele nu pot forma singure. Din cele 7 vocale, 4 dintre ele (*o, i, e, u,*) pot îndeplini rolul și de semivocale. Acestea nu pot alcătui singure o silaba și pot fi rostite doar împreună cu o vocală.

Este important sa înțelegem diferența între sunetul concret (cel rostit, auzit) și sunetul-tip (fonem). Spre exemplu litera *r* poate fi rostită atât prin vibrarea limbii dar și prin vibrarea omușorului. Aceste sunete sunt diferite în pronunțare și recepție dar nu formează înțelesuri noi (fac parte din aceeași clasă de foneme). Vocalele și consoanele reprezintă toate fonemele/sunetele tip.

Vocalele pot fi rostite în diferite moduri : prin deschiderea gurii, participarea buzelor, participarea limbii. Consoanele sunt rostite cu efort articulatoriu.

Corespondența dintre fonem și literă se face în baza mai multor principii :

- aceeași literă – același sunet (ex: masă ; ban ; telefon)
- aceeași literă – sunete diferite (ex : e (est); semivocală e (bea); semivocală i (ea [ia]))
- aceeași literă – grupuri diferite de sunete (ex: litera *x* notează grupul de sunete *cs* (ax, pix, sufix) sau *gz* (exact, examen))
- un grup de litere – un singur sunet (ex: *ce, ci* notează *č* (ceas [čas]; ciornă [čornă]))
- mai multe litere diferite – același sunet (ex: literele *î* (amărî, încăpere, reîncălzi) și *â* (cârd, mânca, român) redau sunetul *î*)

Grupurile de sunete *ghe, ghi, che chi, ce, ci, ge, gi*: reprezintă o serie de sunete derivate din sunete deja știute. Așadar, acestea fac parte din a 4-a categorie, unde:

- *ce, ci* se notează *č* (ceas [čas]; ciornă [čornă]), deci fac parte din clasa de foneme *c*

- *ge, gi* se notează *ğ* (geam [ğam]; giuvaier [ğuvaier]), deci fac parte din clasa de foneme *g*
- *che, chi* se notează *k'* (cheamă [k'amă]; chiar [k'ar]), deci fac parte din clasa de foneme *k*
- *ghe, ghi* se notează *g'* (gheară [g'ară]; ghiol [g'ol]), deci fac parte din clasa de foneme *g*

2.1.1.2 Fonologia

Fonologia reprezintă o subramură a foneticii ce studiază latura funcțională a sunetelor și stabilește sistemul (inventarul) de foneme al unei limbi și caracterul variațiilor acestora. Fonologia este cea care inventariază unitățile fonologice (adică sunetele tip/foneme) și întocmește sistemul fonologic al limbii române.

Sistemul fonologic reprezintă totalitatea unităților fonologice stabilite printr-o operație de segmentare și relațiile dintre ele. Sistemul acceptat cuprinde 7 vocale (dintre care 4 pot juca rol și de semivocale) și 22 de consoane [4]. Unitățile fonologice pot fi clasificate în:

- unități segmentale
- unități suprasegmentale

Unitațile segmentale contractează relații doar *în cadrul* silabei. **Unitățile suprasegmentale** pot contracta relații de dependență *între* silabe. Pot fi *intensive* (accentul) sau *extensive* (intonația).

Structura fonologică reprezintă tipul de combinații ale unităților sistemului fonologic. Cea mai mică structură fonologică este *silaba*. **Silaba** este alcătuită dintr-un segment obligatoriu vocalic, și un segment consonatic care poate lipsi. Sunetele sunt transpuse în scris cu ajutorul *literelor*, denumite și *grafeme*.

După cum am spus și în subcapitolul anterior fonemul este asociat unui grup de sunete, cu caracteristici comune, ce sunt propii unei anumite limbi. Sunetele prin urmare existau deja, noi doar le-am inventariat printr-o serie de operații. Putem creea o infinitate de sunete. Vorbitorii emit vibrații acustice diferite chiar și pentru același sunet. Prin urmare putem clasifica sunetele în:

- sunete echivalente
- sunete nonechivalente funcțional

Sunetele echivalente sunt acele sunete care indiferent de nuanțele lor, nu schimbă rolul funcțional al cuvântului. **Sunetele non-echivalente** sunt acele sunete care schimbă semnificația cuvântului odată cu nuanțarea/schimbarea pronunției.

Când 2 sunete sunt echivalente funcțional diferențele fizice (acustice) nu prezintă importanță în comunicare; totuși distincția/proprietățile ce le disting se numesc *trăsături nedistinctive*. Atunci când sunetele sunt nonechivalente, diferențele fizice prezintă un impediment în comunicare, și se numesc *trăsături distinctive. Două sau mai multe sunete diferite prin trăsături nedistinctive și echivalente funcțional reprezintă același fonem. Doua sau mai multe sunete diferite prin trăsături distinctive și nonechivalente funcțional reprezintă foneme diferite* [8].

În cele ce urmează, vom introduce noțiunile de diftong, triftong, hiat, accent și legile de interacțiune ale sunetelor în limba română (cum se formează silabele)

Diftongul ca și structură este format din alăturarea unei vocale (V) cu o semivocală (S) în aceeași silabă. Există 2 tipuri de diftongi : S+V și V+S. Toate combinațiile de semivocală și vocală ce formează diftongi pot funcția și ca hiaturi. [9]

Triftongul este o structură formată dintr-o vocală plus două semi-vocale, aflate în aceeași silabă. Există 2 tipuri de triftongi : Ascendenți (SSV) și Descendenți (SVS) [10]

Hiatul prezintă o structură de 2 vocale alăturate, dar care se află în silabe diferite. [11]

Accentul reprezintă rostirea mai energică a unei silabe și poate deosebi înțelesul cuvintelor. Aceasta cade pe oricare dintre silabele cuvintelor și se adaugă deasupra vocalei pronunțate mai intens. Cuvintele fără accent sunt monosilabice precum formele pronumelor. [12]

Silaba este sunetul sau grupul de sunete care cuprinde în mod obligatoriu o vocală și care se pronunță printr-un efort expirator (printr-o singură deschidere a gurii).*Silabația* reprezintă împărțirea în corpuri fonetice ale cuvintelor în silabele lor constituente.
[13] Reguli de despărțire în silabe :

- *Regula hiatului*: 2 vocale succesive se despart în silabe diferite.
- *Regula diftong/triftong* : vocala urmată de un diftong/triftong se desparte înainte
- *Regula consoanei intervocalice*: când o consoană se afla intre 2 vocale, despărțirea se face înaintea consoanei.
- *Regula grupului de consoane intervocalice*: când n consoane ($n \geq 2$), se află între vocale.

2.1.1.3 Concluzii fonetica fonologie

Analiza aceasta ne ajută să înțelegem modul în care sunetele relaționează/interacționează între ele și modul în care sunt create și diferențiate. Dacă știm cum ar trebui să se comporte sunetele, atunci putem face o verificare să vedem cât de apropiată este transcrierea noastră față de cea reală. Mai mult, putem observa cât de bine s-au respectat principiile foneticii și ale fonologiei. Ne va ajuta să înțelegem atât scheletul cât și miezul limbii că să știm cum o putem modela matematic. Aici am făcut o analiză particulară pentru limba română. O analiză generală a unei reprezentări matematice ținând cont de principiile foneticii și ale fonologiei va fi făcută în subcapitolul de procesare naturală a limbajului (*NLP*).

2.2 Metode de analiză a discursului

Analiza discursului se face utilizând 3 metode de vizualizare a unui semnal :

1. Vizualizarea formei de undă
2. Vizualizarea spectrului de frecvență
3. Vizualizarea Spectrogramei

1. Forma de undă
Forma de undă este cea care oferă o vizualizare generală a undei de sunet/discursului. Această oferă informații despre variația amplitudinii semnalului în timp. Aceste variații pot reprezenta caracteristici importante ale discursului, și descriu modul în care sunetul este creat.
Porțiunile de liniște, numite și *closures (eng)* arată :

- o descriere a modului de pronunție/creare a sunetului (închis, aspirat, prevoic etc.)
- o trecere de la un sunet/fonem la altul

Alte caracteristici ale formei de unda care pot apărea sunt intonația și *voicing*-ul (subiectivitatea generata de inflexiunile vocii) sau zgomotul de frecare (ce apare când există o trecere mai mare de la un cuvânt la altul.

2. Spectrul de frecvență :
Este o reprezentare în frecvență a semnalului, oferind informații despre amplitudinea fiecărei componente spectrale de frecvență. Pe axa abscisei sunt reprezentate frecvențele semnalului, iar pe axa ordonatei avem amplitudinile componentelor în frecvență.

3. Spectrogramele:

Reprezintă cea mai importantă reprezentare a unui semnal/discurs. Oferă informație, în primul rând, despre variația frecvenței în timp a semnalului. Prin analiza acestei variații se pot deduce caracteristici importante ale discursului observând formații, variații periodice (*pattern-uri (eng)*) în voce. Astfel putem deduce nu doar ce cuvinte au fost rostite ci și în ce mod și cu ce intonație. Acest lucru face analiza cu spectrogramă cea mai utilă metodă din toate cele 3 metode [14].

2.2.1 Variabile Fonetice

Variabilele fonetice sunt trăsături ale discursului pe care un copil le ascultă și le analizează pentru a înțelege și a învăța o limbă nouă. Prin aceste variabile, sunetele pot fi distinse.
1. Consoane oclusive:
Sunetul este oprit pentru o scurtă perioadă de timp, după care este eliberat brusc. Perioada de timp în care sunetul este oprit se numește *perioada de blocare* (*eng. closure*), urmată de *momentul eliberării* (*release (eng)*).

Pentru acest tip de consoane, copilul nou născut ce învață limba maternă, trebuie să-și dea seama de intervalul de timp în care sunetul este oprit. Folosind metoda analizei cu formă de undă putem deduce cât de tare este sunetul dar și să facem o analiză mai riguroasă a momentelor de *oprire* și de *eliberare*. În funcție de această perioadă, se pot scoate sunete (vocale și consoane) diferite (*daaa, baaa, gaaa*). De asemenea, durata de *blocare* poate determina și sensul unui cuvânt. În engleză, alungirea unui sunet nu schimbă sensul, dar în alte limbi îl poate schimba.

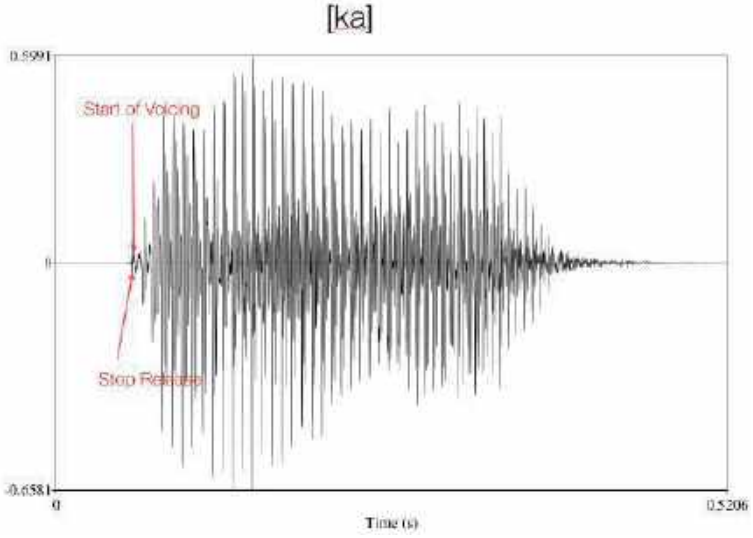

Figura 2.1: Forma de undă a consoanelor oclusive

2. Consoane oclusive aspirate

Pentru acest tip de variabilă sunetul se produce în felul următor:
Crearea sunetului → perioada de blocare → eliberarea sunetului cu o amplitudine mică → perioada în care vocea e eliberată din aparatul vocal și crește amplitudinea (perioada dintre eliberarea sunetului și deschiderea aparatului vocal).

Acest timp se numește *VOT* (*voice onset time (engl)*). În cazul consoanelor oclusive, *VOT* tinde spre 0, iar pentru oclusivele aspirate VOT-ul este > 0. (p h k → aspirate (*pfaa, khaa*)).

În concluzie, în acest caz, eliberarea sunetului este urmată de exprimarea sonoră. Copilul va trebui să ia în considerare, așadar, o nouă trăsătură, anume *VOT*-ul.

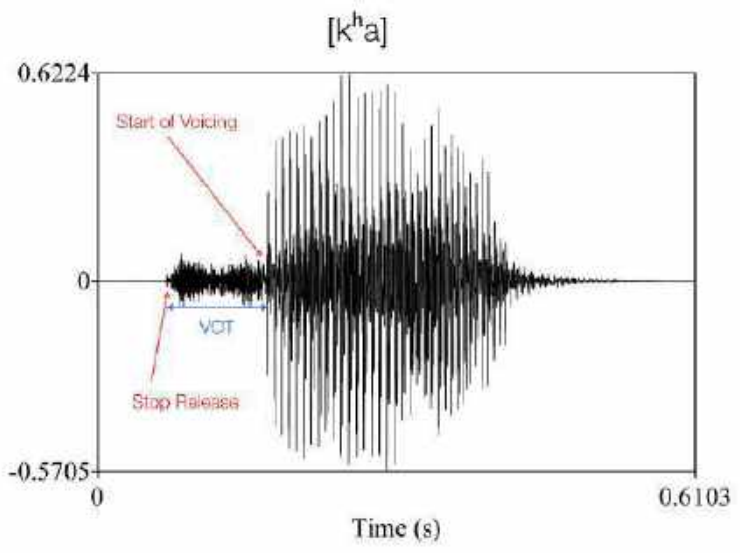

Figura 2.2: Forma de unda a consoanelor oclusive apsirate

3. Consoane oclusive de prevoce (*eng. Prevoiced plosives*)

Pentru acest tip de variabilă sunetul se produce în felul următor: Crearea sunetului → perioada de blocare → exprimarea sonoră → eliberarea sunetului (*eng. release*) În alte cuvinte, putem spune că activarea aparatului vocal precedă eliberarea fluxului de aer. În acest caz, parametrul lingvistic *VOT* are valoare negativă. (de exemplu: *mbaaa*) [15]

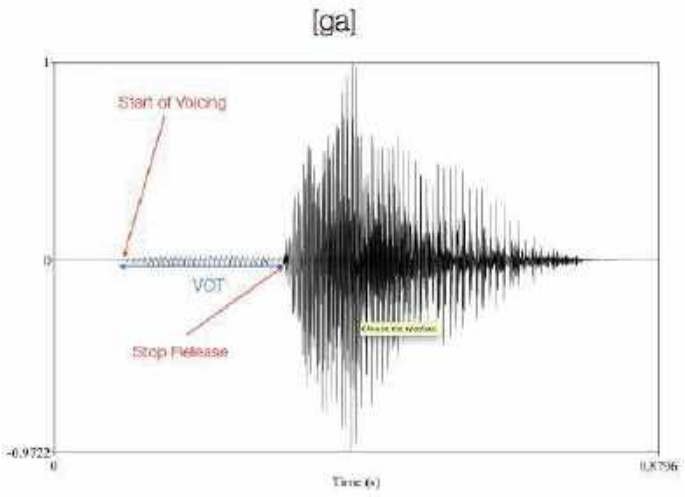

Figura 2.3: Forma de unda a consoanelor oclusive de prevoce

Prin urmare copilul va trebui nu doar sa știe cât timp ține gura închisă dar și când își va

da drumul la aparatul vocal înainte sau după ce își deschide gura.

Frecvențele formante

Pentru fiecare vocală și consoană avem forme diferite ale gurii. Folosind analiza spectrogramelor cel mai ușor se deduc vocalele. Vocalele sunt cele care redau forma aerului dar nu o strică deoarece nu există un obstacol ce poate opri sunetul.

Cum analizam vocalele:

În analiza spectrogramelor pentru vocale observam următoarele reprezentări :

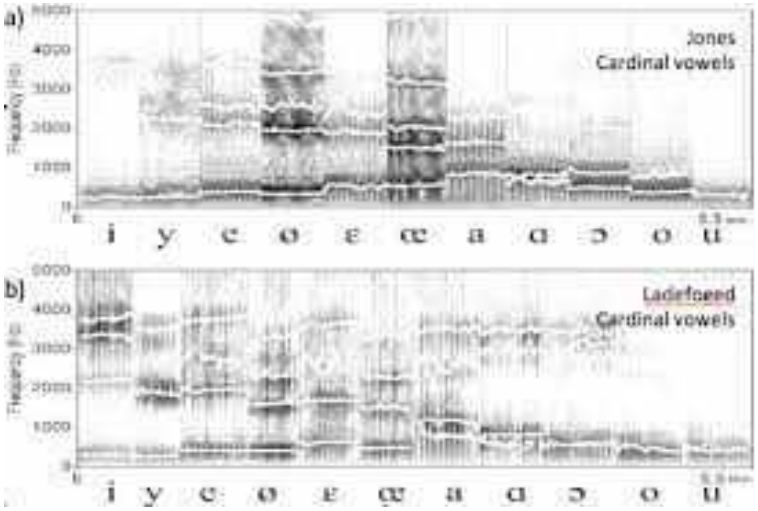

Figura 2.4: Reprezentările în frecvență ale vocalelor

Observăm faptul că zonele mai închise reprezintă o frecvență dominantă. Practic spectrograma ne arată ce *pitch (eng.)* este mai tare la un anumit moment. Fiecare sunet, vocală are frecvențe diferite. Fiecare sunet, este reprezentat de spectrograme pe 3 zone concentrate. Aceste zone reprezintă de fapt frecvențele formante *F1, F2 și F3* (vezi fig.2.5).

Un copil aude toate aceste frecvențe și își dă seama ce sunet este rostit. Cele mai importante frecvențe formante sunt ultimele două. Cea de sus este utilizată să ne dam seama de cum se simte persoana, să identificăm cu cine vorbim, dar nu este folosită pentru diferențierea sunetelor. Prin urmare, pentru analiza și distincția sunetelor vom folosi formanții F1 și F2.

Graficul din fig.2.6 reprezinta zonele de frecventa F1 – F2 pentru fiecare sunet. Copiii învață aceste zone. Cu F1 și F2 în principiu construim sistemul nostru de vocale. Legenda figurii este precum urmează : *celulele galbene* corespund vocalelor din limba română, *celulele crem* indică vocale folosite rar în limba română, *celulele gri* corespund unor vocale rare, fără simbol fonetic. Mai mult, se observă faptul că frecvența maximă a unei vocale este de 2000Hz.

În general o limbă distinge trei sau patru niveluri de deschidere vocalică, foarte puține ajungând la cinci. Limba română distinge trei niveluri de deschidere, de exemplu vocalele din seria [i] - [e] - [a] formează o astfel de scală. În aceeași situație se află limbi precum spaniola, japoneza, greaca și latina [16].

Pentru o analiză a consoanelor ne uităm la variațiile în frecvență, tranzițiile către vocale. În funcție de variația ce are loc înainte de vocală, putem deduce consoana. Așadar, vocala este utilizată ca și referință pentru a găsi consoanele și a ne da seama ce consoană este.

Ca și concluzie a acestui subcapitol putem extrage trăsături pentru semnalele vocale folosind analiza spectrogramelor și a semnalelor în timp [17].

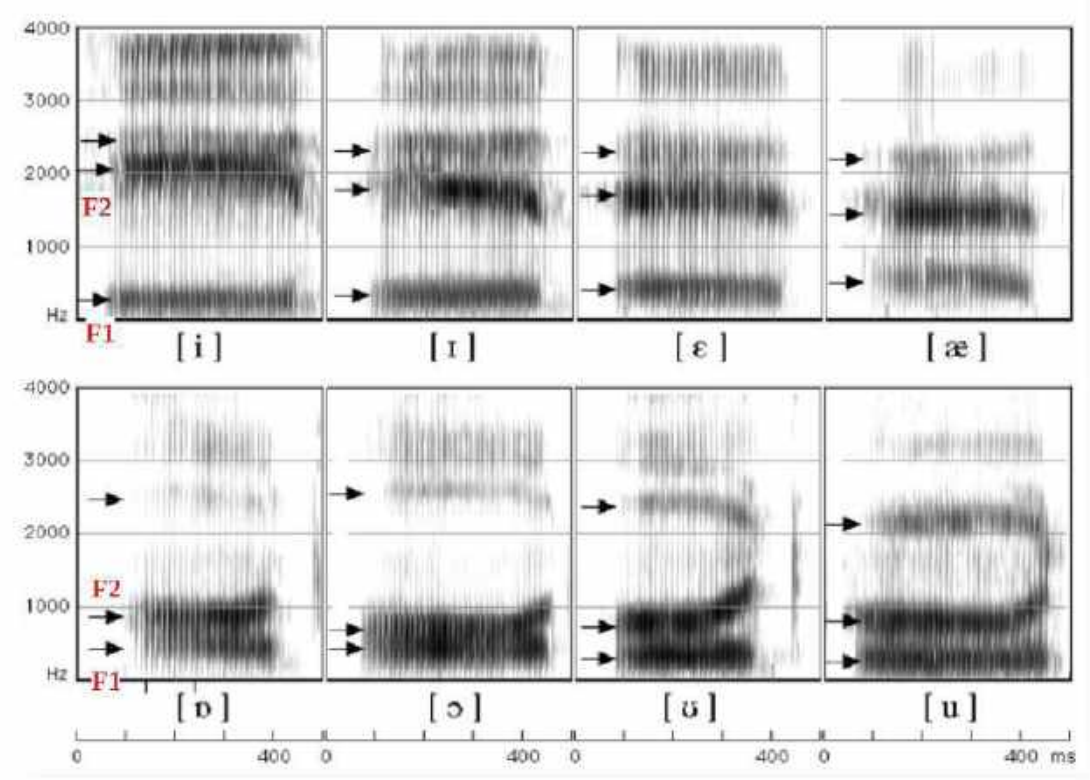

Figura 2.5: Frecvențele formante ale vocalelor

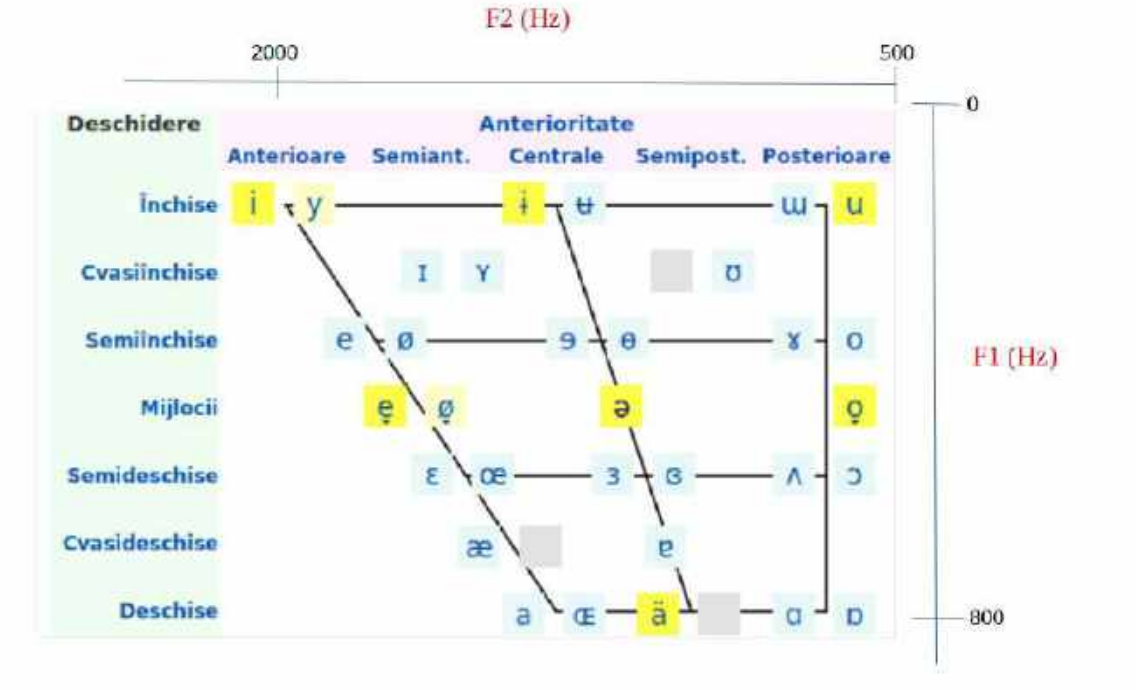

Figura 2.6: Diagrama vocalelor

2.2.2 TFTD

Transformata Fourier Discretă în timp (TFTD) este un instrument matematic de analiză a semnalelor numerice ce face trecerea unei secvențe de timp discretă, neperiodică în spațiul frecvențelor. Spațiul în care este trecută funcția discretă, este un spațiu continuu deci cu un număr infinit de puncte.

Fie $x(n)_{1\leq n\leq N}$ o secvență discretă în timp.

Spunem că $X(\omega)$ este transformata Fourier Discretă în Timp dacă : :

$$X(\omega) = \sum_{n=-\infty}^{\infty} x[n] \cdot e^{-j\omega n} \tag{2.1}$$

unde ω reprezintă frecvența. De asemenea întoarcerea în domeniul timp se face cu formula

$$x[n] = \frac{1}{2\pi} \int_{2\pi} X(\omega) \cdot e^{j\omega n} \tag{2.2}$$

TFTD este prin urmare o transformată bidirecțională.

$$x[n] \leftrightarrow X(\omega) \tag{2.3}$$

TFTD este alcatuită atât dintr-o parte reală cat si din una imaginară.

$$\begin{aligned} X(\omega) &= Re\{X(\omega)\} + jIm\{X(\omega)\} = |X(\omega)|e^{j\cdot\phi} \textit{ unde :} \\ |X(\omega)| &= \sqrt{Re\{X(\omega)\}^2 + Im\{X(\omega)\}^2} \textit{ este modulul transformatei Fourier} \\ \phi &= arctan(\frac{Im\{X(\omega)\}}{Re\{X(\omega)\}}) \textit{ este defazajul} \end{aligned} \tag{2.4}$$

Informația utilă o putem afla afișând atât amplitudinea/magnitudinea componentelor spectrale, cât și defazajul. Graficul magnitudinii indică amplitudinea componentelor în frecvență, iar graficul de defazaj arată defazajul acestor componente față de începutul semnalului în timp [18].

Fie $x[n]$ o secvență discretă în timp, unde

$$x[n] = \delta[n] + \delta[n-1] + \delta[n-2] + \delta[n-3] + \delta[n-4] \tag{2.5}$$

Astfel avem următoarele reprezentări:

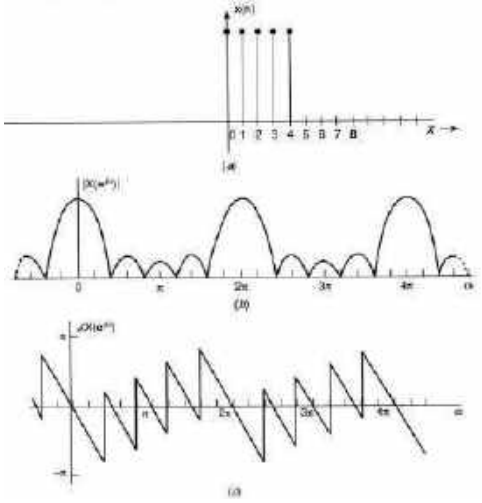

Figura 2.7: Funcția $x[n]$ și reprezentarea sa Fourier

2.2.3 TFD

Transformata Fourier Discretă nu este altceva decât o eșantionare a spațiului de frecvențe rezultat din TFTD. Această transformare este folosită pentru analiza computațională în domeniul frecvenței. Ca și aplicații avem următoarele :

- Analiza spectrală (periodicitatea datelor)
- Compresia datelor
- Filtrare de semnale
- Convoluții calculate rapid

Pentru a putea fi calculată, secvența discretă trebuie să fie una finită. Astfel secvența discretă în timp $x[n]$ va trebui să ia valori pentru n natural de la 0 la $N-1$, pentru a avea un numar N finit de eșantioane [19]. Formula TFDT devine astfel:

$$X(\omega) = \sum_{n=0}^{N-1} x[n] \cdot e^{-j\omega n} \tag{2.6}$$

Ecuația de mai sus pare a fi o relație ce poate fi calculată de către o mașină, însă ω este încă o variabilă continuă, deci poate lua un număr infinit de valori. Prin urmare va trebui să eșantionăm ω. Astfel ω devine :

$$\omega_k = \frac{2\pi}{N} k, k = 0...N-1 \tag{2.7}$$

TFD devine astfel TFDT evaluată la valori discrete de frecvență ω_k :

$$X[k] = X(e^{j \cdot \frac{2\pi}{n} \cdot k}) = \sum_{n=0}^{N-1} x[n] \cdot e^{-j \cdot \frac{2\pi}{N} \cdot kn} \tag{2.8}$$

Inversa acestei transformate va fi în acest caz :

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] \cdot e^{j \cdot (\frac{2\pi}{N}) \cdot k} \tag{2.9}$$

Acum în ambele cazuri avem sume de lungimi finite perfect computaționale. În continuare va fi prezentat un exemplu concret pentru a înțelege mai bine diferența dintre TFDT si TFD. Fie:

$$x[n] = \delta[n] + \delta[n-1] + \delta[n-2] + \delta[n-3] \tag{2.10}$$

vezi fig. 2.8

Calculăm TFDT:

$$\begin{aligned} X(\omega) &= \sum_{n=0}^{3} x[n] \cdot e^{-j\omega n} \\ &= e^{-j \cdot \frac{3 \cdot \omega}{2}} \cdot \frac{sin(2 \cdot \omega)}{sin(\frac{\omega}{2})} \end{aligned} \tag{2.11}$$

Calculăm TFD:

$$\begin{aligned} X[k] &= \sum_{n=0}^{3} 1 \cdot e^{-j \cdot \frac{2\pi}{4} \cdot kn} \\ &= 4, k = 0 \\ &= 0, k = 1, 2, 3 \end{aligned} \tag{2.12}$$

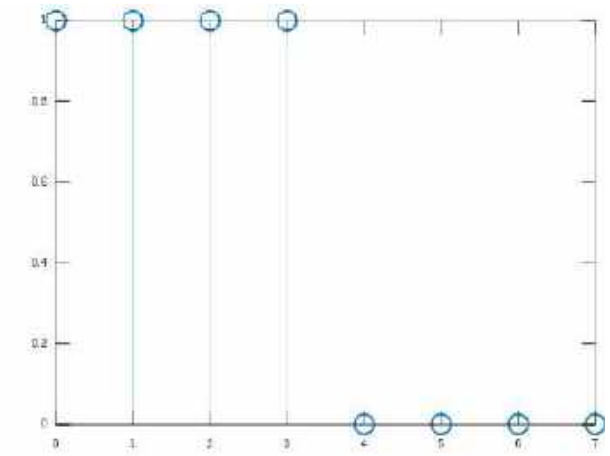

Figura 2.8: Reprezentarea funcției discrete $x[n]$

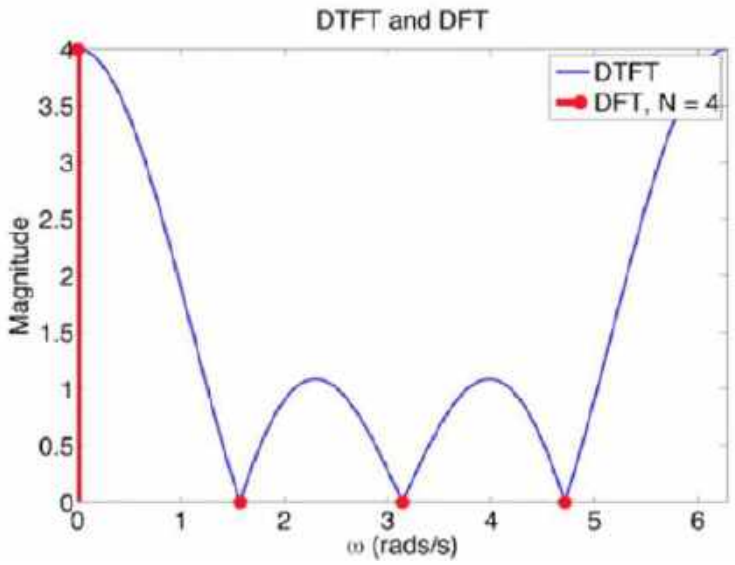

Figura 2.9: Diferența dintre TFTD($DTFT$) și TFD(DFT)

2.2.4 STFT si Spectrograme liniare

STFT (*eng. Short Time Fourier Transform*) este folosită pentru a calcula spectrograme. Acestea reprezintă o reprezentare esențială în prelucrarea și analiza discursului. De asemenea, spectrogramele conțin cele mai importante trăsături cu care putem antrena rețele neurale pentru prelucrări în domeniul audio.

În aplicarea *TFD* pe un semnal în domeniul timp, avem o reprezentare a tututor componentelor în frecvență. Însă aceasta este o imagine statică ce reprezintă o mediere a tuturor frecventelor pe toată durata semnalului. Cunoaștem ce componente în frecvența avem însă nu știm și la ce momente de timp apar predominant. Un semnal audio poate înseamna și o evoluție a frecvențelor în timp. Aveam deci de-a face cu o analiză dinamică. Ne dorim, prin urmare, să cunoaștem variația frecvențelor în timp.

Pentru a realiza acest lucru nu vom face TFD pe toată durata semnalului, ci doar la nivel de segment, numit și "frame".

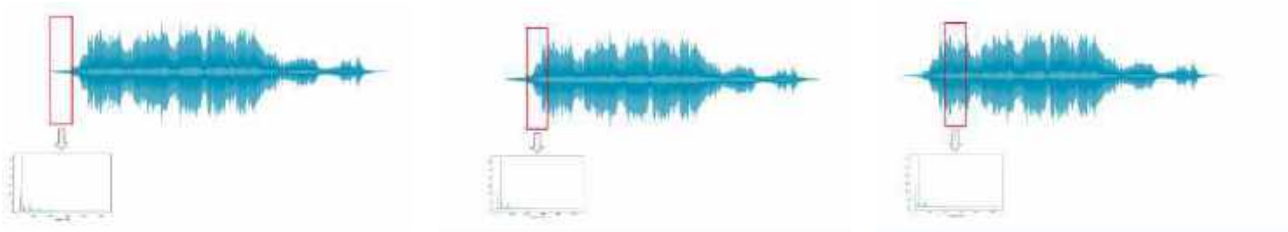

Figura 2.10: Reprezentarea Fourier la nivel de *segment*

Se folosește tehnica ferestrei glisande (*eng. windowing*) prin care trecem o fereastră de o anumită lungime prin fiecare *frame* al semnalului. Practic se înmulțește semnalul original cu o fereastră, ce se va glisa de la dreapta la stânga.

$$x_\omega(k) = x(k)w(k) \tag{2.13}$$

Aplicând o fereastră de tip dreptunghi avem următorul rezultat :

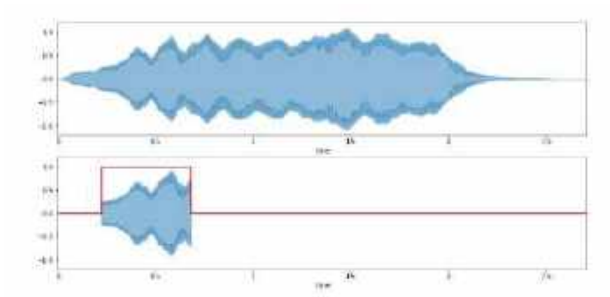

Figura 2.11: Aplicarea ferestrei pe o porțiune de semnal

Pentru transformata STFT avem următorii parametrii :

- Lungimea ferestrei (eng. window size): numărul de eșantioane din fereastră, lărgimea ferestrei în eșantioane
- Lungimea frame-ului (eng. frame size): numărul de eșantioane pe care aplicăm fereatra și TFD. (de obicei lungimea ferestrei se alege la fel cu cea a *frame*-ului)
- Lungimea suprapunerii: cât de mult suprapunem următorul *frame* cu cel anterior

- Lungimea pasului: câte eșantioane dorim să glisăm la dreapta când ne mutăm la un *frame* nou

După alegerea parametrilor, se va glisa fereastra la dreapta până se ajunge la finalul semnalului [20].

TFD vs STFT :

$$X[k] = \sum_{n=0}^{N-1} x(n)e^{-i2\pi n\frac{k}{N}}$$
$$S(m,k) = \sum_{n=0}^{N-1} x(n+mH)\cdot w(n)e^{-i2\pi n\frac{k}{N}} \tag{2.14}$$

Observăm că $S(m,k)$ depinde nu doar de frecvența k (coeficienții fourier pentru frecvența k), dar și de m, unde m reprezintă timpul/numărul *frame*-ului curent în care ne aflăm. (fiecare frame corespunde unui moment de timp). Astfel, STFT depinde atât de frecvență cât și de timp. Prin urmare, TFD face suma pe toată durata semnalului pe când STFD face suma doar pe durata *frame*-ului (se iau în considerare doar frecvențele din *frame*)

$x(n+mH) \rightarrow$ se observă faptul că se iau în considerare doar eșantioanele din *frame* , unde n este primul eșantion din *frame*

Se ia deci semnalul de la eșantionul din frame-ul curent si se înmulțeste cu fereastra $w(n)$.

La TFD extragem un singur vector spectral pentru un număr de eșantioane de frecvență (coeficienții fourier), fiind calculate frecvențele pentru toată lungimea semnalului. Din STFT pe de altă parte, ne rezultă o matrice spectrală de dimensiune ($numarEsantioaneFrecventa$ x $nrFrameuri$). Această matrice conține și coeficienții fourier complecși de la fiecare fereastră. Astfel vom înjumătăți dimensiunea matricei pentru a păstra doar informația utilă [21].

Fie $NE_t = numar\_esantioane\_timp$, $NE_f = numar\_esantioane\_frecventa$
si $NF = numar\_frameuri$.
Astfel :

$$NE_f = \frac{framesize}{2} + 1$$
$$NF = \frac{NE_t - framesize}{hopsize} + 1 \tag{2.15}$$

Dimensiunea matricei va deveni astfel $STFT \rightarrow (NE_f$ x $NF)$. Putem calcula NE_f folosind *framesize*, ce are lungimea calculată în eșantioane de timp și nu de frecvență, deoarece aplicăm STFT într-un număr de puncte egal cu lungimea *frame-ului*. Acum putem spune că avem informații și despre variația frecvenței în timp.

Compromisul timp/frecvență

Apare totuși un compromis atunci când calculăm spectrograma semnalului nostru. Atunci când creștem lungimea *frame*-ului Transformată Fourier va fi calculată în mai multe puncte, deci vor fi luate în considerare mai multe componente spectrale. În consecință, vom avea o reprezentare Fourier mai bună. Însă rezoluția în timp scade. Nu vom mai ști la fel de bine când au loc aceste frecvente.

Atunci când scădem lungimea *frame*-ului Transformata Fourier va fi calculată în mai puține puncte deci vor fi luate în considerare mai puține componente spectrale.În consecință vom avea

o reprezentare Fourier mai slabă. Însă rezoluția în timp crește în acest caz. Vom ști mai bine când are loc fiecare frecvență și cu ce intensitate.

Alegerea lungimii *frame*-ului depinde de aplicație. În cazul nostru ne va intersa să avem o rezoluție în timp cât mai bună. Avem nevoie să cunoaștem când se produc sunetele și să le putem diferenția. Astfel se vor alege frame-uri mai mici.

Un alt factor important este tipul de fereastră ales. În funcție de fereastră, semnalul își va schimba spectrul. Nu se folosește ferestra dreptunghiulară niciodată deoarece creează împrăștieri spectrale (*spectral leakage*), ci se folosește fereastra *Hanning* ce evită acest lucru.

$$w(k) = 0.5 \cdot \left(1 - \cos\left(\frac{2\pi k}{K-1}\right)\right), k = 1 \dots K \tag{2.16}$$

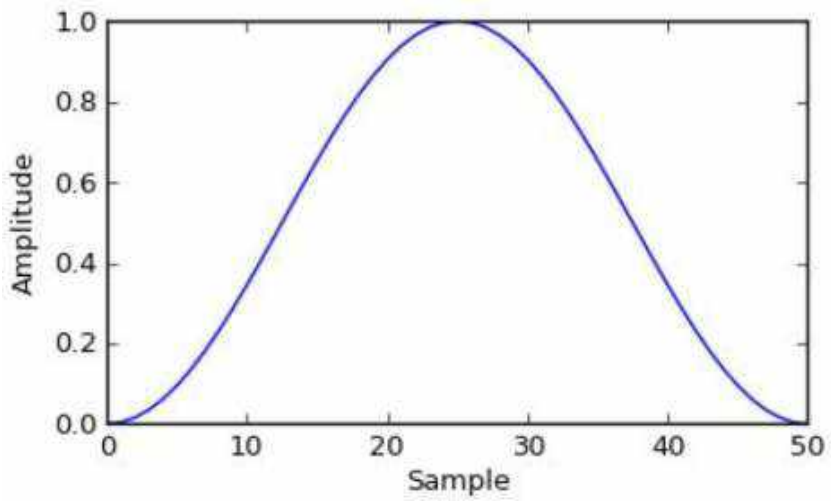

Figura 2.12: Fereastra Hanning cu ecuația propriu zisă

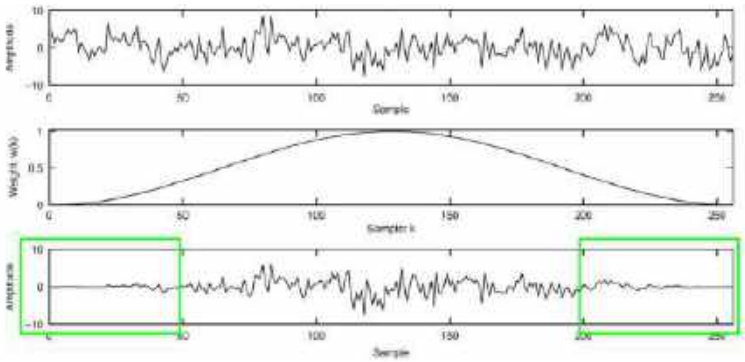

Figura 2.13: Fereastra Hanning aplicată pe un semnal evitând împrăștieri spectrale

Vizualizarea spectrogramei

Pentru vizualizarea spectrogramei se va lua pătratul modulului STFT, pentru a nu mai avea numere complexe ci doar numere reale în reprezentarea noastră.

$$Y(m,k) = |S(m,k)|^2 \tag{2.17}$$

Rezultă astfel un *heat-map (eng)*. Pe OX este reprezentat timpul sau numărul total de *frame*-uri, iar pe OY avem reprezentate toate frecvențele (toate eșantioanele de frecvență).

Modul în care auzim este logaritmic. Așa că va trebui să facem o conversie din liniar în logaritmic pentru amplitudine (valorile în sine, dimensiunea a-3-a adică), pentru o mai bună reprezentare din punct de vedere al percepției. Totuși nu este una completă. Vom discuta în continuare în subcapitolul următor[22].

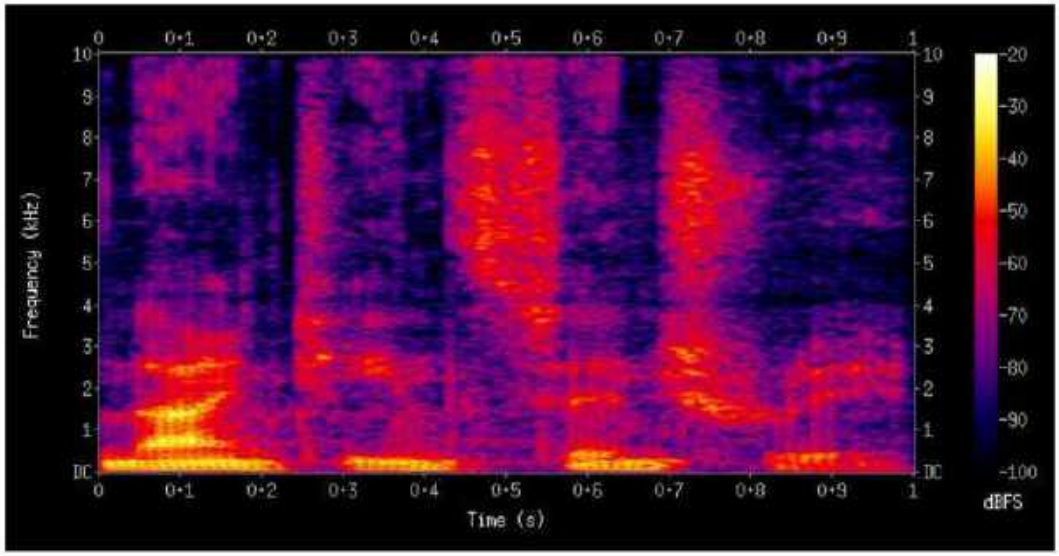

Figura 2.14: Spectrograma

2.2.5 Spectrograme Mel

Spectogramele Mel sunt un tip de spectrograme derivate din cele normale liniare. Se folosesc foarte des în aplicațiile audio de inteligență artificală.

Spectogramele normale au o reprezentare liniară a frecvenței. Însă acest lucru este problematic deoarece omul percepe în mod diferit frecvența sunetelor. În urmă realizării unui experiment psihoacustic s-a constatat următorul lucru :

Fie două note `C2` și `C4` de frecvență joasă cu diferența de 200 Hz între ele și `G6` și `A6` de frecvență înaltă cu diferența tot de 200 de Hz între ele. Experimental vorbind, noi simțim diferența între `C2` și `C4`, însă între `G6` și `A6` nu simțim nicio diferență. Lucrul acesta se datorează faptului că percepem frecvența(*eng. pitch-ul*) într-un mod neliniar. Avem rezoluție mai bună la frecvențe joase decât la cele înalte. Acest lucru arată faptul că odată cu creșterea frecvenței, va fi mai dificil să percepem diferențele dintre note. Putem spune astfel că auzul uman este unul logaritmic.

Atunci când vrem să extragem trăsături dintr-un semnal audio avem în vedere următoarele lucruri:

- vrem o reprezentare a frecvenței în timp bună
- vrem o reprezentare relevantă a amplitudinii în functie de percepția noastră. (lucru ce se poate face utilizând spectrograme normale convertind scara valorilor de amplitudine într-o scară logaritmică)
- vrem o reprezentare relevantă a frecvenței dupa modul în care și percepem (convertirea axei de frecvență)

Primele două lucruri pot fi îndeplinite și de spectrogramele normale, însă numai spectrogramele Mel bifează toate cele 3 lucruri pe care dorim să le avem în vedere. Pentru a înțelege cum putem calcula/genera o spectrogramă Mel trebuie mai întâi să înțelegem ce este scara Mel.

Scară Mel este o scară ce oferă informații bune legate de percepția umană a frecvenței. Este o scară logaritmică ce este relevantă din punct de vedere perceptual. Acest lucru se datorează faptului că punctele echidistante pe scară au și aceeași distanță perceptuală (spre exemplu pentru oricare două puncte de diferență 200 de Meli acum se va simți diferența de percepție oriunde ne-am află pe scară). Practic reprezentarea nu mai este în frecvența ci în percepție. Distanța între puncte nu se mai calculează în frecvență ci în percepție.

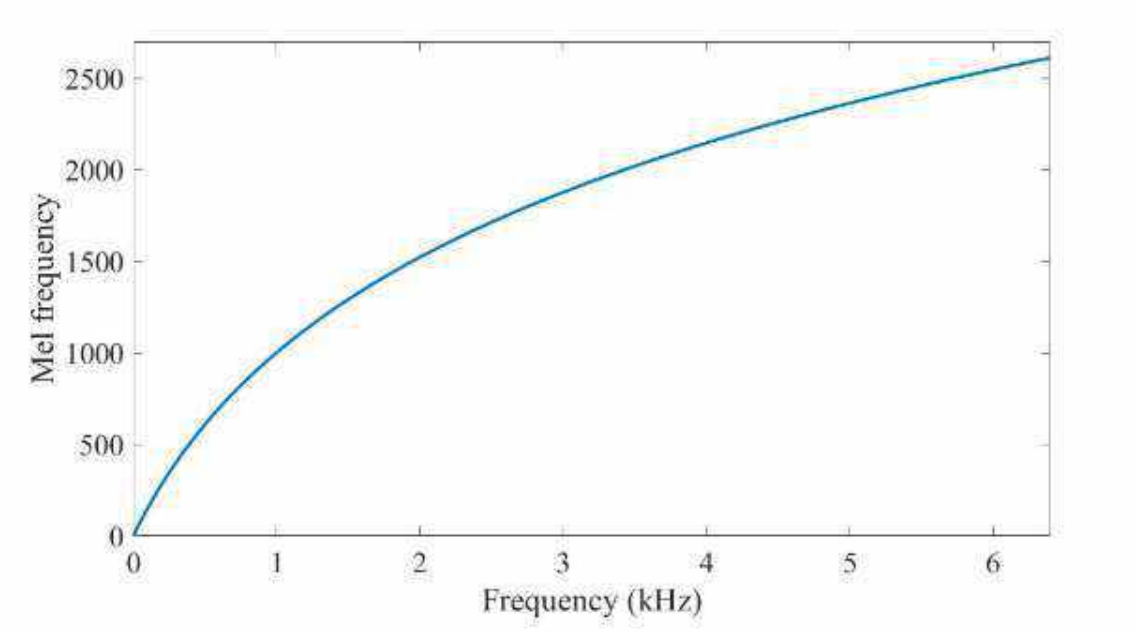

Figura 2.15: Scara Mel

Pentru a converti frecvența din Hz în Mel și invers se folosesc următoarele relații :

$$m = 2595 \log\left(1 + \frac{f}{500}\right)$$
$$f = 700\left(10^{m/2595} - 1\right) \tag{2.18}$$

Cum extragem spectrogramele Mel:

1. Extragem STFT
2. Convertim amplitudinea frecvențelor în dB (trecem în scara logaritmică)
3. Convertim frecvențele din Hz în Mel

Primele 2 puncte le-am realizat deja calculând spectrogramele liniare. Noutatea apare în convertirea frecvențelor din Hz in Mel. În trecerea din spațiul liniar în spațiul perceptiv.

Cum convertim frecvențele din Hz în scară Mel :

1. Se alege un număr de benzi Mel
2. Calculăm/Computăm bancurile de filtre mel.
3. Aplicăm filtrele Mel pe spectrograma liniară

Alegerea numărului de benzi Mel este un parametru ce poate varia in funcție de aplicație. Putem avea 40, 60 chiar și 90 de benzi Mel. Construcția benzilor Mel se face în următorii 4 pași:

1. convertim cea mai joasă și înaltă frecvență din plaja noastră de valori, în scara Mel utilizând formula de conversie.

$$m = 2595 \cdot \log\left(1 + \frac{f}{500}\right) \tag{2.19}$$

2. în acest moment avem un interval de la cea mai joasă frecvența Mel la cea mai înaltă. Acest interval va fi împărțit în mai multe segmente egale, alegând un număr de puncte echidistante echivalent cu numărul de benzi mel.

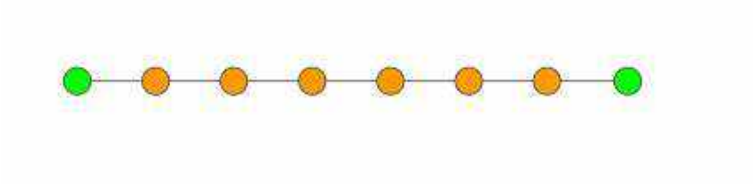

Figura 2.16: Segmentarea intervalului de frecvențe

3. vom converti după aceste puncte înapoi în Hz utilizând formula de mai jos și vom aproxima la cel mai apropiat eșantion de frecvență. Aceste puncte vor reprezenta centrele benzilor mel. Conversia înapoi în Hz se face pentru noi pentru a vedea cărei plaje de valori în frecvență aparține acea bandă mel.

$$f = 700\left(10^{m/2595} - 1\right) \tag{2.20}$$

4. vom crea acum filtrele triunghiulare (benzile Mel). Pe graficul de mai jos "weight" arată cât de mult filtrul influențează semnalul în acel punct. Pentru valoarea 1 înseamnă că semnalul este lăsat să treacă complet, sub 1 începe să se filtreze, lucru ce semnifică faptul că frecvențele filtrate seamănă în percepție cu frecvența centrală, de aceeaa sunt filtrate afară. Centrul unei benzi este frecvența convertită mai sus în Mel. Extrema inferioară și extremă superioară a filtrului sunt centrele benzilor anterioară respectiv successivă.

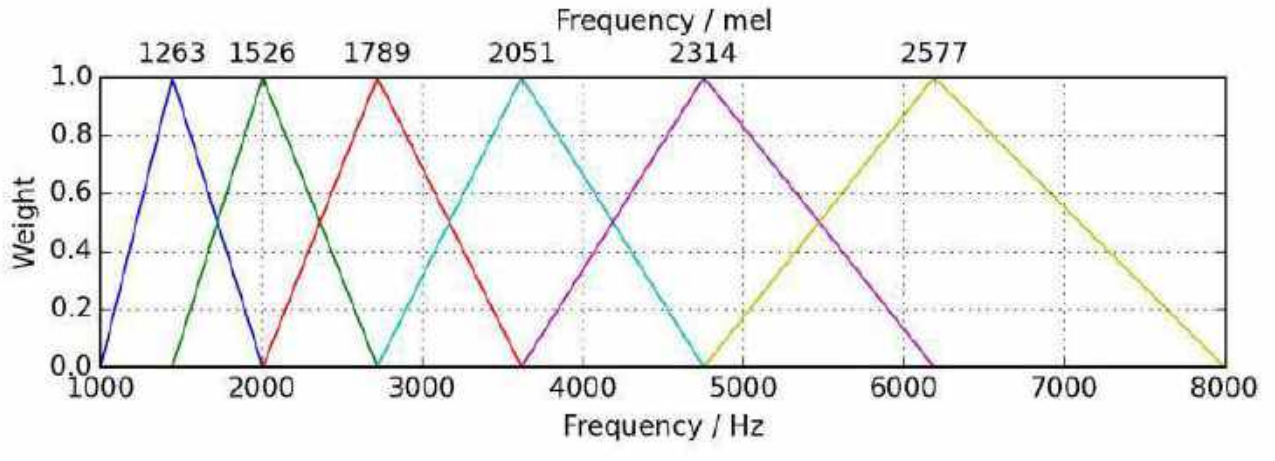

Figura 2.17: Bancurile de filtre Mel

În figură mai observăm că avem 6 benzi (filtre), fiecare cu o frecvență centrală. Observăm că diferența dintre toate centrele bancurilor Mel (partea de sus a graficului) este aceeași : $1526 - 1263 = 2051 - 1789 = 2314 - 2051... = 263$

Dacă ne uităm și la partea de jos diferențele dintre centre în Hz nu sunt aceleași. Cu cât avansăm în frecvență, lărgimea filtrului va crește pentru că frecvențele seamănă din punct de vedere al percepției și astfel putem elimina mai multe componente.
Prin urmare tot ce facem noi este de fapt o filtrare a spectrogramei originale. Filtrăm spectrograma astfel încât să păstrăm o reprezentare corectă în percepție, eliminând din frecvențele ce seamănă din punct de vedere al percepției.

Aceste filtre se reprezintă cu o matrice de dimensiune:

$$M = (numar\_benzi, \frac{framesize}{2} + 1), \tag{2.21}$$

unde *framesize* este numărul de puncte pe care s-a aplicat STFT.

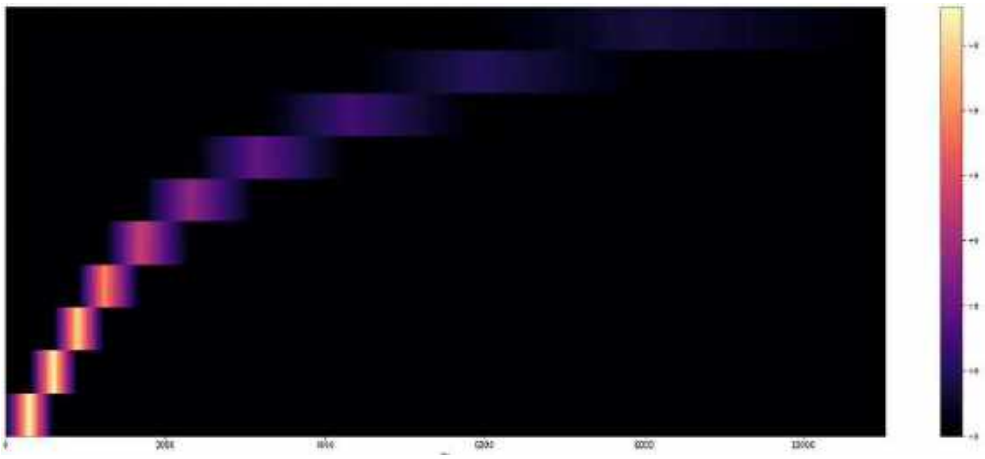

Figura 2.18: Reprezentarea matricei de benzi Mel

Dacă ar fi să reprezentăm în sistemul de coordonate cartezian, pe axa OY avem benzile iar pe axa OX avem frecvențele. Intervale de frecvență de diferite lungimi (intervale mici pentru frecvente mici, intervale mari pentru frecvente mari) vor corespunde unei benzi specifice. Se face de fapt o conversie al unui interval de frecvente în banda mel asociată.

Ne amintim dimensiunea spectrogramei :

$$Y = (\frac{framesize}{2} + 1, nr\_frameuri), \tag{2.22}$$

unde $nr\_frameuri$ reprezintă momentele de timp, iar $\frac{framesize}{2} + 1$ reprezintă numărul de frecvențe, numărul de eșantioane în frecvență.

Avem prin urmare doua matrici M și Y. Având în vedere dimensiunile lor, ele se vor putea înmulți. Prin inmulțirea lor vom obține spectrograma Mel.[23]

$$M \cdot Y = SpectrogramaMel \tag{2.23}$$

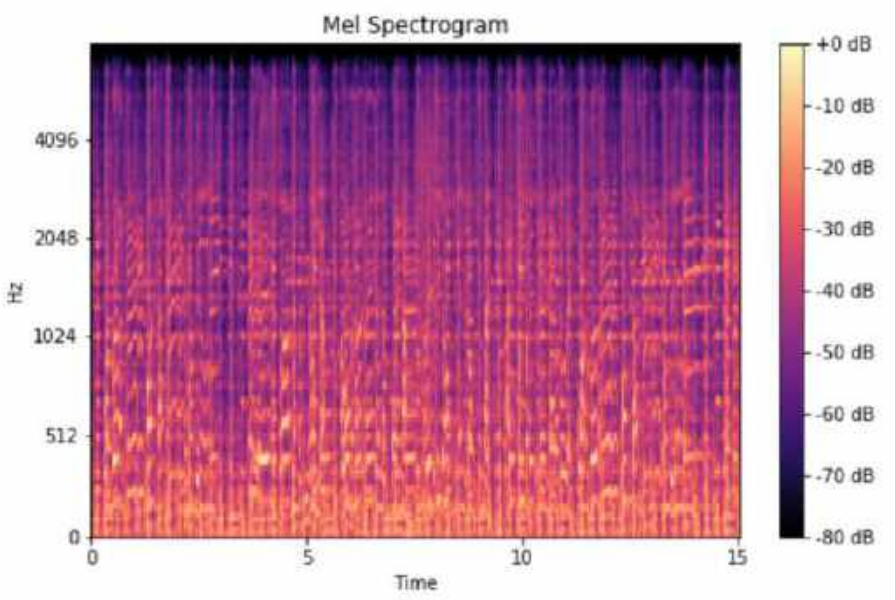

Figura 2.19: Spectrograma Mel

Recapitulând, pașii pe care i-am efectuat au fost :

- Proiectarea filtrelor Mel
- Filtrarea frecvențelor spectrogramei liniare
- Asocierea frecvențelor cu benzile mel

Spectrogramele Mel se pot folosi într-o multitudine de aplicații precum :

- Clasificarea emoțiilor bazate pe semnalul audio
- Clasificarea pe genuri muzicale a melodiilor
- Recunoaștere vocală
- Sintetizare de voce

2.3 Sisteme de transcriere automată a textului în vorbire

2.3.1 Procesarea naturală a limbii (*NLP (eng)*)

Introducere în *NLP* și Reprezentarea Vectorială a cuvintelor (*Word2Vec(eng)*)

Pentru început, pentru a putea transcrie într-un model matematic cuvintele, trebuie să ne punem întrebările: Ce înseamnă limbajul, o limbă? Ce înseamnă "înțelesul" unui cuvânt? Limbajul este o formă de comunicare incertă, în care facem mai mult deducții probabilistice despre ce a vrut să spună interlocutorul. Acesta nu are că funcție doar transmiterea de informații dar îndeplinește și funcții sociale de care noi oamenii suntem dependenți.

Cum reprezentăm înțelesul unui cuvânt?

Cuvintele au în general intelesuri profunde. Există foarte multe nunațe pentru același cuvânt. Cuvântul *doctor* spre exemplu, poate fi utilizat cu mai multe înțelesuri: *doctor* ce lucrează în spital (cu sensul de medic) sau *doctor* în filosofie (cu sensul de om cu studii superioare).

O prima metodă pentru a găsi înțelesul cuvintelor este utilizarea dicționarelor de sinonime. Există *WordNet*, o bază de date lexicală a relațiilor semantice între cuvinte în mai mult de 200 de limbi. *WordNet* leagă cuvintele în relații semantice, inclusiv sinonime. Problema principala în utilizarea unui astfel de dicționar, este că lipsesc nuanțele. Nu oferă toate definițiile, reprezentările cuvântului în toate contextele. Mai mult, este si dificil de întreținut deoarece trebuie actualizat manual.

Reprezentarea cuvintelor ca simboluri discrete

În *NLP*-ul tradițional, privim cuvintele că și simboluri discrete.

Fie cuvintele *hotel, conferință, motel*. Putem reprezenta cuvintele printr-o reprezentare de tipul *one-hot encoding* astfel:

Se consideră un vocabular de dimensiune 10, unde cuvântul hotel este al doilea cuvânt din vocabular, motel este al 7-lea și conferință al 8 lea"

$$\begin{aligned} motel &= [0100000000] \\ hotel &= [0000001000] \\ conferinta &= [0000000100] \end{aligned} \tag{2.24}$$

Valoarea de 1 în vector reprezintă cuvântul ce se află pe poziția respectivă în vocabular. Celelalte valori de 0 sunt celelalte cuvinte.

Problema cu această reprezentare apare în momentul în care avem un vocabular de dimensiune mare. (1 milion spre exemplu). Ar trebui ca vectorii noștrii să aibă dimensiuni extrem de mari atunci, lucru ineficient. Mai mult noi dorim să înțelegem și relațiile și înțelesurile dintre cuvinte. Cuvintele *motel* și *hotel* au o anume asemănare. Îmi doresc să am un mod de vedea matematic asemănarea între ele. Cu reprezentarea *one-hot* ele sunt privite că fiind niște entități total diferite. Matematic vorbind, cei 2 vectori sunt ortogonali (nicio similaritate).

Reprezentarea cuvintelor după context (Word2Vec)

Soluția apare gândindu-ne să găsim înțelesul cuvintelor, uitându-ne la contextul în care se află. Înțelesul unui cuvânt va fi dat de cuvintele ce apar cât mai frecvent lângă el. În realitate, dacă știm în ce context să folosim un cuvânt putem spune că am înțeles ce înseamnă acel cuvânt, chiar dacă nu-i putem oferi o definiție riguroasă. Spre exemplu, doresc să înțeleg ce

înseamnă cuvântul *bancă*. Voi lua mai multe propoziții în care se află acest cuvânt, și în funcție de aceste enunțuri îmi pot da seama ce înseamnă *bancă*.

*Cea mai veche **bancă** din lume este considerată a fi Banca Monte dei Paschi, înființată în 1472.*
*...țn condițiile societății postindustriale, o **bancă** de date include un ansamblu de "fișiere informatizate...*
*Dacă aș avea mijloace, n-aș face nimic altceva decât o **bancă** de lemn în mijlocul mării...*

Când un cuvânt "*w*" (*bancă* în cazul nostru) apare într-un text, contextul este dat de un set de cuvinte ce apar lângă el (o fereastră fixă ce selectează un număr de cuvinte). Înțelesul cuvântului *bancă* va fi dat de contextul în care se află. Vom construi un vector pentru fiecare cuvânt ales astfel încât să fie similar cu vectorii de cuvinte ce apar în contexte similare. În literatura de specialitate :
word vectors = word embeddings = word representations

Mai mult, cuvântul căruia dorim să-i determinăm înțelesul va prezice cuvintele de lângă el sau contextul. Așadar, elementele unei reprezentări sunt de fapt probabilitățile contextului.

Putem spune că această reprezentare *Word2Vec* ne ajută să găsim similiritati între cuvinte bazându-ne pe context deci pe trăsăturile comune, asemănător algoritmului K-means, creandu-se astfel niște clustere de cuvinte în funcție de context. Se creează astfel un spațiu vectorial al cuvintelor (vezi fig. 2.22). [24]

Putem acum matematic să ajungem de la un cuvânt la altul. Pentru cuvântul *king (eng pentru rege)*, spre exemplu, vom scădea cuvântul *man (eng pentru bărbat)* (adică vom extrage din înțelesul cuvântului *rege* trăsăturile de *bărbat*) și vom adună cuvântul *woman (eng pentru femeie)* (adăugăm trăsăturile de *femeie* la cuvântul *rege*), rezultând astfel cuvântul *queen (eng pentru regina)*. Se generează astfel un vector reprezentativ pentru Regina (vezi fig. 2.20). [25]

Putem prin urmare să facem analogii: *cum e regele pentru bărbat, așa și regina pentru femeie.*

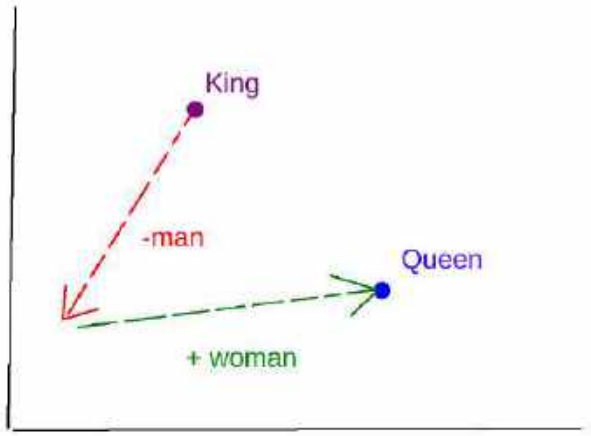

Figura 2.20: Determinarea vectorială a cuvântului *woman (eng)*

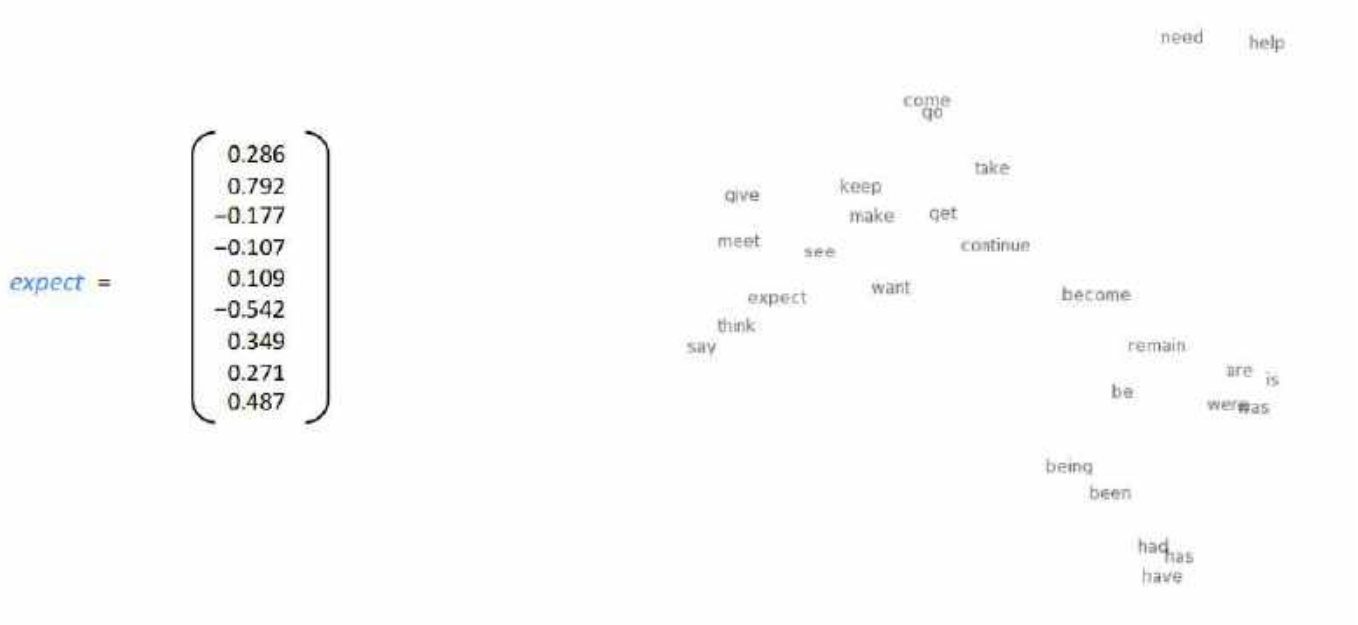

Figura 2.21: Reprezentarea vectorială a cuvântului *expect (eng)*

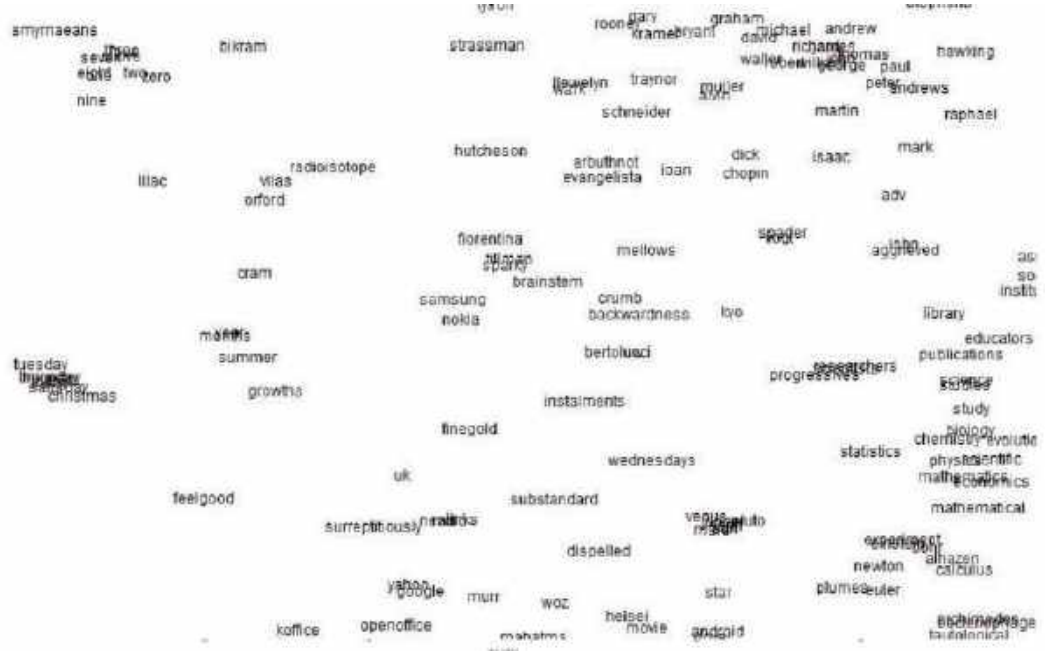

Figura 2.22: Spațiul vectorial al unui număr restrâns de cuvinte

Structuri dependente

Ne punem acum întrebarea: "ce fel de modele matematice putem crea având în vedere structura propozițiilor?". Până acum am vorbit doar de analiza cuvintelor și de înțelesul lor și de faptul că pot avea înțelesuri apropiate. Totuși încă nu am analizat cum le putem pune împreună. Cum le putem conecta. Momentan vom considera cea mai mică unitate ca fiind cuvântul. Cuvintele se combină în fraze, iar frazele se combină în fraze și mai mari formând paragrafe. Ne întrebăm ce fel de structuri avem ? În mod normal am putea folosi *regulile gramaticale* pentru a vedea structurile. Prin *structuri/reguli gramaticale* putem construi un număr infinit de propoziții. Totuși pe lângă regulile gramaticale mai există o soluție mai eficientă anume "Structurile dependente" (*eng. Dependency Structures*). Aceastea constau în ideea de a nu mai categorisi cuvintele în părți de vorbire ci de a spune direct ce cuvânt depinde de celălalt. Astfel, nu mai avem nevoie și de analiză gramaticală. Un exemplu ar fi următorul:

Uită-te în cutia mare de lângă ușa din bucătărie

Exista 2 metode de analiză a acestei propoziții :

1. Analiza gramaticală
2. Putem considera *"Uită"* ca fiind radăcina. Ne uităm după ce cuvinte depind de *"Uită"*. Unde ne uităm ? Ne uităm *"în cutia"*. Dar "*cutia*" ce dependență are ? "*mare*" este dependență cuvântului "*cutie*". O altă dependență a cuvântului "*cutie*" este *"de lângă ușa"* și *"din bucătărie"* deoarece ne spun unde se află "*cutia*"

Figura 2.23: Analiza 1 a structurilor dependente

Prin urmare această metodă ne spune modul în care cuvintele sunt conectate între ele.
Alt exemplu:

"Polițist omoară om cu cuțit."

În acest caz avem 2 interpretări:

1. Ori polițiștii sunt cei care au omorât folosind un cuțit
2. Ori omul avea cuțitul și a fost omorât de către polițiști

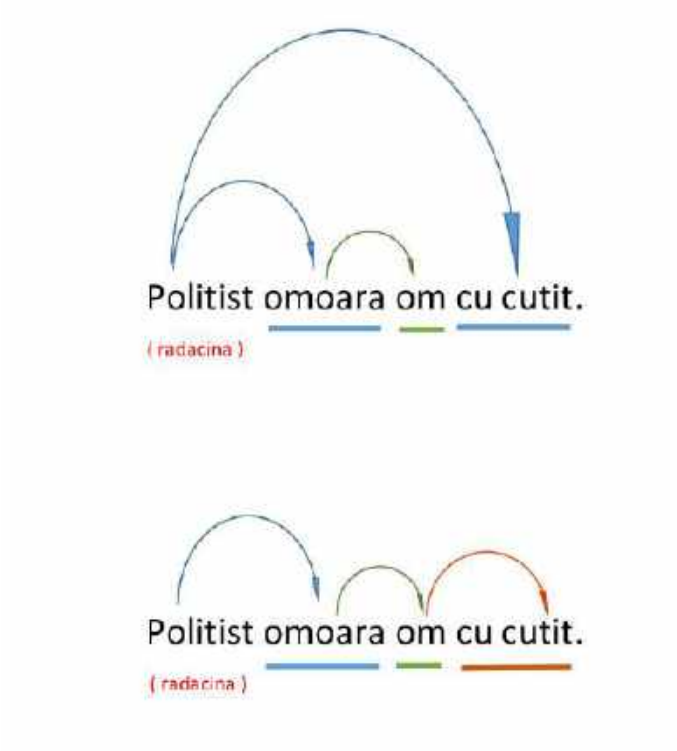

Figura 2.24: Analiza 2 a structurilor dependente

Avem 2 tipuri de dependențe. Ori "*cuțit*" e dependent de "*om*" ori "*cuțit*" e dependent de "*polițist*". Apare deci ceea ce se numește o situație de ambiguitate, în care nu știm ce cuvânt e dependent de celălat. Se mai numește și *prepositional phrase attachement ambiguity (eng).* În programare avem reguli clare, nu există ambiguitate. În limbajul natural totuși apar ambiguități, însă noi suntem capabili să distingem/să interpretăm corect propozițiile. Nu avem nevoie de foarte multe cuvinte că cineva să ne înțeleagă. [26]

Metodă această cu dependențe ne poate ajută să identificăm chiar și *pattern*-uri și relații semantice (aflăm care cuvinte funcționează împreună cu care). [27]

Modele de limbă (*eng. language models*)
Până acum sarcinile ce trebuiau îndeplinite au fost:

1. Sensul cuvintelor (solutia Word2Vec)
2. Dependența structurilor (cum depind cuvintele unele de altele)

Următoarea problema o presupune modelarea limbajului (*language modelling (eng)*). Un model de limba este responsabil cu prezicerea/predicția apariției următorului cuvânt. Dat fiind o secvență de cuvinte dorim să aflăm distribuția de probabilitate a următorului cuvânt știind cuvintele anterioare.

$$P(x(t+1)|x(t),....x(1)),\ unde\ x(t+1)\ poate\ fi\ orice\ cuvant\ din\ vocabularul\ nostru \quad (2.25)$$

Astfel, se asociază o probabilitate unei bucăți de text. Probabilitatea bucății de text este produsul tuturor probabilităților condiționate al fiecărui cuvânt.

$$P\left(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}\right) = P\left(\boldsymbol{x}^{(1)}\right) \times P\left(\boldsymbol{x}^{(2)} \mid \boldsymbol{x}^{(1)}\right) \times \cdots \times P\left(\boldsymbol{x}^{(T)} \mid \boldsymbol{x}^{(T-1)}, \ldots, \boldsymbol{x}^{(1)}\right) = \prod_{t=1}^{T} P\left(\boldsymbol{x}^{(t)} \mid \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(1)}\right) \tag{2.26}$$

Un exemplu de utilizare a modelelor de limbă este atunci când scriem pe *Google* un început de propoziție iar motorul de căutare ne sugerează continuarea. Acum întrebarea este cum putem să determinăm apariția următorului cuvânt dată fiind o secvență?

N-Grame

N-gramele sunt o grupare (*eng. chunk*) de n cuvinte consecutive. Vom colecta statistici să vedem cât de des apar diferite *N-grame* și vom folosi aceste rezultate pentru a prezice cuvântul numărul n.

Presupunem că $x(t+1)$ depinde doar de cele $n-1$ cuvinte precedente. Determinarea acestor probabilități se face, calculând probabilitatea lor de apariție într-un text foarte mare (numărăm aparițiile diferitelor combinații de n-grame).

Fie următoarea propoziție în engleză și w, cuvântul ce trebuie prezis:

As the procuror started the clock, the students opened their....

$$P(w|\text{students opened their}) = \frac{\text{count(students opened their w)}}{\text{count(students opened their)}} \tag{2.27}$$

count(students opened their) = 1000 (apariția structurii "students opened their")

count(students opened their books) = 400 (apariția cuvântului "books" langa structura "students opened their")

P(books) = 0.4 (probabilitatea ca următorul cuvânt sa fie "books")

count(students opened their exams) = 100 (apariția cuvântului "exams" lângă structura "students opened their")

P(exams) = 0.1 (probabilitea ca următorul cuvânt sa fie "exams")

Răspunsul va fi "exams". Acest lucru este dat și de apariția în context a cuvântului *procuror (eng)*. Prin urmare avem nevoie de un context mai mare pentru a prezice mai bine următorul cuvânt. Însă odată cu creșterea numărului n apare următoarea problemă. [28]

Problema rarității (The sparsity problem)

În cazul în care un cuvânt nu apare niciodată atunci va trebui să punem un δ inițial la toate cuvintele pentru a evita riscul apariției valorii de 0. Distribuția de probabilitate va avea astfel niște *spike-uri*. În cazul în care nu găsim nicio structura (probabilitatea apariției trigramei "students opened their" este 0) trebuie să renunțăm la structura și să ne alegem doar ultimele 2 elemente "opened their".

Așadar, cu cât valoarea lui n este mai mare cu atât problema rarității va apărea mai des. De fapt s-a și demonstrat că nu putem alege un $n \geq 5$.

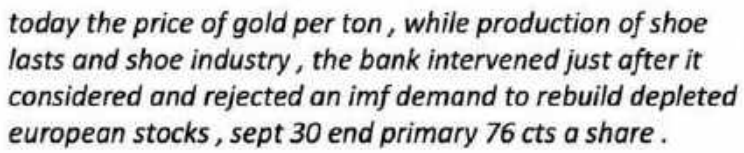

Figura 2.25: Corpus de text in limba engleza

Mai sus se poate observa un text generat corect gramatical folosind predicție cu trigrame. Totuși textul este incoerent. Pentru că s-au folosit trigrame, memoria este doar de 3 cuvinte. Avem nevoie de memorie mai mare pentru a ne da seama de următorul cuvânt. Însă după cum am văzut odată cu creșterea lui n apare problema rarității. Soluția apare în utilizarea rețelelor neurale. [29]

Traduceri

Mai avem o ultima sarcină importantă în domeniul NLP anume traducerea. Dorim să traducem o propozite x dintr-o limbă (limba sursă) într-o propoziție y în altă limbă (limba țintă).[30]

Dorim să aflăm cea mai bună propoziție în *Engleză* y, pentru propoziția x de la intrare în limba *Franceză*.[31]

$$argmax_y(P(y|x)) \rightarrow \ \textit{argumentul maxim al probabilitatii conditionate} \tag{2.28}$$

$$P(y|x) = p(x|y)P(y) \tag{2.29}$$

Vom împărți, prin urmare, această sarcină în 2 probleme separate :

1. $P(x|y)$ Modelează local cum ar trebui sa fie tradus un mic set de cuvinte și fraze

2. $P(y)$ language model este un model de limba discutat anterior (language model) ce oferă o ordine buna a cuvintelor și o structură corectă

Pentru a calcula modelul local de traducere ($P(x|y)$ va fi nevoie sa mai adăugăm încă un parametru anume alinierea α. Astfel probabilitatea ce trebuie sa o calculăm va deveni $P(x, \alpha|y)$. Vom pune atunci următoarele întrebări :

- care e probabilitatea ca cuvântul x sa apară având în vedere pe y
- care e probabilitatea ca anumite cuvinte x sa se alinieze cu y ținând cont și de poziție.

Avem mai multe tipuri de alinieri:

- aliniere 1 la 1
- aliniere 1 la mai mulți
- aliniere mai mulți la 1

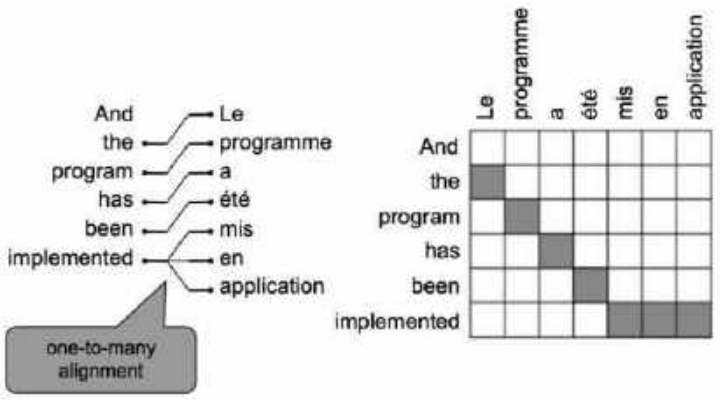

Figura 2.26: Aliniere 1 la mai mulți

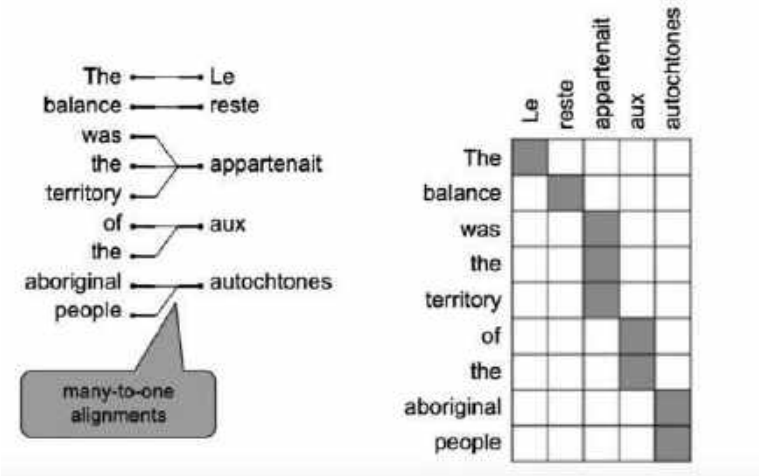

Figura 2.27: Aliniere mai mulți la 1

SubWord Models

Am discutat în subcapitolul precedent despre fonetică și fonologie. Am observat faptul că limbajul uman conține un set mic de unități distincte ce se numesc foneme. Gură noastră poate genera o infinitate de sunete distincte însă noi distingem doar un număr mic dintre ele.
O alternativă de la analiză la nivel de cuvânt, ar fi trecerea la nivel de caracter. Să nu mai analizăm "*word* N-grams" ci "*character* N-grams".

Care sunt motivațiile pentru o astfel de analiză la nivel de caracter ?

1. Unele limbi străine nu au o segmentare foarte clară a cuvintelor
2. Sunt multe cazuri în care limba are cuvinte ce pot fi prepoziții, sau pronume, elemente mici ce pot fi scrise împreună sau separate, precum*: "n-am fost la școală azi vs nu am fost la școală azi" etc.*
3. Unele limbi au un vocabular mult prea bogat.
4. Translația devine câteodată transliteratie. În aceste cazuri nu este nevoie decât să rescriem cuvântul conform sunetelor limbii în care dorim să traducem
5. De asemenea în cazul în care avem cuvinte scrise greșit, analiză la nivel de caracter poate totuși să genereze o soluție.

6. De asemenea pentru sistemele de transcriere în vorbire dorim o analiză la nivel de sunete. Deoarece în vorbire cea mai mică unitate este sunetul/fonemul.

7. De asemenea sistemele de transcriere în vorbire reprezintă tot un fel de traducere dar la nivel de sunet.

Prin urmare sunt cazuri în care vom dori să analizăm mai jos de nivelul cuvintelor. Mai mult, s-a demonstrat că analiză la nivel de caracter a avut un succes la fel de mare că analiză la nivel de cuvânt. Totuși, faptul că funcționează la fel de bine ca analiza la nivel de cuvânt este ușor contraintuitiv, deoarece caracterele de unele singure nu sunt purtătoare de sens. Totuși, în aplicații de rețele neurale mai ales, se găsesc în timp legături matematice între caractere și chiar se crează cuvinte purtătoare de sens.

Problemele discutate anterior rămân valabile: sensul cuvintelor va fi înlocuit de sensul caracterlor în vorbire, dependențele structurale vor fi acum la nivel de caracter (cum depind caracterele unele de altele) iar modelarea limbii și traducerea vor fi și ele la nivel de caracter.

Studierea acestor 4 probleme constituie baza înțelegerii unui sistem de transcriere (traducere) a textului în vorbire. Înainte de traducerea caracterelor, trebuie găsit în prelucrarea lor, înțelesul, dependența lor una de cealaltă dar și distribuțiile lor de probabilitate. Până în acest moment au fost prezentate problemele ce apar în prelucrarea limbajului scris. În cele ce urmează se va discuta despre soluții ale acestor probleme și cum se pot integra și în sistemul de transcriere în vorbire.

2.3.2 Metode deterministe

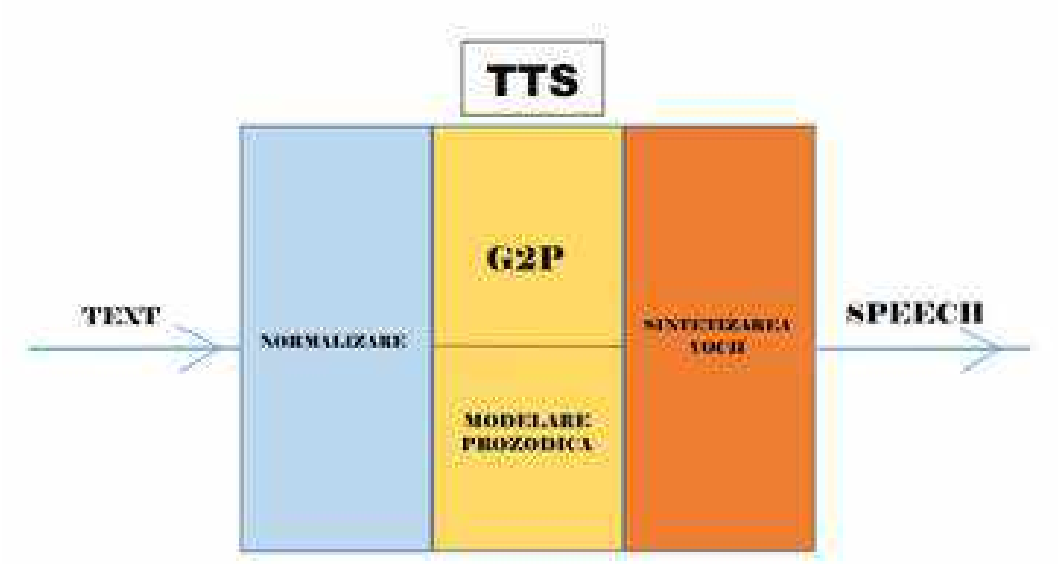

Figura 2.28: Logica unui sistem de transcriere a textului în vorbire (*TTS*)

Sistemele clasice de transcriere automată a textului în vorbire prezintă mai multe blocuri de prelucrare.

Primul bloc este cel de normalizare a textului (Text Normalisation).
Acesta constituie aducerea textului la o formă cât mai ușoară de prelucrare și constituie următorii pași:

- Transcrierea abrevierilor și simbolurilor în cuvinte complete (*Dl* devine Domnul, *Dna* devine Doamna, $ devine dolar etc)

- Trecerea tuturor literelor mari în litere mici
- Transcrierea tuturor numerelor în litere
- Eliminarea simbolurilor ce nu influențează sunetele, pronunția (cratima "-")
- Eliminarea semnelor de punctuație ce arată diverse forme de expresivitate ("!", "..."), păstrând doar semnele minime de punctuație ("," și ".")

Următorul bloc îl constituie blocul de procesare a textului ce este alcătuit din două etape:

- prelucrarea la nivel segmental, ce convertește textul în transcriere fonetică (*Grapheme to Phoneme* sau blocul G2P)
- prelucrarea la nivel suprasegmental, ce reprezintă elementele de prozodie (partea "melodică" a discursului) precum : Pauzele, Intonația, Intensitatea etc.

Blocul G2P, este cel mai important din partea de procesare a textului, deoarece oferă prelucrarea necesară pentru a mapa literele cu sunetele și a genera mesajul transmis. Astfel pentru partea de transcriere fonetică avem mai multe abordări:

1. Folosirea de dicționare - această metodă presupune maparea fiecare cuvânt din vocabularul nostru cu transcrierea să fonetică
2. Utilizarea PLS (*Pronounciation Lexical Specification*) - aceasta este o reprezentare mai detaliată a informației de pronunție a elementelor lexicale.
3. Regulile de transcriere fonetică - prin această metodă putem crea în teorie un model matemtic pentru automaizarea procesului de transcriere.
4. Hibridă - aceasta constituie folosirea regulilor de transcriere fonetică a cuvintelor, iar pentru excepții vom folosi dicționare.

Problema principală ce apare la acest bloc este cea a homografelor. Homografele sunt cuvintele ce arată identic din punct de vedere ortografic, dar au pronunție diferită. Pentru prima soluție nu se va ști ce transcriere fonetică să se aleagă pentru acel cuvânt. Soluțiile 2 3 și 4 rezolvă această problemă deoarece oferă mai multe informații despre rolul acelui cuvânt în text.

Următorul bloc îl constituie **blocul supra-segmental** (modelarea prozodică). Acesta este folosit pentru a crea o transcriere în vorbire cât mai apropiată de caracteristicile umane. Acesta presupune analiză și identificarea "pattern-urilor" în spectrogramele semnalelor audio vocale, pentru a vedea când se produc pauze, cum se schimbă intonația în timp etc. În general ne va interesa doar prelucrarea la nivel segmental, naturaleța în vorbire neefind un factor important în transmiterea unui mesaj obiectiv. Naturaleța în discurs devine o problemă importantă atunci când ne referim la sisteme ce sunt folosite în citirea textelor pentru copii, spre exemplu, ce au nevoie să audă o voce cât mai dinamică și plăcută.

Următorul bloc și ultimul îl constituie **blocul de sintetizare al discursului**. Există 3 metode principale:

1. **Metodă articulară**
   Această metodă presupune o modelare matematică dinamică, variabilă în timp a tubului articulator, deoarece fluxul de aer și el variază. Practic se simulează modul fizic de a crea sunete.

2. **Metodă parametrică**
   Această metodă pornește de la ideea că putem genera discurs atâta timp cât cunoaștem frecvența și banda fiecărui eveniment vocal. Problema apare atunci când observăm că discursul, nu este un semnal static, ci este unul dinamic. Astfel fiecare sunet, eveniment sonor depinde unul de celălalt. Se vor folosi metode precum filtrări paralele sau modele marcov ascunse (HMM).

3. **Metodă prin concatenare**
   Această metodă presupune preinregistrarea unităților de semnale vocale. Gradul de mărime al unității depinde de la aplicație la aplicație. Spre exemplu pentru un sistem de transcriere în vorbire utilizat în gara de tren, cea mai mică unitate o reprezintă descompunerile în mii, zeci și unități ale numerelor ce alcătuiesc numărul de tren. Totuși pentru aplicații generale, cea mai mică unitate o reprezintă grupurile de sunete (diftongi, triftongi). Diftongii și triftongii sunt cei care alcătuiesc silaba oricărui cuvânt. Astfel putem genera o infinitate de cuvinte, ținând cont de tranzițiile consoană, vocală. Dezavantajul acestei metode, este faptul că vom ignoră elementele de prozodie (partea melodică a semnalului vocal uman), iar discursul generat va suna robotic, artificial. [32]

2.3.3 Metode utilizând tehnici de învațare adâncă

Am discutat mai sus despre sistemele clasice de transcriere a textului în vorbire și am observat că există 2 părți fundamentale în această transcriere:

1. Prelucrarea textului

2. Sintetizarea de voce

În continuare vom aborda metodele cele mai cunoscute, utilizând rețele neurale, pentru a rezolva problema *TTS*. Noțiunile despre rețele neurale vor fi aprofundate in subcapitolul următor.

WaveNet

Dintre cele 2 părți, prima ce a avut parte și de o soluționare utilizând tehnici de învățare adâncă, a fost Sintetizarea de Voce. Soluția a venit din partea celor de la *Google* cu rețeaua numită *Wavenet*. Acest tip de rețea este bazat pe modelul *PixelCNN* și este capabil să producă semnale audio extrem de apropiate de cele ale omului. Este un model probabilistic și autoregresiv. Fiecare eșantion de semnal este condiționat de eșantionul anterior. Probabilitățile conditionale sunt modelate de un număr mare de straturi adânci de rețele convoluționale cauzale.
Utilizarea de convoluții cauzale în această arhitectură ne asigură că aceste model ține cont de ordinea în care datele sunt modelate. În acest model, fiecare eșantion prezis este pus la intrare pentru a ajuta la prezicerea următorului eșantion. Deoarece convoluțiile cauzale nu au conexiuni recurente, sunt mai rapide decât rețelele neurale recurente. Una dintre problemele majore folosind convoluții cauzale, este utilizarea unui număr mare de straturi pentru a crește câmpul de recepție. Pentru a rezolva această problemă, autorii au ales să folosească în a doua etapă, convoluții dilatate pentru a permite rețelei să aibă un câmp receptiv mai larg, folosind mai puține straturi. Stratul dilatat permite ca rețeua sa sară peste anumite valori de la intrare cu un anumit pas, în funcție de gradul dilatării. Cu cât dilatarea e mai mare cu atât se obține o corespondență între un șir mai mare de cuvinte. Modelarea distribuțiilor de probabilitate condiționate ale fiecărui eșantion se face folosind funcția softmax la iesire. Rezultatele au fost unele foarte bune de la limba engleză pana la limba chineză [33].

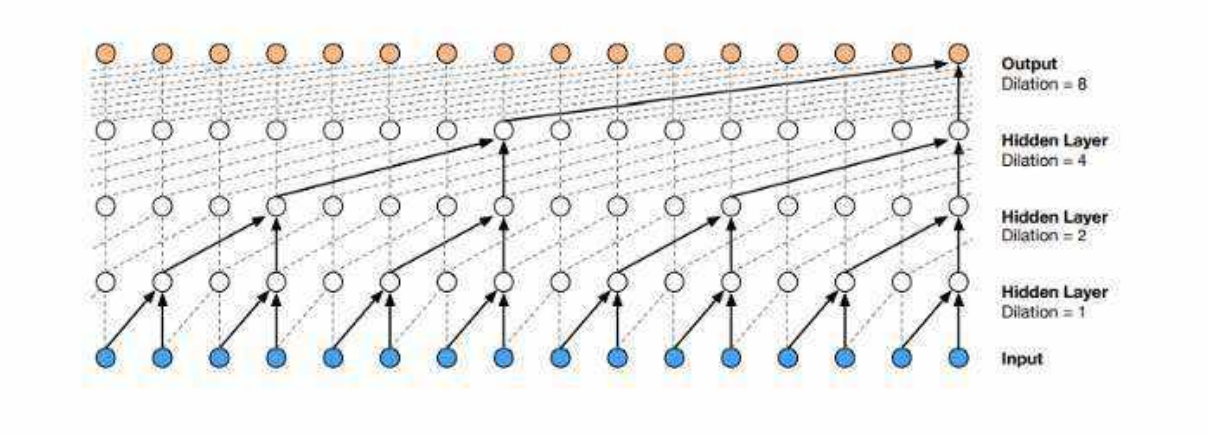

Figura 2.29: Vizualizarea convoluțiilor dilatate a rețelei WaveNet

Deep Voice

După apariția lui WaveNet apare și primul model de rețea neurală pentru întregul sistem de transcriere în vorbire. Aceste este modelul *Deep Voice 1*, creat de cei de la *Baidu's Silicon Valley Artifical Intelligence Lab*. Acest model are structura asemănătoare cu modelele clasice de transcriere. Este alcătuit din 5 blocuri, fiecare dintre ele fiind antrenat separat iar la final unite pentru a obține voce [34].

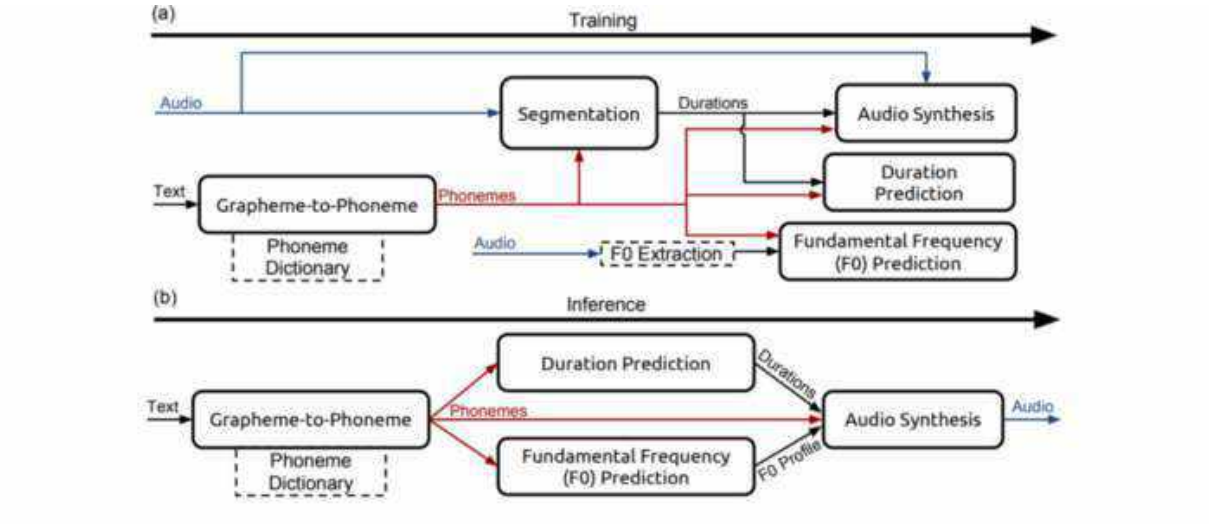

Figura 2.30: Schema bloc a retelei DeepVoice

Primul bloc este cel de conversie a textului în transcriere fonetică pentru a genera un model de pronunție a cuvintelor.

Din acest bloc se ramifică următoarele 4 blocuri.

- Blocul de segmentare ce localizează delimitările între sunete/foneme sau mai precis unde începe și se termina fiecare fonem în semnalul audio
- Blocul de predicție a frecvenței fundamentale corespunzător pentru fiecare fonem
- Blocul de predicție a duratei pentru fiecare fonem
- Blocul de sintetizare a vocii (ce folosește o rețea asemănătoare cu *WaveNet*)

Tacotron

Următorul model de rețea este primul model *End to End (eng)* pentru sistemele de transcriere din text în vorbire. Prin urmare, nu va mai fi nevoie sa antrenam separat mai multe blocuri pentru a obține rezultatul final. Ci întreaga rețea este un singur bloc ce se ocupa atât de partea de prelucrare de text cat și de partea de sintetizare de voce.

Tacotron generează voce la nivel de secventa (*frame (eng)*) fiind mult mai rapid decât modelul autoregresiv de generare utilizat de WaveNet și DeepVoice.

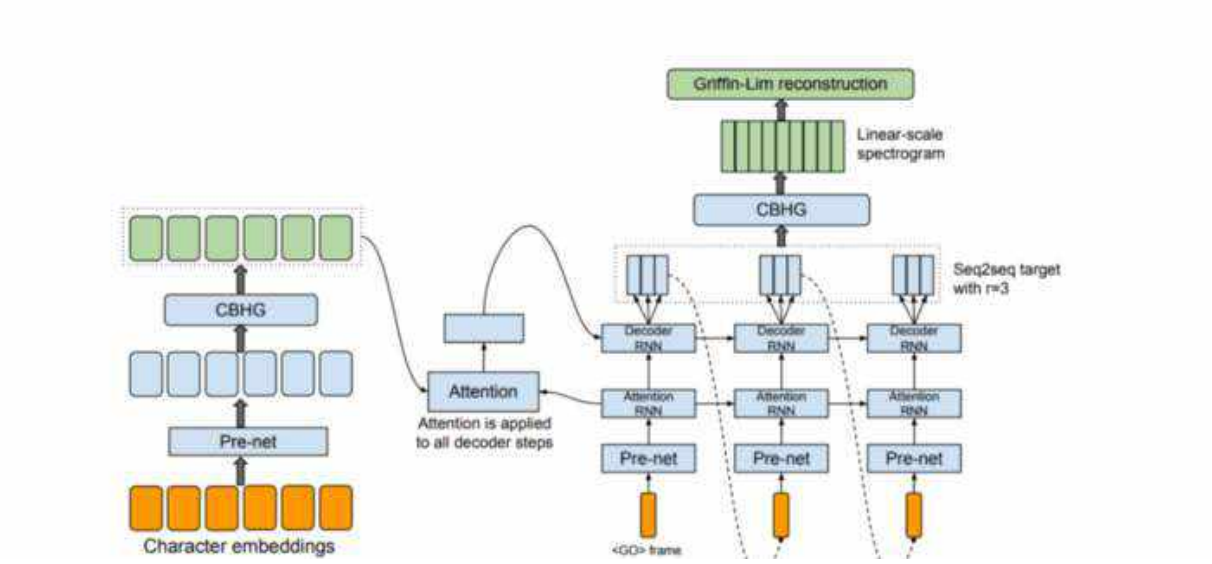

Figura 2.31: Schema bloc a rețelei Tacotron

Modelul este antrenat pe perechi de audio, text, ceea ce face această rețea ușor de adaptat la baze de date noi. Este un model secvență la secvență (*sequence to sequence*) ce funcționează pe arhitectura Codor, Atenție, Decodor, întâlnită și la rețelele de Traducere. Modelul primește la intrare caractere (litere) și oferă la ieșire o Spectrogramă liniară. Spectrograma este apoi convertită în semnal audio folosind algoritmul *Grifin Lim* [35].

Partea foarte importantă din acest model, o constituie modulul *CBHG* ce este alcătuit dintr-un șir de filtre convoluționale unidimensionale, din conexiuni reziduale (*Highway networks (eng)*) și o rețea bidirecțională de tip *GRU* (*Gated Recurent Unit (eng)*) [36].

Codorul este cel care extrage trăsăturile de text și vor fi codate. Acestea și secvențele de spectrogramă vor fi oferite ca intrare în decodor. Atenția ne va ajuta să păstrăm ordinea logică a sunetelor atunci când reproducem vocea [37].

2.4 Rețele neurale în NLP

Introducere *Deep Learning*

Pentru a înțelege modul de funcționare a rețelelor neurale adânci este important sa înțelegem următoarele definiții:

- Un sistem inteligent este un sistem capabil să preia informația, să o prelucreze, să o analizeze și să ia *decizii*.
- O rețea neuronală artificială reprezintă un sistem adaptiv capabil să furnizeze răspunsuri pentru un anumit context necunoscut, după ce a fost antrenată pentru situații similare.
- Procesul de învățare este procesul de ajustare corespunzătoare a unor parametrii asociați unităților funcționale, în scopul obținerii de răspunsuri corecte la ieșire, pentru diverse valori (necunoscute) de intrare.

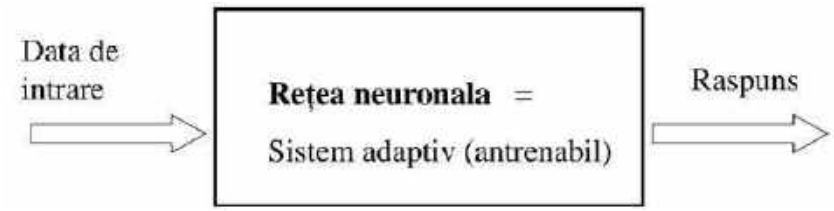

Figura 2.32: Schema bloc a unei rețele neurale

Neuronal artificial reprezintă o modelare matematică a unui neuron biologic ce calculează diverse ieșiri în funcție de parametrii de intrare.

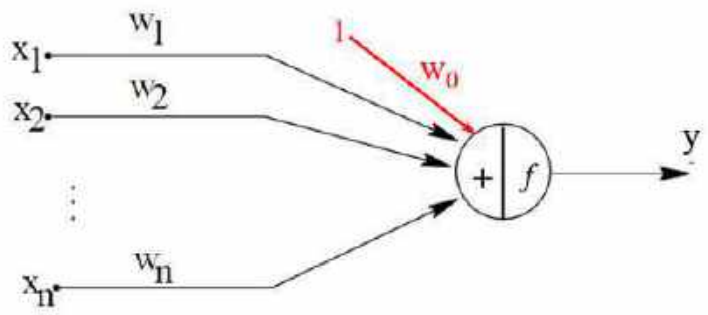

Figura 2.33: Neuronul artificial (*Adaline*)

$$y = w_0 + \sum w_i x_i \tag{2.30}$$

Funcțiile de activare utilizate sunt :

- ReLu (*Rectified Liniar Unit*)
- Softmax

ReLu este o funcție de activare liniară preferată în domeniul rețelelor neurale. Împărțirea binară (utilizată de modelul perceptron cu funcția *sigmoid*) nu oferă o informație calitativă și cantitativă, ci consideră valorile aproape de pragul de împărțire la fel de importante ca cele

depărtate. Noi însă dorim să știm cât de bine evaluează rețeaua ieșirile și să facem o apreciere mai corecta a lor. Acest lucru este posibil utilizând funcția liniară *ReLu*.

Softmax teoretic nu este o funcție de activare a unui singur neuron ci este o metoda de normalizare a ieșirilor dintr-o rețea neurala. Noile ieșiri vor varia intre (0,1), iar suma lor va fi 1. Prin urmare, ieșirile sunt transformate în mărimi echivalente cu probabilități. Această funcție se utilizează de obicei în probleme de clasificare pentru a genera la ieșire o probabilitate pentru fiecare element și a se alege elementul cu probabilitatea cea mai mare.

Fiind dat un lot de învățare (perechi intrare-iesire) *(xi, yi)* $i = 1, M$, scopul rețelei va fi deci să determine acele ponderi *w0, w1, w2, ... , wn* care minimizează funcția de cost(eroare) dintre *xi* și *yi*.

Rețelele adânci reprezintă acele rețele ce conțin un număr semnificativ de straturi de neuroni. Scopul straturilor multiple este de a învăța sistemul în plus să extragă singur trăsăturile importante ale vectorilor de intrare. [38]

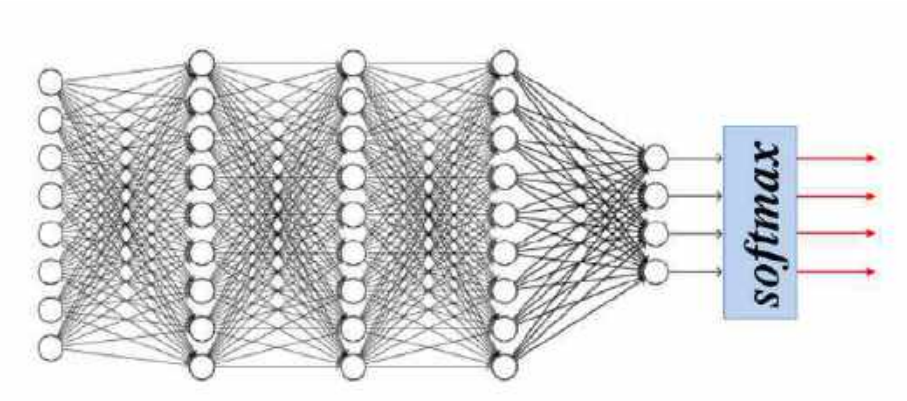

Figura 2.34: Rețea neurală adâncă

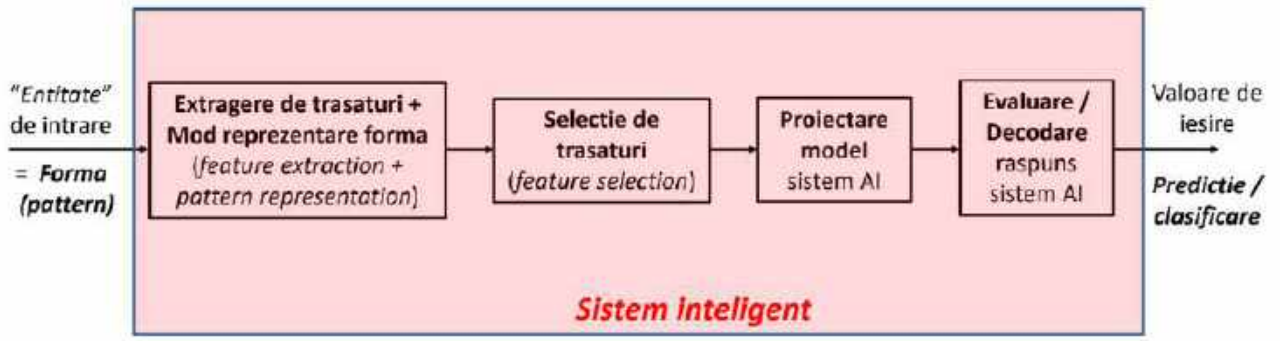

Figura 2.35: Reprezentarea bloc a rețelelor neurale adânci

Named Entity recognition

Presupunem că avem un corpus mare de text și dorim să clasificăm fiecare cuvânt în tipul pe care îl reprezintă : locație, persoană etc.

Vom considera o fereastră de selecție pentru un număr de cuvinte și vom clasifica ce se află în mijlocul selecției. Fiecare cuvânt va avea câte un vector reprezentativ. Vom concatena vectorii într-unul singur și vom clasifica. Prin această metodă vom păstra și informație despre poziție. Știm că respectivul cuvânt ce vrem să-l clasificam se află în mijloc.

În cazul de față s-au luat vectorii corespunzători fiecărui cuvânt ce sunt de dimensiune 4. În consecință, vectorul concatenat la intrare va avea o dimensiune de *(20 X 1)*.

Urmează stratul ascuns ce oferă la ieșire un al doilea vector. Acest vector poate conține informații despre interacțiunile non liniare între cuvintele de la intrare. Dacă spre exemplu, primul cuvânt este *muzeu* și al doilea e o prepoziție, asta ar trebui sa ofere rețelei un indiciu că urmează o locație în mijlocul vectorului.

Putem observa deci, faptul că straturile ascunse ne ajută să calculăm interacțiuni/legături noi între cuvinte.

Intrările în rețea pot fi atât *embedding*-uri preantrenate cât și inițializate aleator și învățate pe parcurs de către rețea [39].

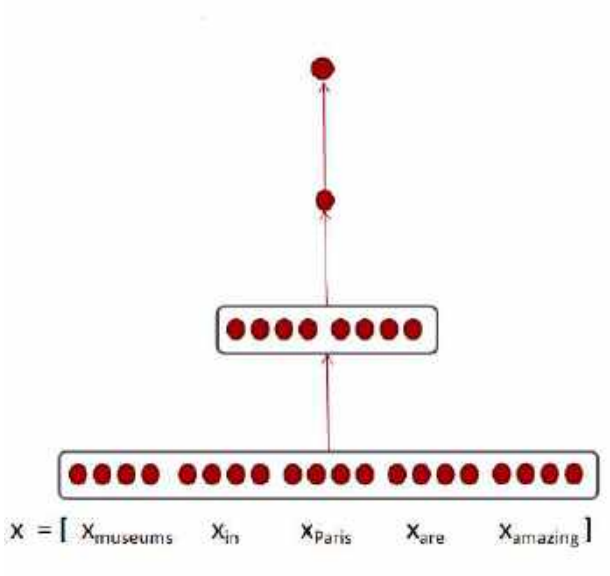

Figura 2.36: Solutie neurala a problemei *Named Entity recognition*

Modele pentru dependențe structurale

Pentru rezolvarea problemei se va considera urmatorul model: [40]

- Vom avea o stivă și un buffer.
- Fiecare cuvânt va avea o reprezentare de vector
- Fiecare etichetă pentru părțile de propoziție corespunzătoare unui cuvânt va avea o reprezentare de vector
- Fiecare etichetă pentru dependențe va fi și el reprezentat de un vector. Acești vectori se vor aduna și va rezulta un vector cu toate informațiile necesare.

- Se vor lua toți vectorii din stivă și se vor concatena și vor fi trecuți printr-un strat ascuns și un strat de ieșire cu 3 clase (*right arc, left arc, aducere*).

Ieșirea ne va dicta ce operație să facem mai departe :

- pentru operația de *right arc* înseamnă că am găsit o dependență față de cuvântul din dreapta și îl vom elimina din stivă
- pentru operația de *left arc* înseamnă că am găsit o dependență față de cuvântul din stânga și îl vom elimina din stivă
- pentru *aducere* vom mai aduce un cuvânt din buffer

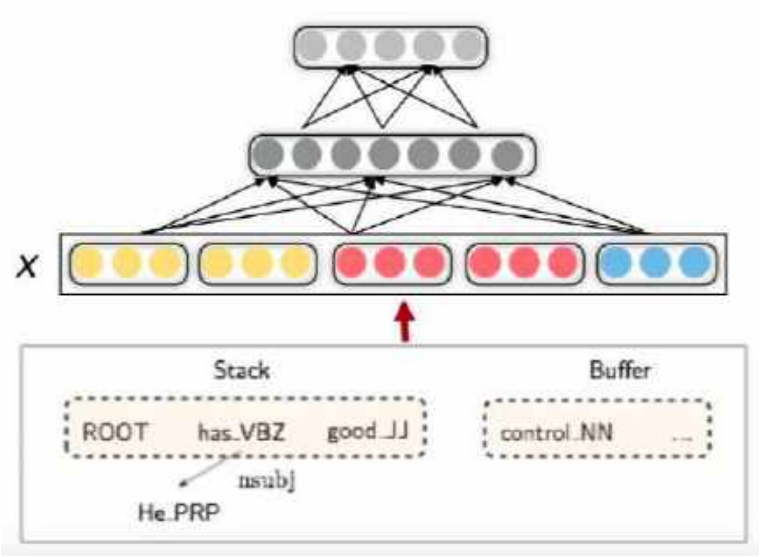

Figura 2.37: Parsare neurală a dependențelor cuvintelor

Rețele neurale recurente pentru modele de limbă

Pentru problemele de modelare a limbii putem aplica metoda ferestrei de alegere al unui context (N-grame). Fiecare cuvânt va avea un vector reprezentativ. Vom concatena toate cuvintele și le vom trece printr-un strat ascuns. La stratul de ieșire vom avea o funcție Softmax. Ieșirea va fi de lungime v unde v este vocabularul de cuvinte utilizat. Astfel aplicând o funcție Softmax pe un strat de lungime v, vom avea o distribuție de probabilitate pentru fiecare cuvânt din vocabular. Vom putea lua deci o decizie pentru alegerea următorului cuvânt. (vezi fig. 2.38).

Metoda aceasta a ferestrei este totuși limitată deoarece ne oferă doar o dimensiune fixă de prelucrare a cuvintelor. Avem nevoie de o rețea ce poate procesa orice dimensiune de intrare. Astfel sunt introduse *rețelele neurale recurente*(RNN).

În acest tip de rețea, fiecare ieșire a stratului ascuns este calculată în funcție de ieșirea anterioară și de intrarea curentă. Vom avea aceeași matrice de ponderi W ce se aplică pe fiecare dintre stările (iterațiile) modelului. Ultimul strat (ultima iterație) va fi intrarea în stratul de ieșire pe care vom aplica funcția Softmax pentru creearea distribuției de probabilitate pentru alegerea cuvântului următor. (vezi fig. 2.39) [41].

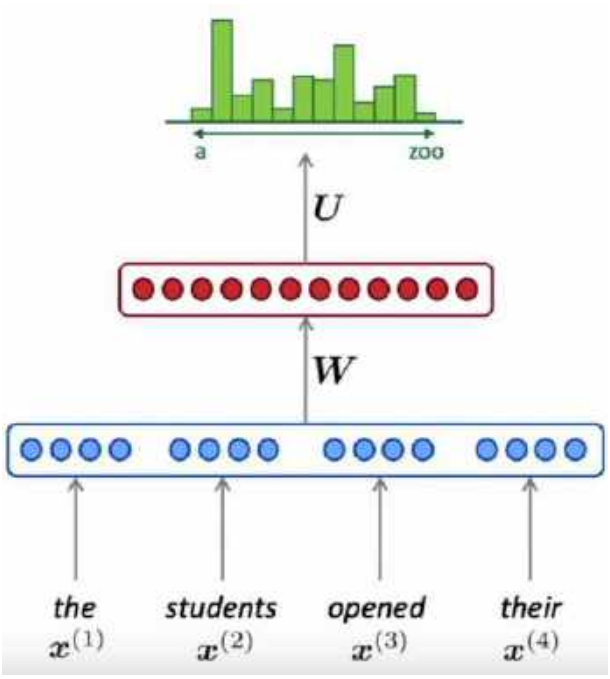

Figura 2.38: Straturi neurale clasice in realizarea modelului de limba

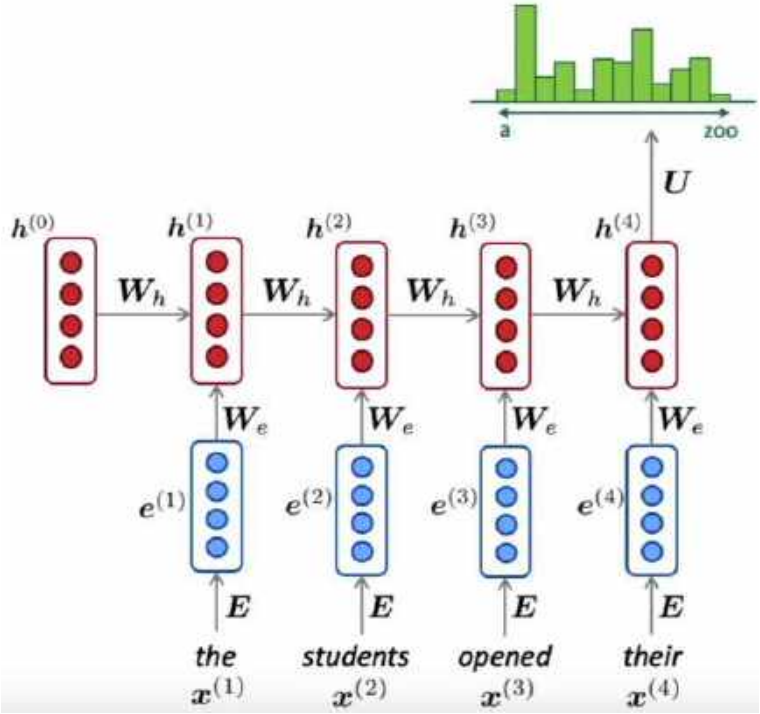

Figura 2.39: Rețele recurente în realizarea modelului de limbă

Avantajele acestor tip de rețele sunt :

- putem procesa oricât de multe cuvinte la intrare
- se folosește informația distribuită de-a lungul tuturor iterațiilor(avem acces la informațiile pașilor predecesori)
- dimensiunea modelului nu crește pentru cuvinte mai multe
- eficiență în utilizarea aceleași matrice de ponderi aplicate la fiecare *timestep*

Principalul dezavantaj îl constituie viteza acestui model. Acest tip de rețea este foarte lent. Mai mult, deși putem accesa informații de la pașii anteriori, la un moment dat, informația se poate pierde datorită dispariției gradientului (*the vanishing gradient problem*). Acest lucru se fixează utilizând RNN-uri de tip GRU (*Gate Recurrent Unit*)

Una dintre aplicațiile foarte întâlnite ale RNN-ului, în afara de modelarea de limba, este codarea textului (Encoder module). Putem genera ieșirea fiecărei iterații, și astfel avem o reprezentare a contextului, o informație generală despre fiecare cuvânt.

Am menționat mai sus problema dispariției gradientului (*the vanishing gradient problem*). Dacă avem secvente prea lungi valoarea gradientului, ce reprezintă variația erorii cu ponderile, poate deveni foarte mică și nu mai avem cum să actualizăm ponderile relativ la toți pașii anteriori. Soluție este următoarea:

Ne vom gândi la un model de rețea recurentă ce are **memorie**. Un model ce poate păstra idei pe care doresc sa le accesez mai târziu. Astfel apar rețelele LSTM (*Long Short Term Memory*) si GRU (*Gate Recurent Unit*) ce sunt un tip de rețele recurente. În cazul nostru vom folosi doar GRU.[42]

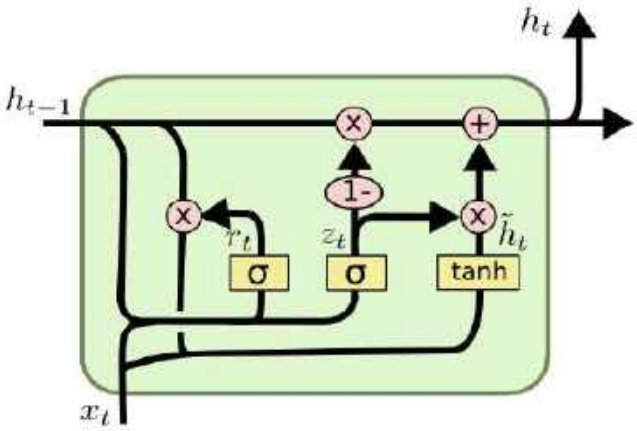

Figura 2.40: GRU (*Gate Recurrent Unit*

Acest model conține porți pentru a controla fluxul de informații.

- poarta de actualizare (*update gates* $\boldsymbol{z}^{(t)}$) ce controlează ce informații din straturile ascunse să fie actualizate și care sa fie păstrate
- poarta de resetare (*reset gates* $\boldsymbol{r}^{(t)}$) ce controlează care parte din stratul anterior ascuns va fi folosită ca să creăm noua ieșire a stratului ascuns. Prin urmare selectăm care părți din iterația anterioară sunt folositoare și care nu.

- conținutul stării următoare (*new hidden state content* $\tilde{\boldsymbol{h}}^{(t)}$) este calculat utilizând poarta de resetare. Aceasta selectează părțile utile ale stării anterioare. Se folosește de asemenea și conținutul intrării curente
- se calculează noua stare ($\boldsymbol{h}^{(t)}$) utilizând atât poarta de actualizare ($\boldsymbol{z}^{(t)}$), ce selectează ce se păstrează din starea anterioara, cât și conținutul nou format ($\tilde{\boldsymbol{h}}^{(t)}$)

$$\boldsymbol{z}^{(t)} = \sigma\left(\boldsymbol{W}_z \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_z \boldsymbol{x}^{(t)} + \boldsymbol{b}_z\right) \tag{2.31}$$

$$\boldsymbol{r}^{(t)} = \sigma\left(\boldsymbol{W}_r \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_r \boldsymbol{x}^{(t)} + \boldsymbol{b}_r\right) \tag{2.32}$$

$$\tilde{\boldsymbol{h}}^{(t)} = \tanh\left(\boldsymbol{W}_h \left(\boldsymbol{r}^{(t)} \circ \boldsymbol{h}^{(t-1)}\right) + \boldsymbol{U}_h \boldsymbol{x}^{(t)} + \boldsymbol{b}_h\right) \tag{2.33}$$

$$\boldsymbol{h}^{(t)} = \left(1 - \boldsymbol{z}^{(t)}\right) \circ \boldsymbol{h}^{(t-1)} + \boldsymbol{z}^{(t)} \circ \tilde{\boldsymbol{h}}^{(t)} \tag{2.34}$$

Mai sus au fost scrise relațiile de calcul ale porților de control. Parametrii de tip W exprimă ponderile de actualizare, resetare, conținut ale stării anterioare $\boldsymbol{h}^{(t-1)}$, iar parametrii de tip U exprima aceleași ponderi însă ale stării curente $\boldsymbol{x}^{(t)}$. De asemenea valorile b reprezinta *bias-ul* acestor porți [36].

Vor fi momente când vom dori să avem informații atât din stânga cât și din dreapta cuvântului. Atunci motivația noastră va fi să utilizăm RNN bidirecționale. Modul de funcționare este prin rularea în paralel a unui *forward* RNN (analizează cuvintele de la stânga la dreapta) și a unui *backward* RNN (analizează cuvintele de la dreapta la stânga). Fiecare RNN are matricea lui proprie de ponderi. La ieșirea unei stări dintr-un astfel de model, un cuvânt va avea de 2 ori mai multă informație (un vector de 2 ori mai mare) deoarece conține informație atât despre vecinul său din dreapta cât și despre cel din stânga. [43]

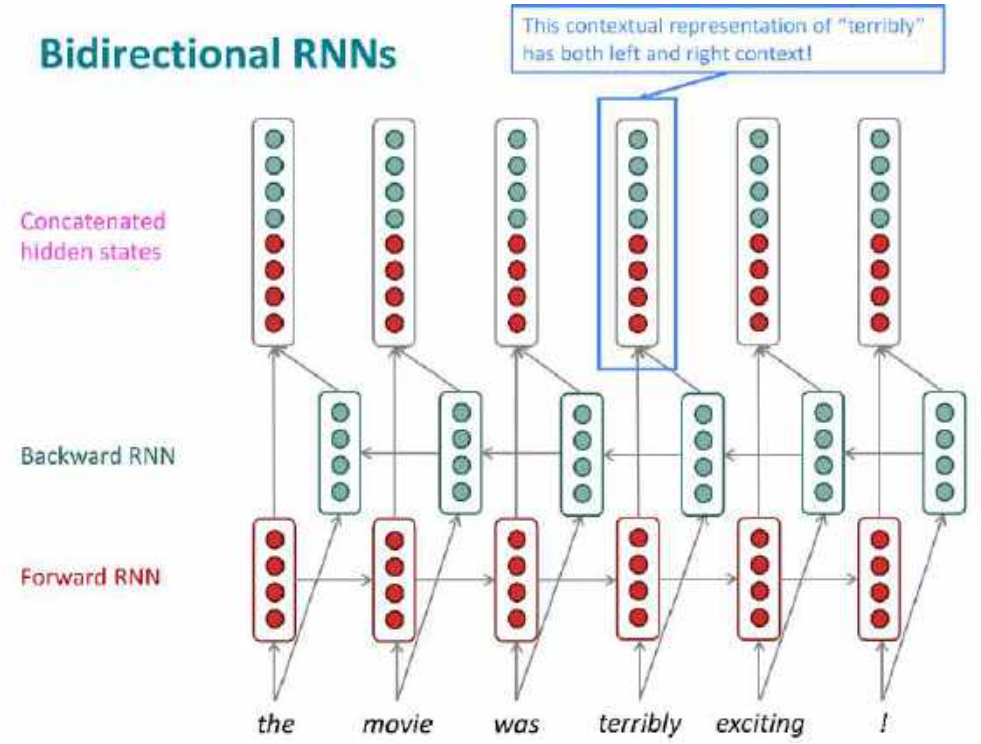

Figura 2.41: Rețele neurale recurente bidirecționale

Rețele Convoluționale în NLP

Dezavantajul utilizării RNN-urilor în analiza textului este că RNN ne oferă informații doar succesive. Fiecare cuvânt depinde de cel din spate. Putem folosi și un RNN bidirecțional, dar se păstrează convenția de a avea informații doar despre legăturile vecine dintre cuvinte. Ne interesează să găsim legături și între oricare combinații dintre cuvinte. Vom renunța la metoda *seq2seq* în favorul rețelelor Convoluționale. Avantajul convoluțiilor este că ne prelucrează mai multe tipuri de combinații între cuvinte și se pot găsi mai ușor legăturile între ele.[44]

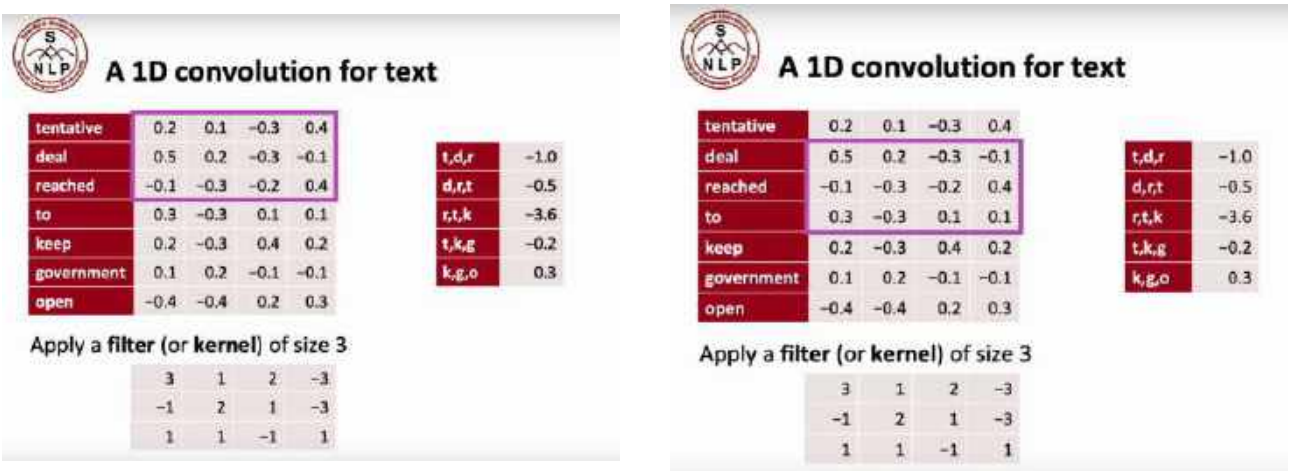

Figura 2.42: Calculul convoluției pe o secvență

Observăm că dimensiunea embedding-ului oferă numărul de canale, iar din numărul de cuvinte ce doresc să le prelucrez împreună rezultă dimensiunea kernelu-lui. Se aplică un kernel/filtru peste combinația de cuvinte, obținem un rezultat și continuăm să trecem cu filtrul peste toate combinațiile de cuvinte.

De asemenea, mai putem aplica și tehnica *padding* (adăugăm cuvânt cu zerouri la începutul și la finalul secvenței) pentru a putea lua în considerare și relațiile între primele respectiv ultimele 2 cuvinte.(vezi fig.2.43). Dacă doresc sa am mai multe canale la ieșire, voi aplica mai multe kerneluri (vezi fig.2.44).

După ce obțin rezultatul, voi aplica *max pooling* și astfel pot să găsesc combinația cea mai semnificativă de cuvinte, ce îmi poate spune spre exemplu dacă propoziția are sau nu un caracter politicos. (vezi fig.2.45).

O altă tehnică utilă este dilatarea. Aceasta presupune realizarea de salturi între combinații. Decât să măresc kerneul și să am mai mare procesarea de calcul, mai bine fac salturi și voi cuprinde tot la fel de multe perechi de cuvinte. Astfel nu se vor mai combina *1 - 2 - 3* ci *1 - 3 - 5* . (vezi fig.2.46) .

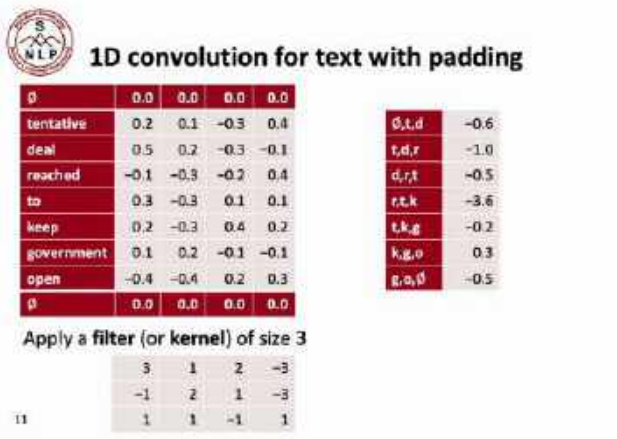

Figura 2.43: Tehnica *padding*

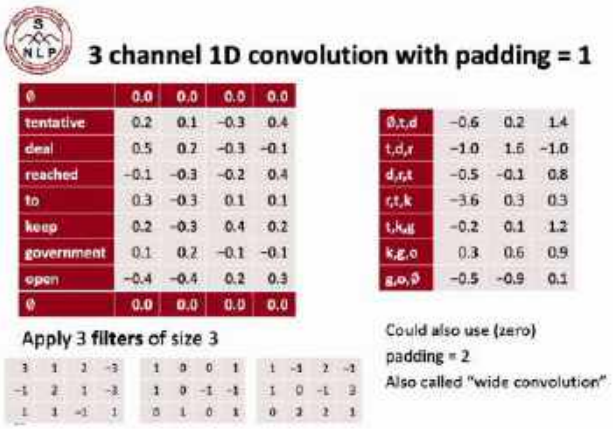

Figura 2.44: Utilizarea mai multor *kernele*

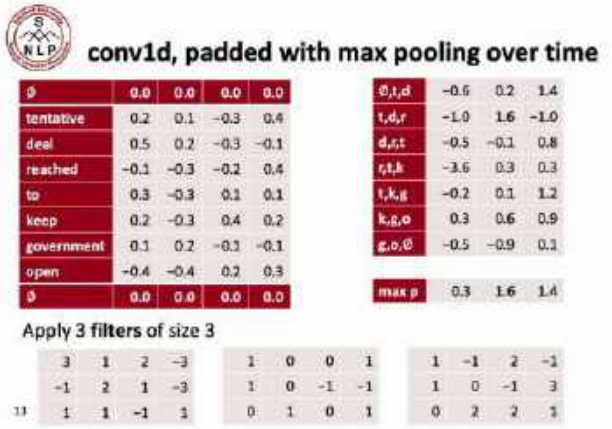

Figura 2.45: Tehnica MaxPooling

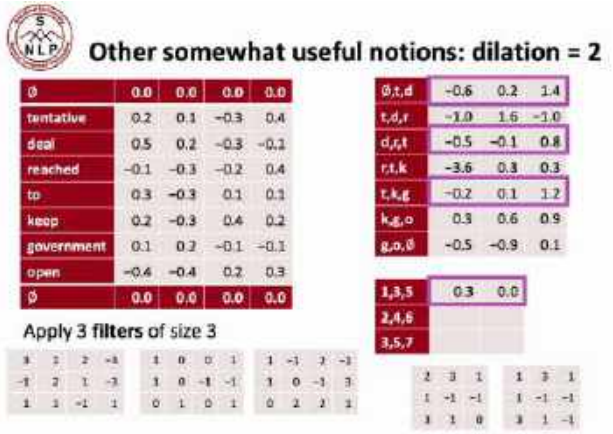

Figura 2.46: Convolutii Dilatate

În fig.2.47 putem observa un model de rețea neurală ce folosește convoluții pentru a clasifica un text ca fiind fie *pozitiv* fie *negativ* în exprimare. La intrare avem o secvență de 7 cuvinte, fiecare cuvânt având un embedding de dimensiune 5. Se vor aplica mai multe tipuri de ferestre pe matricea de la intrare. Avem câte 2 ferestre (kernele) de dimensiune 2, 3 si 4. Cele de dimensiune 2 au la ieșire un vector de dimensiune 6, cele de 3 au un vector de dimensiune 5, iar cele de 4 au un vector de dimensiuni 4. Pe fiecare vector de ieșire se va aplica un *maxPooling* și se va concatena cu perechea sa. La final, toate perechile se vor concatena într-un singur vector pe care se va aplica o funcție de Softmax pentru o ieșire cu 2 clase. Putem astfel să clasificăm textul ca fiind unul *pozitiv* sau *negativ* ca ton.

Observăm deci faptul că rețelele convoluționale extrag trăsături mai bune decât RNN. Apare deci motivația să folosim CNN-uri pentru codarea textului. De asemenea, sunt și mult mai rapide ca RNN-uri deoarece pot paraleliza acțiunea, aplicând aceeași fereastră/filtru pe intrări. La RNN fiind calcul secvențial, fiecare stare e diferită și depinde una de cealaltă. [45]

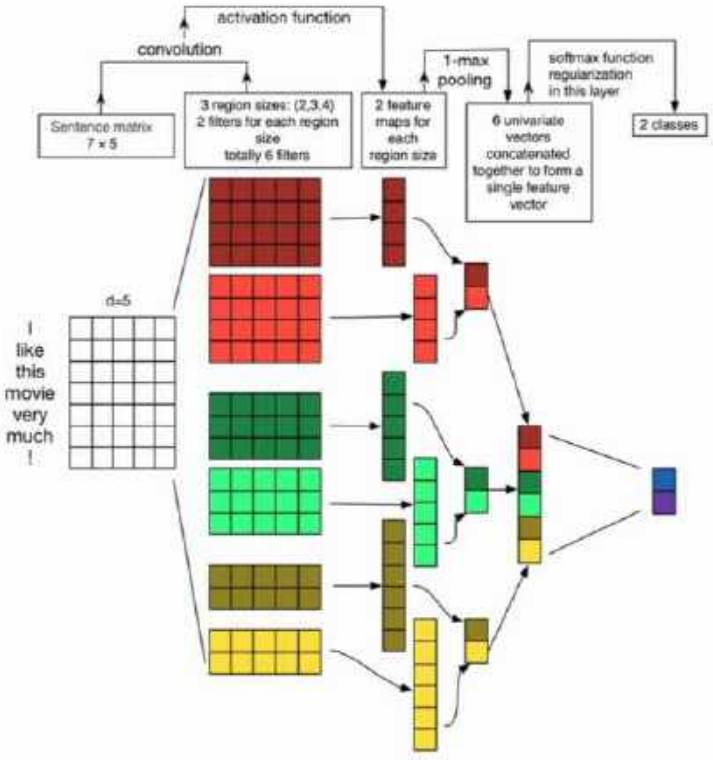

Figura 2.47: Retea neurala ce utilieaza convolutii

Traduceri, Secventa la Secventa, Atentie

Soluția la problema traducerii apare utilizând arhitectura de rețele *seq2seq (eng).* Mapăm o secvență dintr-un RNN (propoziția sursă) la altă secvență dintr-un RNN separat (propoziția țintă).[46]

Ce se întamplă la testare :

Observăm că în partea de testare se oferă la intrarea codorului secvența sursa în limba franceză (vezi fig 2.48). Aceasta este codată de rețeaua recurentă iar ieșirea este dată ca stare inițială în decodor. Prima intrare în decodor este caracterul de $< START >$ ce indică începutul secvenței țintă. Decodorul va trebui sa știe sa prezică primul cuvânt din secvența țintă bazându-se doar pe codarea propoziției sursă. După, ieșirea primei iterații se oferă ca intrare în următoarea stare a decodorului, acesta trebuind să prezică următorul cuvânt bazându-se pe noua intrare (ce a fost prezisă anterior) și starea anterioară a rețelei. Procesul se repetă iterativ până când rețeaua va prezice caracterul $< END >$ ce reprezintă finalul propoziției. (vezi fig.2.48).

Arhitectura secvență la secvență oferă un model de limbaj condiționat. Decodorul reprezintă partea de modelare a limbii (prezicerea următorului cuvânt y). Predicțiile lui y sunt date de sursa x codată (*encoder module*), lucru ce semnifică partea de condiționare. Observăm că modelul neural calculează direct $P(y|x)$, nemaifiind nevoie să împărțim această sarcină în 2 probleme diferite.

Ce se întâmplă la partea de antrenare :

Oferim la intrarea Codorului propoziția sursă, și se iterează cuvânt cu cuvânt. Oferim la

intrare în Decodor propoziția țintă (limba în care vrem să traducem) și se iterează cuvânt cu cuvânt ținând cont de ultima ieșire din codor. La decodor fiecare ieșire prezice probabilitatea următorului cuvânt, iar fiecare eroare (notată cu J în imagine) este calculată la fiecare iterație, relativ la intrarea următoare (se verifică cât de bine a prezis intrarea 1 faptul că urmează intrarea 2 - vezi fig.2.49).

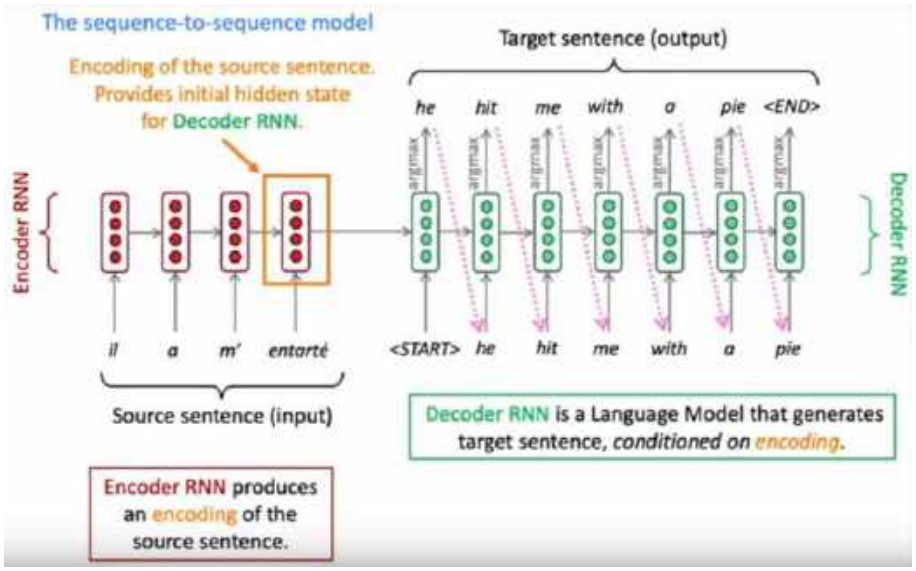

Figura 2.48: Problema traducerii în faza de testare

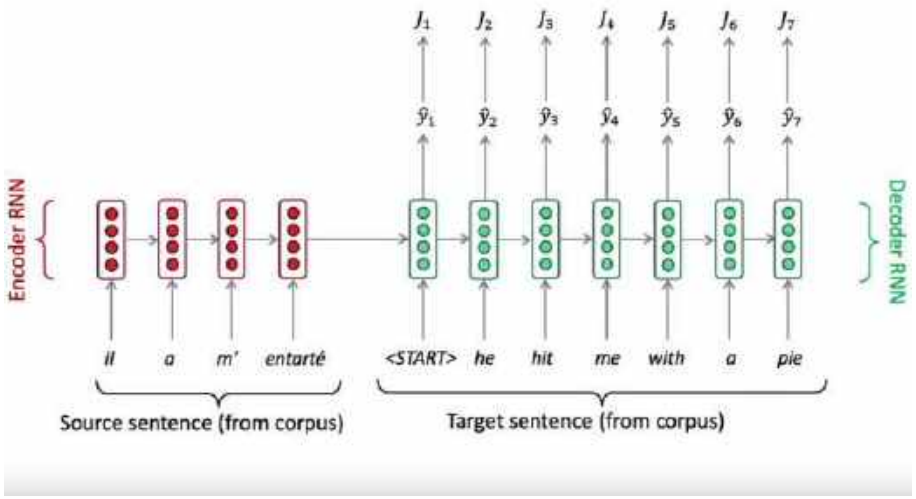

Figura 2.49: Problema traducerii în faza de antrenare

Atenția

Toată informația pentru toată propoziția sursă este păstrată într-un singur vector (ieșirea stratului ascuns după ultima iterație). Aceasta constituie o problemă. Apare ceea ce se numește *information bottleneck* (se pune mult prea multă presiune pe un singur vector). O soluție este utilizarea atenției. Aceasta presupune ca fiecare iterație din decodor sa aibă o conexiune directă către codor, focusându-se doar pe partea importantă din propoziția sursă. Astfel avem și o asigurare a alinierii mai bune a informației.

Pentru fiecare iterație ni se va returna un scor de atenție ce reprezintă produsul dintre fiecare ieșire corespunzătoare cuvintelor ce aparțin secvenței sursă și cuvântul curent din decodor.

Rezultatul obținut va fi un vector pe care îl vom trece printr-un SoftMax, pentru a obține o distribuție a probabilității pentru fiecare cuvânt. Se face apoi o sumă ponderată și vom avea după, informație despre stările cu probabilități mai mari. Vom selecta apoi cuvântul pe care pică probabilitatea mai mare de influența a predicției următoarei ieșiri din decodor.[47]

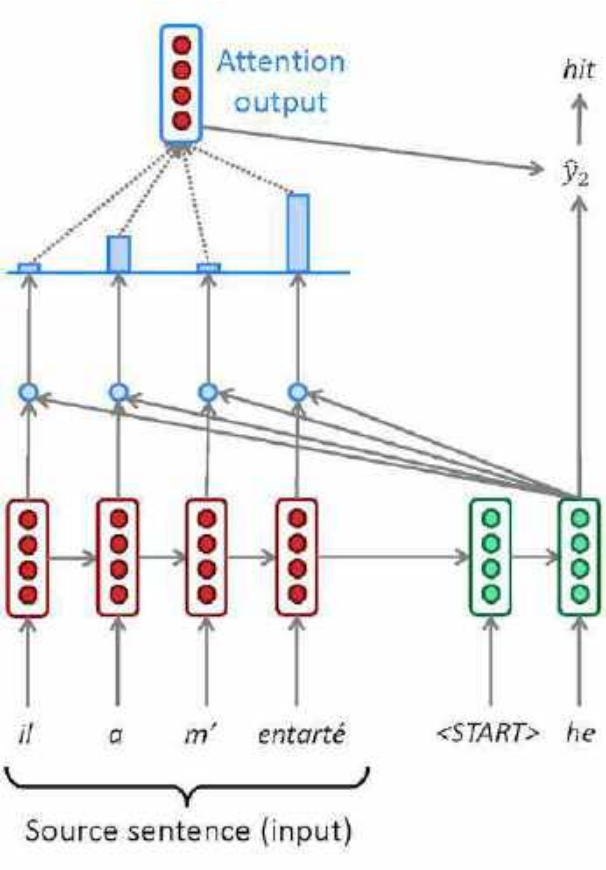

Figura 2.50: Model de atenție

Concluzii atenție:

- atenția ajută performanțele de calcul
- ajută problema *bottleneck (eng)*
- ajută problema dispariției gradientului
- ajută la o aliniere mai bună între secvențe

Capitolul 3
Setup Experimental

3.1 Baze de date

Pentru sistemele de traducere este nevoie de utilizarea unor baze de date ce conțin perechi de tipul (text, înregistrare). Înregistrările trebuie să fie cât mai scurte pentru a găsi mai ușor dependențele între cuvinte. Cu cât înregistrările sunt mai lungi cu atât și textul specific lor este mai lung. În consecință, nu doar prelucrarea la nivel de text va fi mai dificilă (găsirea de dependențe între sunete), dar și generarea de eșantioane de spectrogramă va fi (task-ul decodorului de modelare a sunetului ce presupune preziceri de eșantioane relativ la eșantioanele anterioare). Prin urmare, este important să avem o bază de date ce conține înregistrări cât mai scurte (între 3 și 10 secunde). Mai mult, înregistrările/textele trebuie să conțină o varietate cât mai mare în vocabular ca să se cuprindă cât mai multe combinații sonore specifice limbii. Pentru îndeplinirea acestei sarcini am ales 4 baze de date, una în limba engleză și trei în limba română, ultima fiind o baza de date proprie pentru a genera o voce asemănătoare cu vocea mea.

3.1.1 Baza de date *LJSpeech*

Prima bază de date pe care am utilizat-o a fost *LJSpeech Dataset*. Este o bază de date folosită pentru mai multe sarcini în domeniul procesării naturale a limbajului. Este o bază de date ce a fost folosită atât pentru soluții de sintetizare a vocii cât și pentru recunoaștere a vocii, obținând rezultate bune de fiecare dată. În interesul nostru este partea de sintetizare a vocii, mai precis convertirea din text în voce. Este o bază de date *open source* [48], utilizată de majoritatea cercetătorilor în acest domeniu. Textele sunt luate din domeniul public, apărute în 1884 și 1964. Înregistrările au fost făcute în 2016 de către proiectul *LibriVox* și făcute disponibile publicului.

Baza de date cuprinde 13.100 de clipuri audio de la un singur vorbitor, ce citește 7 cărți non-ficționale. Pentru fiecare clip audio există o transcriere. Clipurile variază ca și durata între 1 și 10 secunde, iar în total formează 24 de ore de voce înregistrată. Textele au fost asociate manual către audio. Fiecare clip audio are un singur canal și are o codare pe *16 biți PCM WAV*, cu o frecvență de eșantionare de 22050Hz. De asemenea, textul a fost normalizat parțial, având deja numere, unități monetare și diverse abrevieri expandate în cuvinte întregi.

3.1.2 Romanian Speech Corpus (RSC)

Baza de date *RSC* (*Romanian Read Speech Corpus for Automatic Speech Recognition*) introduce cea mai mare bază de date în Română pentru aplicații în principal de recunoaștere vocală. Înregistrările reprezintă fragmente din literatură, știri, interviuri și cuvinte izolate. *RSC* a fost creată de către Laboratorul de Cercetare în Discurs și Dialog *SpeeD*. Înregistrările au fost efectuate în diferite condiții utilizând diferite microfoane și sisteme de înregistrare, utilizând o aplicație online de înregistrare dezvoltată de echipa de cercetare. Vorbitorii au fost în general studenți și angajați ai Facultății de Electronică Telecomunicații și Tehnologia Informației a Universității Politehnica din București.

Corpusul conține 136.120 de fișiere audio colectate de la 164 de vorbitori nativi de limba română, 107 dintre ei de gen masculin și 57 de gen feminin. Vârstele au variat între 19 și 60 de ani, cu o medie de 24 de ani. În general sunt între 130 și 11.000 de fișiere audio pentru fiecare vorbitor. Baza de date are în total 100 de ore de înregistrări, iar durata medie a unei înregistrări este de 2.6 secunde.

Lungimea fiecărei fraze variază de la un singur cuvânt la 50 de cuvinte. Multe dintre înregistrări conțin doar un cuvânt precum "steag", "gâștile". Aceste cuvinte au fost selectate pentru a acoperi lista tuturor combinațiilor de silabe în limba română. Reținem că aceasta este o baza de date pentru recunoaștere de voce, nu pentru sintetizare. Astfel nu va fi nevoie de un număr mare de cuvinte. Recunoașterea de voce este în general folosită pentru a trimite comenzi simple, de la diferiți vorbitori, nefiind în principal nevoie de o recunoaștere a unui text lung. În sintetizarea de voce ne dorim însă să producem fraze cât mai lungi și cât mai naturale și nu cuvinte scurte. Folosind această bază de date, vom observa că modelul nostru nu va învăța foarte bine să sintetizeze fraze lungi și să găsească legături între cuvinte (*problema Dependency Parsing*).

Alte propoziții au fost selectate din romane românești sau internaționale traduse în limba romana. Spre exemplu :

- "Minciuna a avut succes." selectata din romanul "Alchimistul" de Paulo Coelho
- "Se apropia sfârșitul anului și dascălul examina toată clasa". - "Viața ca o pradă" de Marin Preda.

De asemenea multe dintre propozițiile cele mai lungi au fost selectate din știri online precum: "Modul in care poate fi făcut acest lucru tine de fiecare guvern."

Resursele lingviste se pot împărți în două *corpus*-uri : 315.000 culese din știri și alt *corpus* conținând 40.000 de cuvinte din diverse transcripturi. Aceste corpusuri au fost alese pentru a obține *n-grame* pentru modelare de limbaj (*n-grams in language modelling*) utilizând *toolkit*-ul *SRI-LM* și după au fost interpolate cu o pondere de 0.5 cu același *toolkit*. S-au utilizat aceste n-grame probabilistice pentru a decoda și ajusta limbajul. Conform unor experimente [Georgescu et al.2017], s-a arătat ca pentru un bun raport performanta si resurse hardware, este nevoie sa utilizam 2-grame, în timp ce pentru *languge rescoring* este nevoie sa utilizam 4-grame. Se justifică astfel de ce baza de date a preferat înregistrarea unor clipuri ce conțin în general doar unul sau doua cuvinte.
RSC se împarte atât într-o parte de antrenare cât și într-una de evaluare precum urmează :

- setul de antrenare conține 133.616 de fișiere audio de la 156 de vorbitori
- setul de evaluare conține 2504 de fișiere de la 21 de vorbitori (dintre care 13 vorbitori fac de asemenea parte din setul de antrenare)

De asemenea, baza de date este deja normalizata [49].

3.1.3 Romanian Speech Synthesis (RSS) Database

Lucrarea introduce o noua baze de date cu inregistrari de inalta calitate in limba romana, special gandite pentru sintetizare de voce, numita *RSS* [50]. Această bază de dată conține 3500 de propoziții de antrenare și 500 pentru testare, citite de către un vorbitor de gen feminin și a fost înregistrată utilizând multiple microfoane. Toate înregistrările au avut o frecvență de eșantionare supra-eșantionată la 96kHz cu codare de 24 de biți per eșantion, ca după să fie subeșantionate la o frecvență de 48kHz. Această metodă de supra-eșantionare este o metodă utilizată pentru a reduce zgomotul ce poate apărea în timpul înregistrării. Deoarece s-a supraeșantionat cu un factor de 4 relativ la frecvența Nyquiest de 24kHz, iar după s-a subeșanionat la 48Khz, raportul semnal-zgomot a crescut cu un factor de 4. Pentru înregistrare și subeșantionare s-a folosit software-ul ProTools HD.

Clipurile audio conțin articole de ziar, enunțuri din română, 2 povești scurte scrise de Ion Creangă și propoziții semantic imprevizibile, utilizate pentru testele de intengibilitate. Poveștile au fost divizate în propoziții și citite în ordinea originală a operei. Corpusul conține seturi atât de antrenare cât și de testare. Timpul total de înregistrare al setului de antrenare este de 3.5 ore, și conține aproximativ 3500 de propoziții : 1500 selectate aleator din ziar, 1000 selectate bazandu-ne pe diftongi și 1000 de propoziții din basme. Pentru setul de test timpul de înregistrări este în jur de 0.5 ore și conține 200 de propoziții alese aleator din ziare , 100 propoziții alese aleator din romane și **200 propoziții cu semantică imprevizibilă**.

Diftongii (*Diphones*) sunt în general utilizați ca cea mai mică unitate pentru metoda concatenării în sintetizarea vorbirii, iar grupurile de mai multe de 3 sunete (*Quinphones*) se folosesc ca cea mai mică unitate pentru sintetizarea cu modele Markov ascunse . Ne interesează să avem o acoperire fonetică cât mai mare pentru a avea o baza de date cât mai variată.

Toate propozițiile din ziar au fost citite utilizând o voce monotonă fără individualități (particularități) vocale, în timp ce basmele au avut mai mult ritm narativ și elemente de prozodie.

Datorită variației de caractere codate (pentru limba română avem caractere ce nu fac parte din codarea ASCII), corpusul de text a fost mai întâi normalizat înainte de procesările ulterioare.

3.1.4 Baza de date personală

Baza de date personală a fost construită utilizând clipuri audio de durata între 2 și 20 de secunde cu o frecvență de eșantionare de 22400Hz. Echipamentele utilizate au fost *Mixer Behringer x32 full size, Microfon akg d5* și ca software de înregistrare am utilizat unul *open source, Audacity.* Înregistrările au fost făcute într-o camera ce nu era izolată fonic, însă microfonul utilizat este unul ce elimină zgomotul. Textele propuse pentru baza de date au fost : romanul *Micul Prinț*, scris de Antoine de Saint Exupery, și diverse articole de știri de pe site-ul *stiripentrucopii.com.* Motivul alegerii acestor texte este faptul că ele sunt destinate copiilor. În consecință, am pornit de la presupunerea că textele pentru copii conțin cuvintele fundamentale ale limbii române și cele mai ușoare forme de exprimare. Așadar, am dorit să antrenăm rețeaua să învețe elementele fundamentale ale limbii, pentru ușurință. Corpusurile de text au fost divizate manual în propoziții și citite în ordinea originală. Totuși, clipurile audio înregistrate au mai trecute după printr-o etapă de segmentare deoarece am observat că au existat destul de multe clipuri ce depășeau 20 de secunde. Timpul total de înregistrări este de aproximativ 3 ore. Citirea textelor nu este una cu o voce perfect inexpresivă, ci există elemente de expresivitate și prozodie în citire.

După înregistrare și segmentarea clipurilor a urmat etapa de normalizare a textului. Problemele întâmpinate la înregistrare au fost următoarele :

- prima problemă a fost cea a segmentării clipurilor. Au fost foarte multe clipuri de lungimi mult prea mari, deoarce nu am selectat un număr maxim de cuvinte pentru clipuri
- următoarea problemă a fost nivelul de dB pentru fiecare clip. Clipurile au început să se audă din ce în ce mai încet din cauza poziției mele în timp incorecte în fața microfonului

Pentru a rezolva aceste probleme, s-a studiat ce clipuri din cele 500 au lungimi mai mari de 20 de secunde pentru a fi segmentate, iar după s-a studiat ce clipuri sunt sub un anumit nivel dB, cu referință la prima înregistrare.

Figura 3.1: Înregistrarea bazei de date proprii

3.1.5 Preprocesarea textului (Normalizare)

3.1.5.1 Normalizarea bazelor publice

Etapa de normalizare a fost necesară doar pentru baza de date în engleză. Bazele de date RSC și RSS au fost deja prenormalizate de către autori. Pentru normalizarea în limba engleză s-au utilizat expresii regulate (Regex) pentru eliminarea anumitor caractere, sau substituirea unora. Astfel au fost eliminate toate formele de expresivitate din punctație (s-a renunțat la "!" sau "..." în favorul "."), liniile de dialog, abrevierile, caracterele mari au devenit mici.

3.1.5.2 Normalizarea bazei de date proprii

Deoarece nu a existat nicio formă de normalizare precedentă, cea mai multă preprocesare a fost făcută pe bază proprie de date utilizând *expresii regulate*.

Astfel, pe lângă eliminarea/substituirea caracterelor (se adaugă și cratimă), a trebuit să fie făcută și conversia numerelor în scris (24 → douăzeci și patru). De asemenea, denumirile străine au fost transcrise după modul în care sună în limba română.

RAW	NORMALIZAT
„	NONE
”	NONE
“	NONE
. . .	.
;	.
Eugène	Eugen
businessman	bisnismen
(	NONE
)	NONE
-	NONE
:	.
!	.
+	plus
- (dacă se află langă numere)	minus
=	egal
/	împărțit la
*	înmulțit cu
. (pentru site-uri web)	punct
A-Z	a-z
număr (ex : 1998)	transcriere în litere (ex: o mie nouă sute nouă zeci si opt)

Tabela 3.1: Tabela de normalizare a textului

Studentul are 221341 de foi !!! A ales sa le lase acasa spunandu-si ”ce bine ca nu le-am luat”...

studentul are două sute douăzeci și unu de mii trei sute patruzeci și unu de foi . a ales sa le lase acasa spunandusi ce bine ca nu leam luat .

3.1.6 Prepocesarea audio si asocierea [clip audio-text]

După realizarea script-ului de normalizare a textelor în limba română, a urmat o reevaluare a clipurilor audio. Am realizat un script ce detectează durata fiecărui clip și am observat faptul că aproximativ 70 de clipuri aveau o durata mai lungă de 20 de secunde. Așadar, am creat un nou program ce a sortat clipurile audio mai lungi de 20 de secunde în foldere după durata lor. După a urmat o segmentare manuală a fiecărui clip în bucăți de câte 5-10 secunde.

Următorul pas a fost verificarea aceluiași nivel de dB pentru fiecare clip audio și am constatat faptul că ultimele clipuri aveau un nivel mult mai scăzut față de clipurile debutante (datorită îndepărtării mele de microfon în timpul înregistrării). Astfel, am luat ca referință primul clip (ce avea 80dB cu o referință de 1) și am amplificat clipurile ce se aflau sub pragul de 75dB astfel încât să se apropie de volumul clipului luat că referință. Pentru acest lucru am utilizat tool-ul din linia de comandă *sox*.

După acești pași, a urmat crearea unui fișier metadata.txt ce conține perechile [clip audio-text]. Textul a fost segmentat manual în timpul înregistrării și asociat cu fișierele audio ulterior, utilizând un script propriu.

3.1.7 Preprocesarea Spectrogramelor liniare și Mel

Preprocesarea spectrogramelor liniare și Mel a fost realizată doar pentru setul de date în engleză *LJSpeech* pentru a fi utilizate în implementarea proprie a rețelei. Pentru bazele de date în română, această preprocesare nu a mai fost făcută deoarece au fost testate pe rețeaua propusă de *Tao* [1], rețea ce are incorporat deja un script de calculare a spectrogramelor.

Calcularea spectrogramelor liniare și a spectrogramelor Mel corespunzătoare clipurilor audio a fost făcută întocmai metodelor descrise în capitolul 2 utilizând librăria *librosa*.

Astfel pentru calculul spectrogramei liniare am ales pentru parametrii următoarele valori:

- sr = 22050 → (frecvența de eșantionare)
- n_fft = 2048 → (puncte fft (eșantioane de frecvență)
- frame_shift = 0.0125 → (secunde)
- frame_length = 0.05 → (secunde)
- hop_length = int(sr*frame_shift) → (eșantioane)
- win_length = int(sr*frame_length) → (eșantioane)
- n_mels = 80 → (numărul de Bancuri de filtre Mel)
- power = 1.2 → (exponent de amplificare)
- n_iter = 50 → (numărul de iterații pentru algoritmul Griffin Lim)
- preemphasis = .97
- r = 5 → (factor de reducere)

Pentru textul :

"In being comparatively modern."

s-a obținut următoarea spectrogramă :

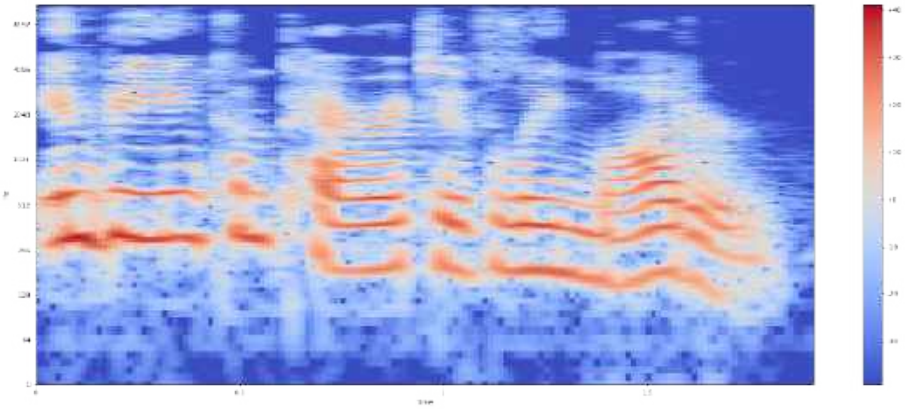

Figura 3.2: Spectrograma liniară a secvenței

Pentru calculul Spectrogramei Mel mai întâi s-au calculat cele 80 de filtre Mel:

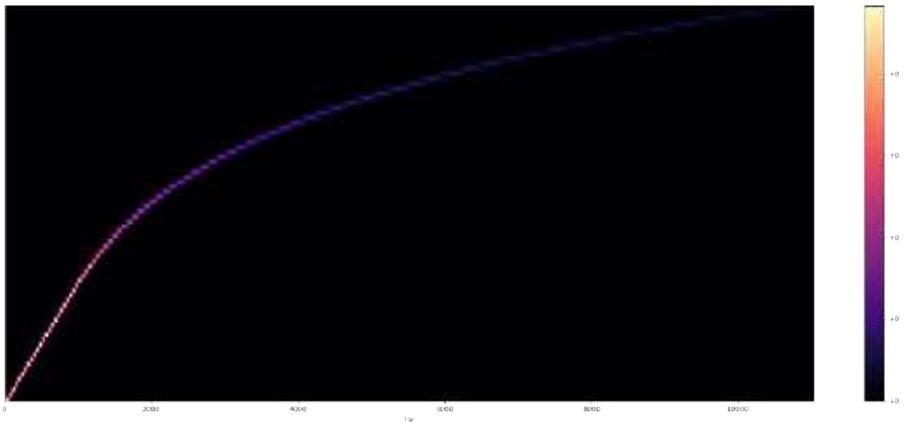

Figura 3.3: Reprezentarea matricei de transformare cu 80 de filtre Mel

Următorul pas a fost sa filtrez spectrogramele liniare utilizând filtrele mel, obținând astfel spectrograma Mel. Aceasta a fost ulterior normalizată pentru a lucra cu valori mai mici:

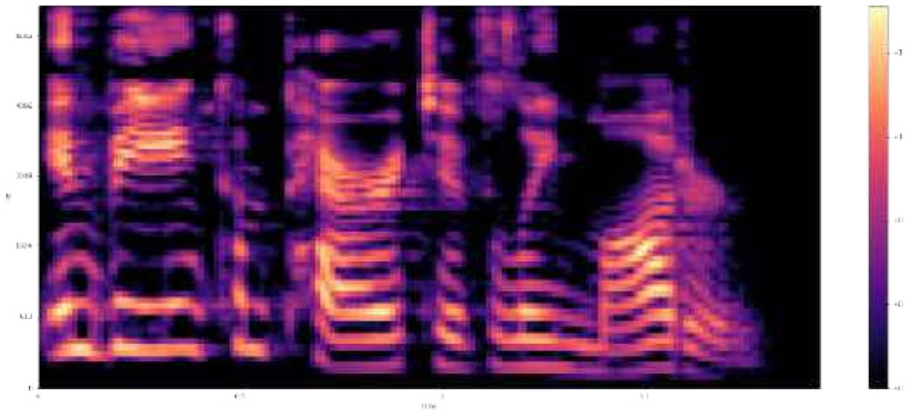

Figura 3.4: Conversia spectrogramei liniare în spectrogramă Mel

De asemenea asupra spectrogramei s-a aplicat și un factor de reducere de $r = 5$ pentru a ușura calculul / procesarea rețelei neurale :

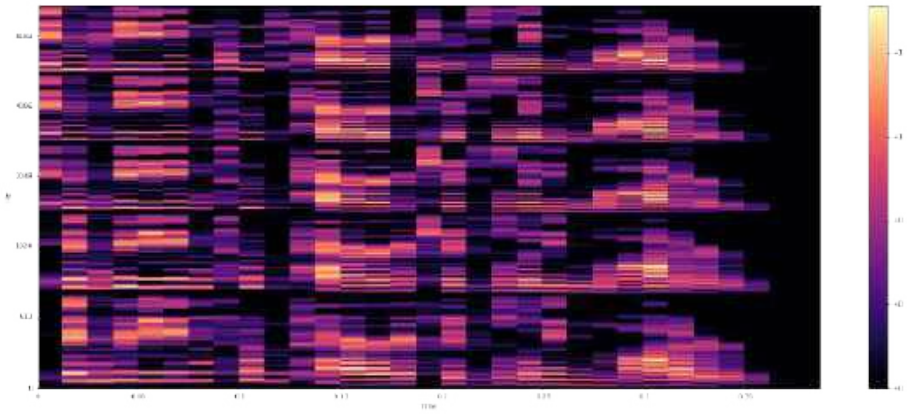

Figura 3.5: Spectrograma Mel redusă cu factor de 5

3.1.8 Algoritmul Griffin Lim

Algoritmul este utilizat pentru a converti spectrogramele liniare înapoi în clipuri audio. Acesta folosește metoda de reconstrucție în fază bazându-se pe redundanța STFT-ului. Metoda de reconstrucție este una iterativă [35].

3.2 Rețeaua Tacotron

3.2.1 Implemenatarea propusă de *Tao*[1]

În prima fază, am descărcat o arhitectura propusă și am observat cum reacționează acest tip de rețea la diferite baze de date. Rezultatele au fost bune doar pentru doua dintre bazele de date, obținându-se o voce destul de clară și coerentă.

Fiecare bază de date a avut un număr diferit de caractere utilizate și o frecvență de eșantionare proprie pentru clipuri. Acești doi parametrii au trebuit sa fie mereu ajustați. Bazele de date au fost apoi normalizate în modul expus anterior.

Ulterior, s-au rulat scripturile specifice pentru antrenarea rețelei. Antrenarea a durat nu mai mult de 12 ore, iar analiza s-a realizat cu *tensorboard* pentru monitorizarea funcției de eroare (*loss*). Rezultatele și concluziile vor fi elaborate în capitolul următor.

3.2.2 Implementare proprie

După familiarizarea cu rețeaua *Tacotron* am încercat să creez o implementare proprie ale acestei rețele.

Pregătirea textului pentru antrenare. Keras generator

Partea de preprocesare și pregătire a datelor reprezintă un pas cheie în obținerea unor rezultate bune. Fără o prelucrare corectă a datelor de intrare, rețeaua neurală nu va reuși să învețe corect. Pentru implementarea proprie am decis să utilizez baza de date *LJSpeech* pentru limba engleză, deoarece este o bază de date foarte des folosită în acest domeniu și este cunoscută pentru rezultatele bune. Scopul a fost pregătirea unei baze de date deja corect înregistrată și segmentată și implementarea mai ales a arhitecturii rețelei [51].

Primul pas a fost calcularea spectrogramelor liniare și Mel a fișierelor audio după cum am expus în subcapitolul precedent.

Al doilea pas a fost normalizarea textului. Normalizarea a fost una minimalistă deoarece deja au fost făcute anumite expandări de către autorii *LJSpeech* (expandarea numerelor, a simbolurilor, a abrevierilor)

După normalizarea textului a urmată etapă de *tokenizare*. Această etapă este esențială în partea de procesare. Nu putem lucra cu textul propriu zis pentru procesare ci avem nevoie de o reprezentare ușor de procesat pentru calculator. Vom asocia astfel fiecărei litere un număr utilizând funcția *char2idx*.

După ce avem acum atât textul cât și spectrogramele (ieșirile) pregătite este momentul să le pregătim în loturi de antrenare (batch-uri). În cazul de față la intrare vom avea atât intrarea pentru Codor cat și pentru Decodor. Astfel, perechile vor fi de tipul (($x\_text, y\_spectrograma$), $y\_spectrograma$). O primă idee a fost să încărcăm în memorie toată baza de date deodată și să creăm un vector ce conține că elemente perechi de tupluri intrare-ieșire (($x\_text, y\_spectrograma$), $y\_spectrograma$). După, în momentul antrenării urma să împărțim acest vector în loturi și să le oferim pe rând ca și date de intrare în rețea. Problema a fost apariția erorii *OOM (out of memory)* datorată numărului prea mare de date încărcate în memorie deodată. O soluție ar fi

fost reducerea lotului (de la 16 perechi intrare-ieșire la 4). Dar și așa au existat momente în care programul se oprea din execuție. Prin urmare am apelat la soluția generatorului.

Principalul avantaj în utilizarea generatoarelor constituie faptul că loturile nu sunt încărcate deodată în memorie ci vor intra secvențial. Astfel memoria nu va mai fi încărcată cu tot setul de date deodată. Un alt avantaj al acestei metode constituie partea de *padding*. *Padding*-ul, după cum am discutat și în subcapitolul despre rețele convoluționale, este o tehnică folosită pentru ca lungimile vectorilor de trăsături să fie aceleași, de obicei prin adăugare de zerouri (sau alte elemente ce decidem să fie ignorate de rețea) la începutul sau la finalul vectorului. Problema la prima metodă era faptul că *padding*-ul era aplicat la nivelul întregii baze de date, pe când cu generator *padding*-ul se aplică doar la nivel de lot. În consecință, setul de date nu va mai fi influențat atât de tare de adăugarea de zerouri, iar vectorii de trăsături vor putea avea dimensiuni mult mai variate.

Pentru a crea un generator, vom scrie o clasă DataGenerator ce are următoarele metode:

- metoda de initializare in care specificam parametrii pentru pregatirea lotului de antrenare: dimensiunea lotului (*batch-size*), daca vrem sau nu sa amestecam datele (*shuffle*), lista iesirilor (labels), lista intrarilor ($listID$)

- a doua metoda va fi metoda de generare a datelor. Aceasta metoda va primi ca si parametru de intrare o lista cu intrarile ce vor face parte din batch-ul respectiv ($listIDstemp$)

  După, vom inițializa variabilele de intrare și ieșire (X și Y) cu vectori de dimensiune fixă. Dimensiunea fixă va varia în funcție de lot. Înainte de inițializare trebuie să vedem care este variabila de intrare de dimensiunea cea mai mare. După vom adăuga zerouri la finalul vectorilor pentru a avea vectorii de trăsături din același lot de aceeași dimensiune. După ce am inițializat cu vectori nuli variabilele X și Y, trebuie sa umplem acum acești vectori cu datele propriu zise, descărcându-le din fișierele propriu zise. Observam faptul că aici este momentul încărcării datelor în memorie. Se încarcă doar fix cât să umplem *batch*-ul curent . Astfel fiecare element din X și Y va lua valoarea elementului corespunzător din lista elementelor de intrare. Acum avem setat lotul curent de antrenare.

- A doua metoda va fi *__len__*ce denota numărul de loturi dintr-o epocă. O epocă înseamnă o parcugere a întregului set de date. Prin urmare numărul de loturi va fi egal cu dimensiunea întregului set de intrări (len($listIDS$)) împărțit la dimensiunea lotului.

- Următoare metodă *__getitem__*este metoda utilizată pentru a putea selecta *batch*-urile din setul de antrenare. Prima variabilă este *index* ce contine toți indicii corespunzători pentru selecția intrărilor pentru următorul *batch*. Această variabilă ne ajută să selectăm ce date vom prelua din fișiere pentru pregătirea batch-ului. Practic construim lista $list\_IDS\_temp$ ce va fi data ca parametru de intrare în apelarea funcției *__data_generation__*. După, apelam funcția de generare și ieșirile le returnăm în sub forma ((X, Y), Y).

- Ultima metoda *on_epoch_end* după cum sugerează și numele este o metodă apelată pentru a amesteca sau nu indicii listei intrărilor ($list\_IDs$).

În acest moment datele sunt normalizate dar și pregătite sa fie oferite ca intrare în mod secvențial pe loturi.

Construirea arhitecturii

Rețeaua Tacotron e menită sa fie End to End. Însă pentru a fi ușoara prelucrarea, am împărțit rețeaua în doua *task-uri*.

1. Transcrierea textului în Spectrograma Mel
2. Conversia spectrogramei Mel în spectrograma liniară și conversia ei în clip audio utilizând algoritmul Griffin Lim

Cea mai dificilă parte a constituit etapa de conversie a textului în spectrogramă Mel. În prima fază vom elabora etapa mai ușoară, anume conversia spectrogramei Mel în spectrogramă liniară.

Conversia spectrogramei Mel în spectrogramă liniară

Pentru această parte am utilizat 2 metode :

- Metoda cu tehnici de învățare adâncă
- Metoda deterministă

Pentru metoda cu tehnici de învățare adâncă sarcina a presupus ca o spectrogramă Mel dată la intrare să fie convertită în spectrogramă liniara la ieșire. Acest lucru nu ar trebui sa fie imposibil deoarece există deja algoritmi bine definiți, deci nici rețelei noastre nu ar trebui sa-i fie greu sa găsească relațiile matematice necesare.

Datele au fost pregătite utilizând generatori, singura diferență fiind că intrarea o constituie doar perechile ($x\_spectrograma\_liniara, y\_spectrograma\_mel$).

Dimensiunea lotului de antrenare a fost 8 . Arhitectura rețelei a constat din blocul CBHG. Acesta a fost alcătuit din:

- succesiunea a 16 straturi de convoluții cu un fitru de lungime 127 ($embed\_size//2$, unde dimensiunea $embedding - ului$ este de 255) si o dimensiune fixa de kernel = 1.
- un strat de *maxpooling*
- doua straturi de convoluții 1D, cu filtre de lungime 127, și kernel_size = 3
- un strat 127 de neuroni pentru aducerea datelor la aceeași dimensiune cu intrarea
- ieșirea stratului ascuns de neuroni este adunata cu o conexiune reziduala la intrarea în modul
- un strat GRU Bidirecțional

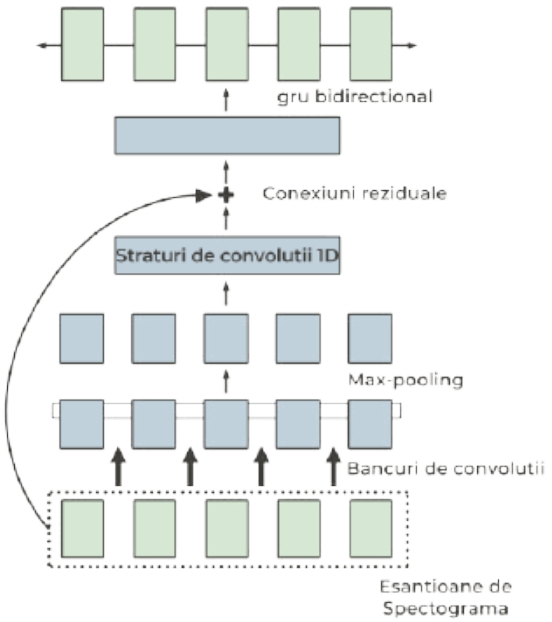

Figura 3.6: Arhitectura rețelei de conversie a spectrogramelor

Problema de față este una de secvență la secvență. Avem nevoie să convertim secvențele unei spectrograme Mel în secvente de spectrograma liniară. Am discutat deja că într-o astfel de problemă fiecare secvență depinde de celelalte și este în general preferată o arhitectură de tip Codor – Decodor .

În cazul nostru, pentru codarea secvenței de intrare s-au utilizat cele 16 bancuri de convoluții urmate de un strat de maxpooling pentru extragerea trăsăturilor cele mai importante și o proiecție de convoluții. Straturile de convoluții sunt cele mai des întâlnite în extragerea trăsăturilor de imagini (în cazul nostru spectrograme) dar și pentru a găsi legături în secvențe (cum am discutat în subcapitolul anterior). După codarea acestei secvențe, decodarea se produce utilizând un strat GRU bidirecțional, avantajul fiind faptul că este un tip de rețea recurentă cu memorie și putem păstra mai bine informațiile despre secvențele din spate. De asemenea este și o rețea bidirecțională deci prelucrăm informația atât de la stânga la dreapta cât și de la dreapta la stânga, știind astfel cum depinde elementul curent atât de element precedent cât și cel succesiv.

Rezultatele au fost unele bune. Amintim faptul că s-a folosit ca și antrenare baza de date LJSpeech pentru această sarcina. Am presupus că în acest caz nu contează limba pentru a converti o spectrograma mel într-una liniară, deoarece dependențele sunt aici doar din punct de vedere matematic nu și lingvistic. Astfel în faza de test am avut rezultate acceptabile și pentru spectrograme în limba română.

Pentru metoda determinista am utilizat funcția *inverse_mel_to_spec* a modulului *librosa*. Funcția utilizează algoritmul *Non-negative Lease Squares* . Parametrii de intrare pentru aceasta funcție sunt: spectrograma mel, frecventa de eșantionare, și numărul de puncte în care se calculează $STFT$-ul.

Un lucru de luat în vedere este faptul ca obținerea spectrogramei Mel din spectrograma liniara este un proces ce implica pierderi de informație deoarece spectrograma Mel nu este decât o filtrare a spectrogramei originale.

În ambele cazuri pentru conversia din spectrograma liniara în audio, s-a utilizat algoritmul *Griffin Lim*.

Rezultatele folosind modulul *librosa* au fost mai bune, decât folosind rețele neurale.

Conversia textului în Spectrogramă Mel
Pentru aceasta sarcina avem următoarea arhictectura :

1. *Codorul* :
   - Stratul de Character Embedding (Character2Vector)
   - Un strat de prepocesare (Prenet)
   - Modulul CBHG (bancurile de Convlutii)
   - Highway Nets
   - GRU
2. *Decodorul*
   - Preprocesare (Prenet)
   - Doua straturi GRU
   - Un strat de dense de ieșire ce calculează probabilitățile fiecărei secvențe de spectrogramă
3. *Atenția*

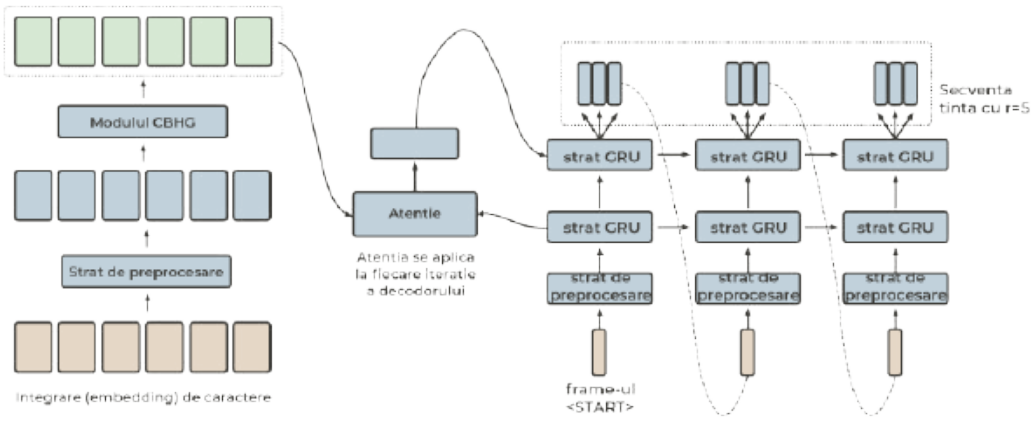

Figura 3.7: Arhitectura rețelei de conversie a textului în spectrogramă Mel

Codorul

Primul strat în codor este stratul de conversie al caracterelor în vectori (Character Embedding). Acest strat este esențial pentru a găsi legăturile la nivel de caracter ale cuvintelor, mai precis legăturile sunetelor ale limbii respective. *Embedingg*-ul in acest caz se alege de o dimensiune mai mare decât vocabularul (toate caracterele alfabetului + semne de punctuație) adică 256. S-a arătat experimental că alegerea unui embedding mai mare decât vocabularul este în regulă. Spre exemplu fie un lot de antrenare de lungime 2 ce conține următoarele secvențe :

["Ana are mere."; "Tata are mașini".]

Vom asocia în etapa de preprocesare cum am discutat, fiecărui caracter un număr, iar după vom egaliza dimensiunea vectorilor prin adaugărea de zerouri la dreapta (*padding*). Astfel lotul de antrenare va arăta de forma :

[1 2 1 4 1 6 3 4 5 3 6 3 0 0 0; 7 1 7 1 4 1 6 3 4 5 1 9 8 10 8]

Urmează acum stratul de *embedding* în care vom asocia fiecărui caracter (convertit acum în număr) un vector aleator de lungime 256 (embed size)

După etapa de *Embedding* urmeazaă etapa de preprocesare ce este doar o trecere printr-un strat de neuroni (*Dense Layers*).

După această prelucrare urmează extragerea de trăsături utilizând bancurile de rețele convoluționale și proiecțiile de la ieșire cu modulul *CBHG*.

În cazul nostru, pentru bancurile de convoluții, mărimea kernelului este dată de numărul stratului (pentru stratul n de convoluție rezultă kernelsize = n). La ieșirea din bancurile de filtre se aplică operația de *maxpooling (eng)* pentru a extrage trăsăturile predominate ale secvenței de la intrare. La final, ieșirile se trec din nou prin 2 straturi de convoluții. La rezultatul obținut se adaugă conexiuni reziduale (*highway-nets*), ce sunt legături ale stratului de intrare cu stratul curent. Acesta este utilizat pentru a păstra informația de la intrare, deoarece după o serie de bancuri de convoluții se poate întâmpla să se piardă informația inițială de la intrare.

Ultimul strat al acestui modul, îl constituie stratul GRU bidirecțional ce prelucrează secvențele de text procesate prin straturile anterioare, astfel încât să fie aliniate corect cu decodorul prin intermediul atenției. Parametrii principali ai acestui modul îi constituite *nr_of_hidden_units = 128* și *return_seq = true*. Numărul de neuroni (hidden units) reprezintă numărul de neuroni prin care va trebui să treacă pe rând fiecare vector asociat unui caracter. De asemenea ne vom dori să știm ieșirea fiecărei indexări (state)/ieșirea de la fiecare time-step (*return_seq = True*) pentru a ști pe ce caractere să ne focusăm atenția atunci când vom decoda. Pe lângă acești doi parametri mai avem parametrul *return_state = true* ce returneaza ultima stare a codorului.

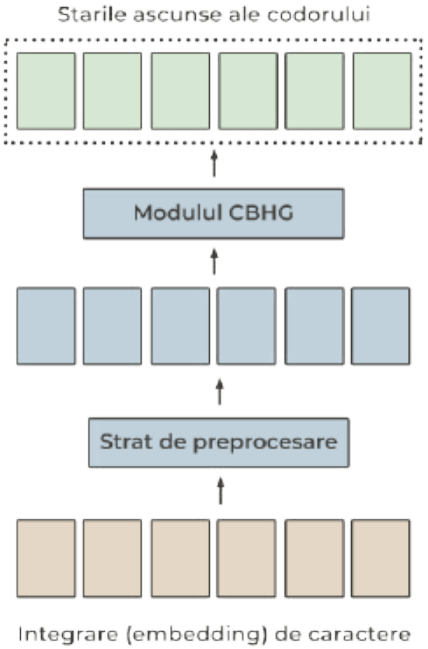

Figura 3.8: Arhitectura Codorului

Decodorul

Acesta este utilizat pentru decodarea secvențelor din Codor. Are stratul de prepocesare urmat de 2 straturi de rețele recurente de tip GRU. Deoarece problema decodorului este de a învăța să genereze secvența, și nu de a prelucra, se vor folosi rețelele recurente. De asemenea, rețelele recurente din decodor vor fi legate la GRU-ul din codor, tocmai ca prin ajutorul atenției, să reușească să alinieze caracterele cu eșantioanele corecte din spectrogramă generată.

Aici intrarea au consituit-o eșantioanele de spectrogramă Mel, preluate din generatorul discutat la partea de preprocesare a datelor. Și aici parametrii principali ai straturilor GRU îi constituie *numărul de unități*, *return seq si return state*. Și în acest caz vom dori să returnăm fiecare timestep deoarece va fi oferit mai departe ca intrare în următorul GRU și pentru că vrem să obținem la ieșire tot o secvență. E bine să ne reamintim că ieșirea de la fiecare timestep constituie de fapt prezicerea următorului timestep.

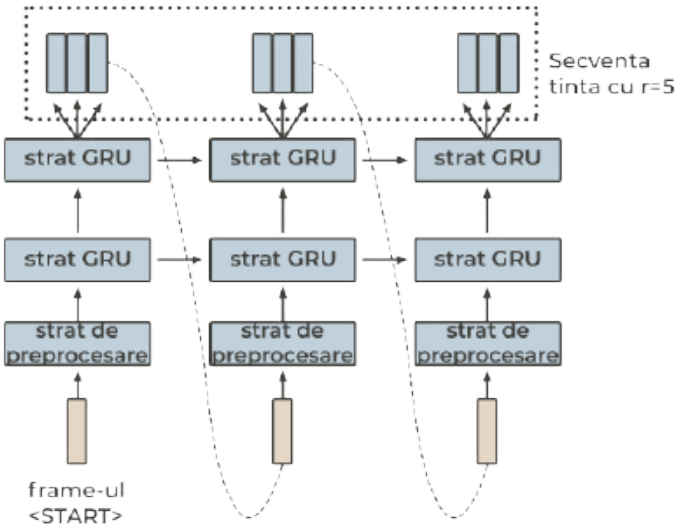

Figura 3.9: Arhitectura Decodorului

Atenția

Atenția folosită în această aplicație a fost atenția Bahdanau. Aceasta functionnează în felul următor :

Față de abordarea clasică unde ieșirea codorului era constituită doar de ultima iterație a rețelei recurente, aici vom lua fiecare stare ascunsă (iterație) în considerare și le vom oferi ca intrare în modulul de atenție. Toate stările curente ale Codorului și starea anterioară a Decodorului sunt utilizate pentru a calcula vectorul de context ce decide care caracter are o importantă mai mare la acel moment.

Cunoaștem faptul că Codorul scoate starea ascunsă h_i pentru fiecare iterație a secvenței sursă. Starea ascunsă h_i este un vector de dimensiune 256 (numărul de neuroni la ieșire ales este 128, însă este o rețea bidirecțională și atunci ieșirea i se dublează). Deci pentru fiecare iterație i vom avea la ieșire un vector h_i de dimensiune 256.

Cunoaștem de asemenea faptul că decodorul pentru fiecare iterație t, generează o stare ascunsă s_t. Ne interesează să calculăm alinierea (*alingment score (eng)*) dintre s_{t-1} (starea anterioară a decodorului) și fiecare iterație h_i cu i de la 1 la N, unde N este numărul de cuvinte a secvenței sursă, pentru a prezice corect starea s_t [52].

$$a_{t,i} = align(s_{t-1}, h_i) \tag{3.1}$$

Pentru a calcula alinierea este nevoie de funcția *score*. Deoarece folosim atentie *Bahdanau* vom utiliza următoarea funcție de scor:

$$\text{score}\left(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i\right) = \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W}_1 \boldsymbol{s}_{t-1} + \boldsymbol{W}_2 \boldsymbol{h}_i\right) \tag{3.2}$$

Se aduna ieșirea curenta h_i a codorului cu starea precedentă a decodorului s_{t-1}, iar rezultatul se trece printr-o funcție tangențială. După, rezultatul este trecut printr-un singur neuron pentru

a obține un singur număr reprezentativ. Procesul se repetă pentru fiecare i de la 1 la N, astfel obținându-se un vector de aliniere (*alignment Score vector (eng)*) de dimensiune N, unde N este dimensiunea secvenței sursă. Vectorul de aliniere este apoi trecut prin funcția *softmax* de unde rezultă și formula :

$$\begin{aligned} \alpha_{t,i} &= \text{align}\left(y_t, x_i\right) \\ &= \frac{\exp\left(\text{score}\left(s_{t-1}, \boldsymbol{h}_i\right)\right)}{\sum_{i'=1}^{n} \exp\left(\text{score}\left(\boldsymbol{s}_{t-1}, \boldsymbol{h}_{i'}\right)\right)} \end{aligned} \tag{3.3}$$

Am generat astfel vectorul de ponderi (*attention weights vector (eng)*) cu elemente $a_{t,i}$, unde i ia valori de la 1 la N, ce oferă probabilitățile pentru fiecare caracter din secvența sursă. După calcularea ponderilor de atenție urmează calcularea vectorului de context. Acesta se calculează prin înmulțirea pe rând a fiecărei stări h_i (un vector de 256 de elemente) cu elementul $a_{t,i}$ ce este un scalar, reprezentând "importanța" stării h_i. La final rezultatele se sumează obținându-se vectorul de context. Vectorul de context este apoi concatenat cu starea precedentă s_{t-1} și folosit pentru e prezice următoarea stare s_t [53].

$$\mathbf{c}_t = \sum_{i=1}^{n} \alpha_{t,i} \boldsymbol{h}_i \tag{3.4}$$

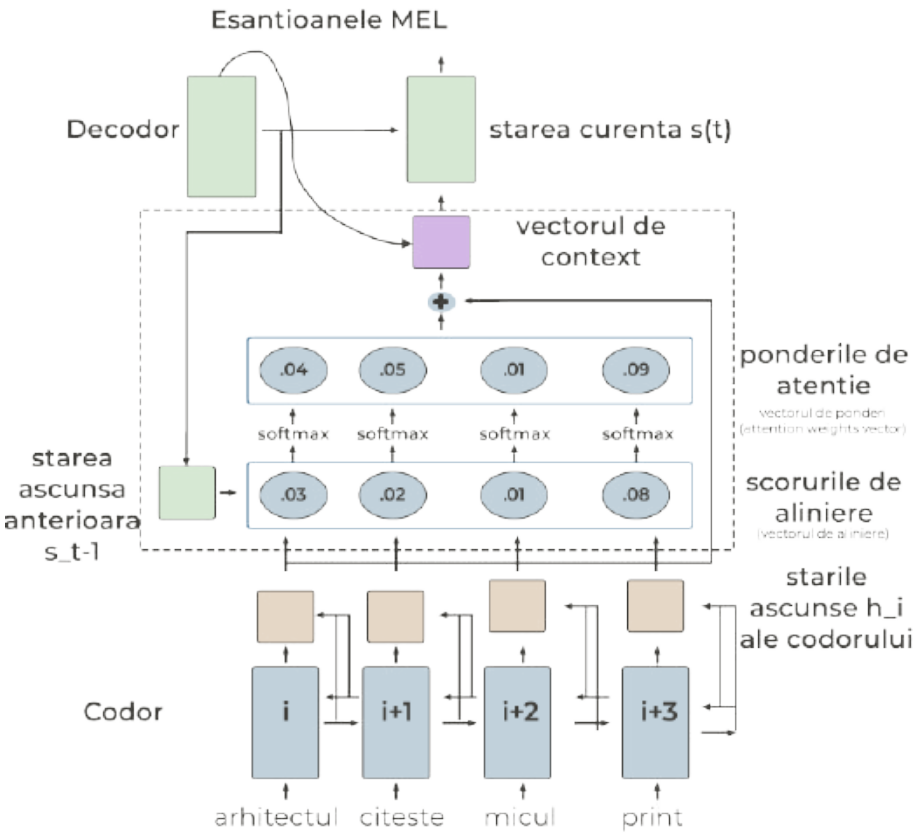

Figura 3.10: Arhitectura atenției Bahdanau

Parametrii de antrenare

După cum am menționat în capitolul precedent, în utilizarea rețelelor neurale recurente poate apărea problema dispariției gradientului. O soluționare a acestei probleme este utilizarea metodei *graddient clipping*. Această metodă presupune forțarea valorilor gradientului către un minim sau maxim specific în cazul în care gradientul nu se află în intervalul dorit [54].

Un alt parametru important utilizat pentru antrenare este optimizatorul ce reprezintă o metodă de a varia schimbările ponderilor și a ratei de învățare astfel încât gradientul să nu rămână blocat într-un minim local. S-a ales utilizarea optimizatorului *Adam*.

De asemenea pentru funcția de cost s-a ales MAE (*Mean Absolute Error*)

$$MAE = (\frac{1}{n}) \sum_{i=1}^{n} |y_i - x_i| \tag{3.5}$$

Analiză spectrală	pre-emphasis:0.97; frame_length: 50ms; frame_shift: 12.5ms; window_type: Hann
Character embedding	256
Codor (CBHG)	bancuri Conv1D: K=16, conv-k-128-ReLU MaxPooling: stride=1, width=2 Conv1D Proiecții:conv-3-128-ReLU → conv-3-128-ReLU Highway net:4, 128-ReLU GRU Bidirecțional:128
Codor preprocesare	Dense: 256-ReLU Dense: 128-ReLU
Decodor preprocesare	Dense:256-ReLU Dense:128-ReLU
Decodor RNN	2 GRU : 256 units
Atenție RNN	1 GRU Bidirecțional : 128
Post Procesare CBHG	bancuri Conv1D: K=16, conv-1-128-ReLU MaxPooling: stride=1, width=2 Conv1D Proiecții:conv-3-128-ReLU → conv-3-128-ReLU Highway net:4, 128-ReLU GRU Bidirecțional:128
Factor de reducere (r)	5

Tabela 3.2: Parametrii Rețea

Capitolul 4
Rezultate experimentale

Rezultatele experimentale sunt urmatoarele :

4.1 Rezultate rețea propusă de *Tao*[1]

Pentru antrenarea rețelei propusă, s-au folosit următorii parametrii în prelucrarea spectrogramelor:

- n_mels: 80
- n_fft: 1025
- sr: frecventa de esantionare a clipurilor audio este diferita pentru fiecare baza de date
- frame_length: 0.04 secunde
- frame_shift: 0.0125 secunde
- preamphasis = 0.97
- r = 5
- embedding_size = 256

4.1.1 Baza de date LJSpeech

Cost					
Parametru	Antrenare	Validare	Pas	Domeniu(e)	Domeniu(Hz)
Spectrograma Liniară	0.0394	0.0436	98k	[0,300]	[0, 585]
Spectrograma Mel	0.0361	0.0419	98k	-	-
Total	0.0756	0.0856	98k	-	-

Tabela 4.1: Rezultate LJSpeech

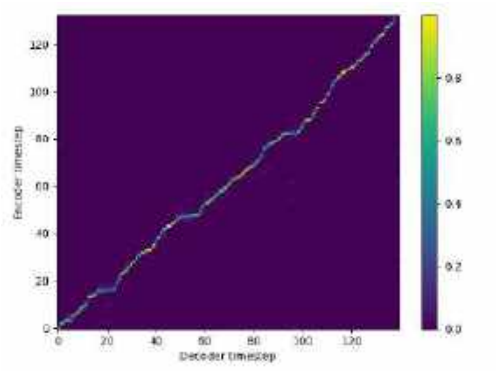

Figura 4.1: LJSpeech atenție

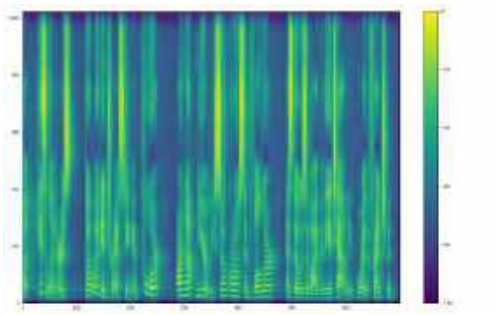

Figura 4.2: LJSpeech spectrogramă

4.1.2 Baza de date RSC

4.1.2.1 Antrenare folosind doar cuvinte

Cost					
Parametru	Antrenare	Validare	Pas	Domeniu(e)	Domeniu(Hz)
Spectrograma Liniară	0.0352	0.0465	146k	[0 600]	[0 1170]
Spectrograma Mel	0.0248	0.0427	146k	-	-
Total	0.0601	0.0892	146k	-	-
Spectrograma Liniară(min)	0.0373	0.0397	32k	[0 600]	[0 1170]
Spectrograma Mel(min)	0.0304	0.0358	32k	-	-
Total(min)	0.0678	0.0755	32k	-	-

Tabela 4.2: Rezultate RSC cuvinte

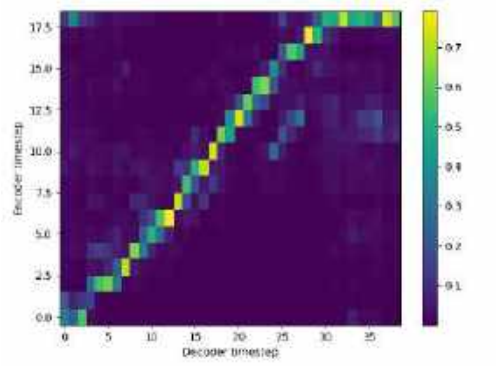

Figura 4.3: RSC cuvinte atenție(min)

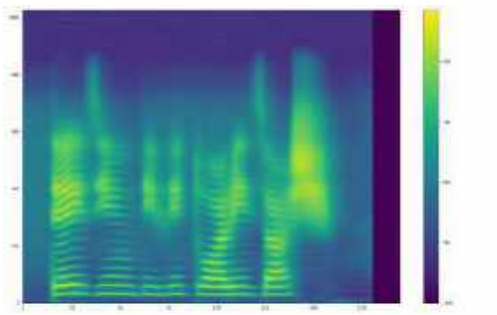

Figura 4.4: RSC cuvinte spectrogramă(min)

4.1.2.2 Antrenare folosind doar propoziții

Cost					
Parametru	Antrenare	Validare	Pas	Domeniu(e)	Domeniu(Hz)
Spectrograma Liniară	0.0418	0.0653	90k	[0 600]	[0 1170]
Spectrograma Mel	0.0368	0.0644	90k	-	-
Total	0.0787	0.1299	90k	-	-
Spectrograma Liniară(min)	0.0292	0.0495	40k	[0 600]	[0 1170]
Spectrograma Mel(min)	0.0306	0.0573	40k	-	-
Total(min)	0.0598	0.106	40k	-	-

Tabela 4.3: Rezultate RSC propoziții

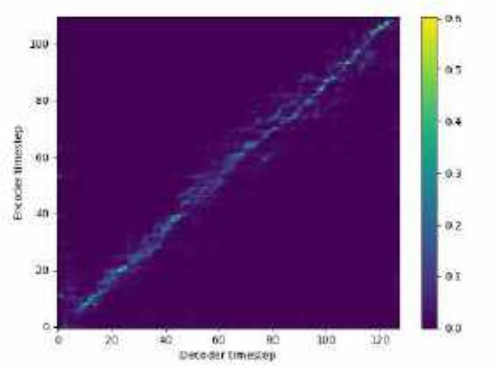

Figura 4.5: RSC propoziții atenție(min)

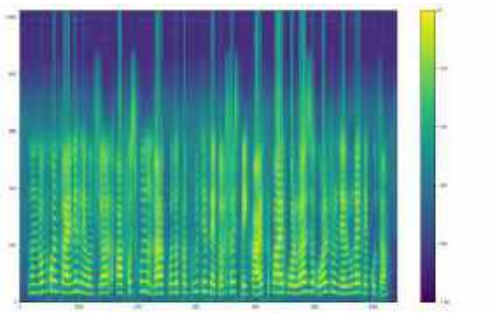

Figura 4.6: RSC propoziții spectrogramă(min)

4.1.2.3 Antrenare folosind cuvinte urmate de propoziții

Cost					
Parametru	Antrenare	Validare	Pas	Domeniu(e)	Domeniu(Hz)
Spectrograma Liniară	0.0450	0.0646	98k	[0 700]	[0 1365]
Spectrograma Mel	0.0375	0.0627	98k	-	-
Total	0.0825	0.1273	98k	-	-
Spectrograma Liniară(min)	0.0404	0.049	36k	[0 700]	[0 1365]
Spectrograma Mel(min)	0.0392	0.0500	36k	-	-
Total(min)	0.0796	0.099	36k	-	-

Tabela 4.4: Rezultate RSC cuvinte-propoziții

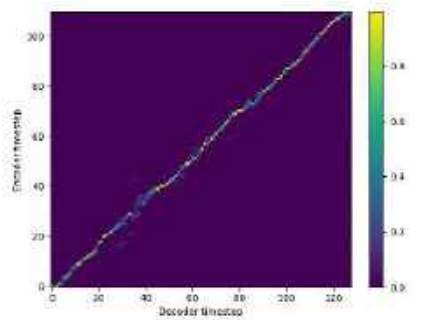

Figura 4.7: RSC cuvinte-prop. atenție(min)

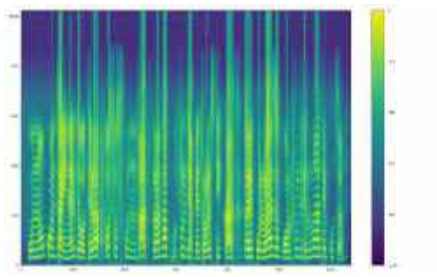

Figura 4.8: RSC cuvinte-prop. spec.(min)

4.1.3 Baza de date RSS

Cost					
Parametru	Antrenare	Validare	Pas	Domeniu(e)	Domeniu(Hz)
Spectrograma Liniară	0.0404	0.04108	98k	[0 200]	[0 390]
Spectrograma Mel	0.03405	0.0407	98k	-	-
Total	0.0745	0.0818	98k	-	-

Tabela 4.5: Rezultate RSS

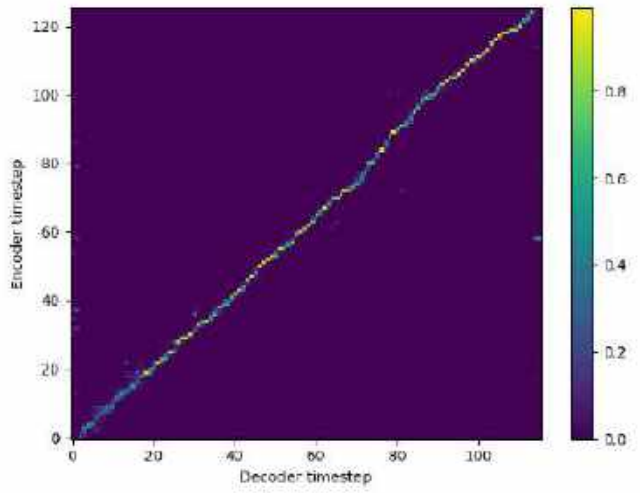

Figura 4.9: RSS atenție

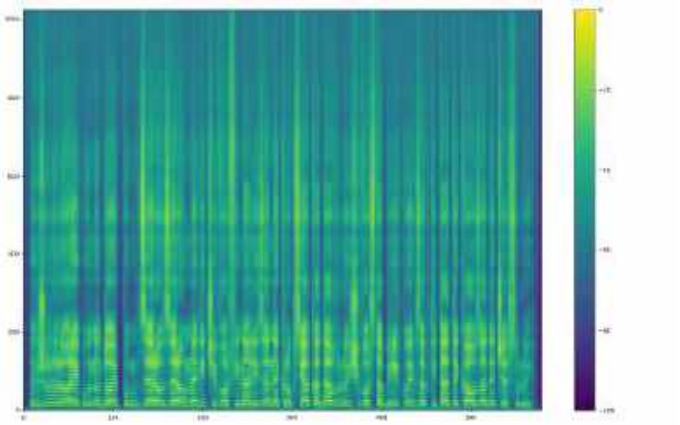

Figura 4.10: RSS spectrogramă

4.1.4 Baza de date proprie

Cost		
Parametru	Antrenare	Pas
Spectrograma Liniară	0.0363	46k
Spectrograma Mel	0.0305	46k
Total	0.0669	46k
Spectrograma Liniară(min)	0.0262	16k
Spectrograma Mel(min)	0.0257	16k
Total(min)	0.0519	16k

Tabela 4.6: Rezultate baza de date proprie

Cost			
Parametru	Validare(12s)	Validare(6s-train)	Validare(2s)
Spectrograma Liniară	0.0462	0.0402	0.0539
Spectrograma Mel	0.0533	0.0425	0.05737
Total	0.0996	0.0827	0.1114
Spectrograma Liniară(min)	0.0386	0.0377	0.0512
Spectrograma Mel(min)	0.0490	0.0427	0.0605
Total(min)	0.0876	0.0804	0.1134
Domeniu(e)	[0 50]	[0 60]	[0 40]
Domeniu(Hz)	[0 100]	[0 120]	[0 80]

Tabela 4.7: Rezultate validări

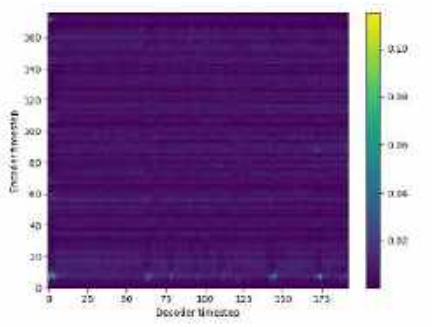

Figura 4.11: atenție(12s)(min)

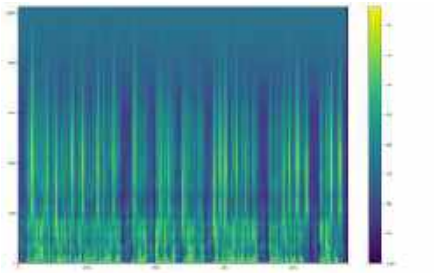

Figura 4.12: spectrogramă(12s)(min)

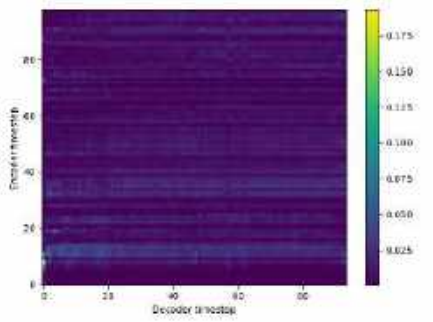

Figura 4.13: atenție(6s-train)(min)

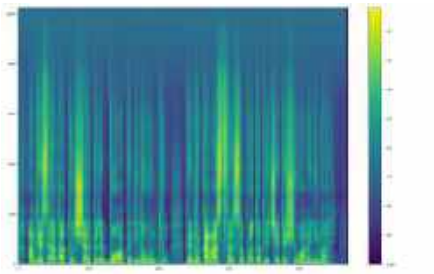

Figura 4.14: spectrogramă(6s-train)(min)

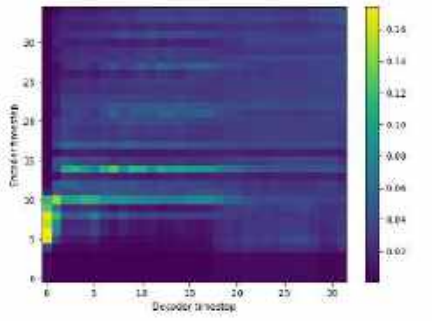

Figura 4.15: atenție(2s)(min)

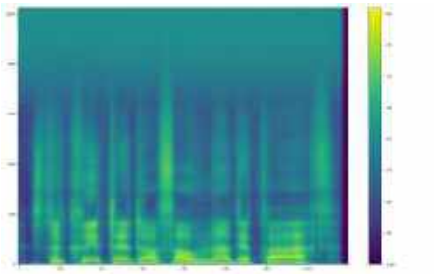

Figura 4.16: spectrogramă(2s)(min)

4.1.5 Observații

LJSpeech și RSS

Aceste baze de date au reacționat cel mai bine la rețeaua Tacotron. Se observă caracteristica accentuată și liniară a alinierii caracterelor la eșantione de spectrogramă (vezi fig.4.1 si fig.4.9. De asemenea și spectrogramele au reprezentarea cea mai corectă(vezi fig.4.2 si fig.4.10. Pentru *LJSpeech* domeniul de frecvență accentuat a fost înregistrat în intervalul [0 585]Hz, iar pentru *RSS* în intervalul [0 390]Hz. Având în vedere faptul că perfomanța autorilor a ajuns la intervalul [0 780]Hz [37], putem spune ca și rezultatele noastre sunt destul de bune. Rețeaua noastră poate scoate propoziții de calitate pentru secvențe lungi pentru ambele baze de date și răspunde bine și la testele de ambiguitate.

RSC

Pentru antrenarea rețelei doar cu cuvinte din baza de date RSC, minimele locale au fost atinse în alte puncte. În acele puncte atenția este distribuită mai uniform iar modul în care sunt rostite cuvintele este mai corect (vezi fig.4.3). Cu cât am lăsat mai mult rețeua la antrenat, atenția devine ușor neechilibrată, focusul sunetelor devenind mai accentuat doar în anumite zone. În consecință pronunția devine ușor perturbată. Totuși valoarea funcției cost este chiar mai mică ca la LJSpeech. Acest lucru se poate datora faptului că baza de date RSC a fost testată doar pe cuvinte pe când LJSpeech a fost testată pe propoziții lungi. Este mai ușor să generezi semnal audio pentru texte cât mai scurte. Prin urmare loss-ul mai mic al bazei de date RSC nu implică și rezultate mai bune față de LJSpeech. Antrenând rețeaua cu aceasta baza de date, rețeaua nu reușește să genereze voce clară pentru secvențe lungi. Mai mult, calitatea audio nici ea nu este foarte bună deoarece RSC este o bază de date pentru recunoaștere de voce nu sintetizare. Acest lucru se poate observa și din analiza spectrogramelor (vezi fig.4.4). Putem observa o plaja mare de frecvențe accentuate în intervalul [0 1170]Hz, lucru ce denotă zgomotul de frecvențe mari al semnalului produs.

La antrenarea doar cu propoziții din RSC, atenția este cel mai puțin accentuată deoarece baza de date conținea doar 1024 de propoziții pentru o singură voce (vezi fig.4.5). Calitatea audio aici este slabă dar totuși atenția rămâne liniară. Un alt lucru interesant este faptul că în punctele de minim, atenția nu mai era corect distribuită. Prin urmare chiar dacă funcția cost este una mai mică (calitatea audio mai bună), se poate genera totuși o distribuție de atenție greșită. Și aici domeniul de frecvență generat rămâne în intervalul [0 1170], multitudinea de frecvente reprezentând zgomotul ce are loc în semnal (vezi fig.4.6).

La antrenarea cu cuvinte și după cu propoziții rețeaua devine capabilă să vorbească propoziții mai lungi, cu o calitate mai bună decât rețeaua antrenată doar cu propoziții (se observă și din valorile funcțiilor de cost). Atenția este și ea una destul de accentuată însă nu reușește să scoată o pronunție foarte bună (vezi fig.4.7). Zgomotul din spectrogramă este același ca în cazurile anterioare (vezi fig.4.8)

Baza de date proprie

Pentru baza proprie de date rețeaua oferă un semnal audio destul de clar pe setul de test. Problema apare în cadrul atenției. Se poate observă că pentru o secvență de 12s atenția nu apare aproape deloc. Testată pe o propoziție de 6s din setul de antrenare se pot vedea niște linii drepte. Acest lucru arată faptul că rețeaua nu este capabilă să ia decizii în alinierea caracterelor cu eșantioanele de spectrogramă. Un posibil motiv ar fi:

- Inflexiuni de voce
- Baza de date neechilibrată (mult prea multe de 20 de sec)
- Baza de date are doar 800 de clipuri

- Erori la normalizare. Ambiguitate în înțelegerea caracterelor (ex: în loc de *întâi decembrie* este scris *unu decembrie*).
- Mici erori cauzate de diferența de nivel dB

Se observă de asemenea că accentuarea atenției crește în calitate odată cu scăderea numărului de cuvinte, însă totodată și costul crește. Plaja de frecvențe generată de aceste spectrograme se află în intervalul [0 120]Hz. Prin urmare rețeaua nu reușeste sa scoată sunete extrem de clare și nici să le alinieze corect.

Putem concluziona faptul că, chiar dacă avem o bază de date cu calitate bună de înregistrări, dacă totuși apar inflexiuni în voce (ce reprezintă trăsăturile supra segmentale ale semnalului vocal) și baza de date este neechilibrată, nu putem scoate o atenție bună. Comparativ cu RSC când am încercat să antrenăm doar cu cele 1024 de propoziții unde deși calitatea este mai slabă audio, totuși textele erau citite fără inflexiuni, iar clipurile audio erau mult mai echilibrate din punct de vedere al duratelor. Astfel RSC antrenată doar cu propoziții a putut genera atenție.

4.2 Rezultate rețea implementare proprie

4.2.1 Conversia spectrograma Mel spectrograma liniara

Implementare neurală

	Cost	
Limba	Rețea	Determinist
Engleză	0.0989	0.123
Română	0.0751	0.0951

Tabela 4.8: Rezultat conversie spectrograma

4.2.2 Observații

Rețeaua a fost mai întâi antrenată fără a implementa o funcție pentru rata de învățare și fără *gradient clipping*. Rezultatele spectrogramelor pentru limba engleză au fost destul de bune (fig.4.19 comparând cu *ground truth*-ul fig.4.17). Am încercat după sa testăm rețeaua și pe o spectrogramă pentru un clip în limba romană iar rezultatele au fost asemăntatoare cu cele pentru limba engleză (fig.4.23). În cele ce au urmat am încercat sa îmbunatățim rezultatele adăugând ca parametru de învățare *gradient clipping* însă rezultatele după cum se pot observa pentru ambele limbi (vezi fig.4.20 și fig.4.24) au fost mai slabe. Am decis să continuăm atunci cu modelul anterior. S-a încercat după și o metodă deterministă utilizând modulul *librosa* (vezi fig.4.18 și fig.4.22), iar rezultatele au fost și aici un pic mai slabe decât cele calculate cu rețeaua. Comparând spectrogramele originale cu rezultatele produse de rețea și cele deterministe putem observa că erorile cele mai multe apar la frecvențele înalte. Acest lucru se datorează faptului ca filtrele Mel au benzile cele mai largi când filtrează frecventele cele mai înalte și prin urmare pierderile sunt cele mai mari.

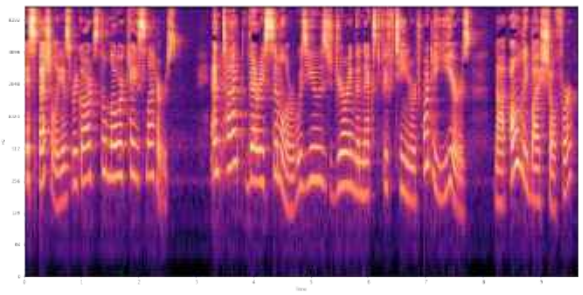

Figura 4.17: Engleza original

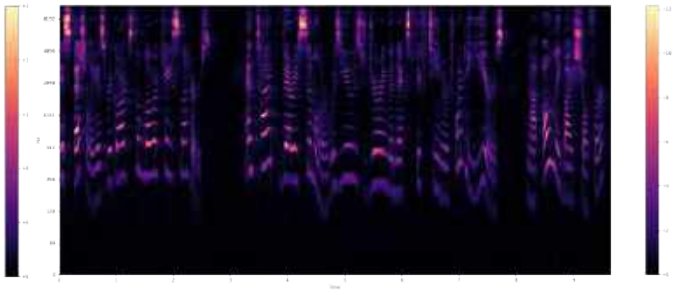

Figura 4.18: Engleză determinist

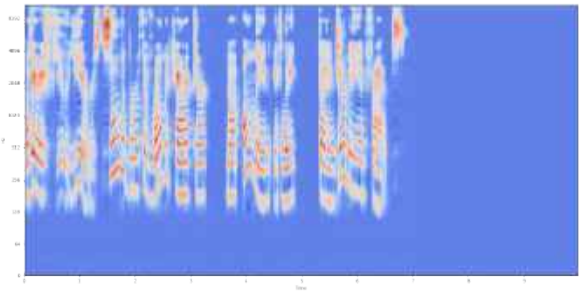

Figura 4.19: Engleză rețea

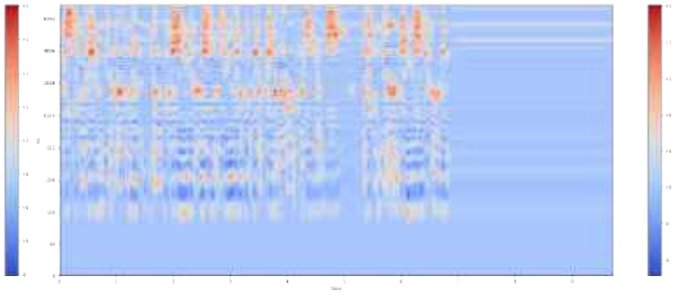

Figura 4.20: Engleză clipping

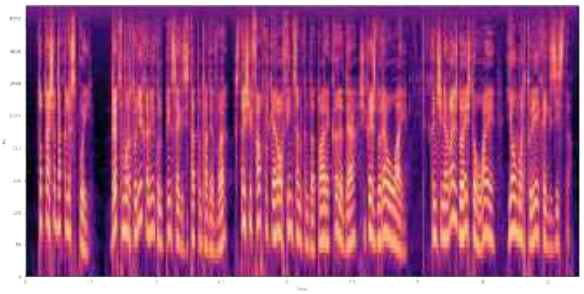

Figura 4.21: Română original

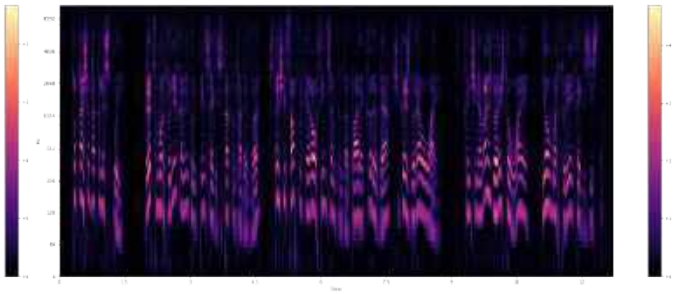

Figura 4.22: Română determinist

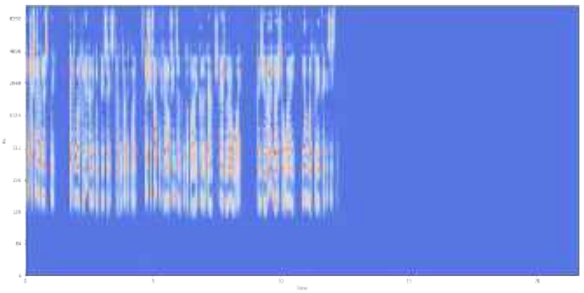

Figura 4.23: Română rețea

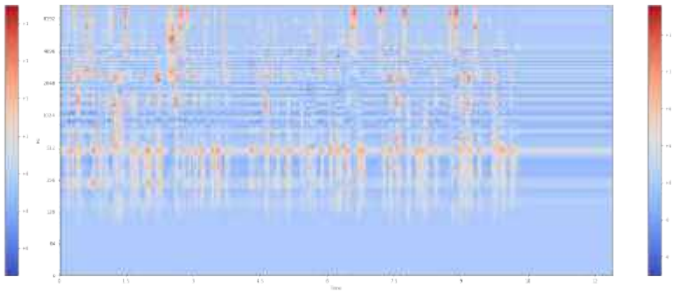

Figura 4.24: Română clipping

4.2.3 Conversia text în spectrogramă Mel

Pentru sarcina de transcriere din text în spectrogramă rezultatele nu au fost unele foarte bune. Am început să antrenăm mai întâi rețeaua cu toată baza de date cu un lot de 16 secvențe însă nu am obținut niciun rezultat. S-a încercat după testarea funcționalității rețelei pentru a verifica dacă este capabilă să învețe niște lucruri mai simple. Astfel i-am dat rețelei să învețe mai întâi o singură secvență. După, un singur lot de 16, după 128, 256 și implicit 1120. Pentru un număr mai mare ca 1120 rețeaua a început să nu mai învețe. Surprinzător, când am încercat să testăm cu niște propoziții din setul de test rețeaua reușea să scoată un semnal vocal. Totuși este bine de precizat că faza de testare nu a fost făcută în modul corect. Ci s-au oferit eșantioanele de spectrogramă la începutul fiecărei stări ale decodorului, rețeaua doar trebuind să prezică starea următoare. Nu am oferit starea prezisă anterior ca fiind intrarea stării următoare.

Următorul pas a fost să inspectăm atenția. Aceasta a fost rescrisă de la zero și am analizat pas cu pas ce se întâmplă. Am observat că matricile de ponderi W rămâneau mereu constante după fiecare pas. Nu reușeau să se updateze. State-urile ce ieșeau din codor mereu erau validate cu aceeași probabilitate. Alinierea în loc să fie o funcție liniară era o linie dreaptă. În consecință, rețeaua considera toate caracterele ca fiind importante pentru fiecare state din Codor.

Putem spune deci, că deși la ieșire rețeaua poate scoate destul de ușor un semnal vocal, fără atenție rețeaua nu va reuși să îmbunătățească calitatea înregistrărilor. Fără atenție nu există aliniere și deci semnal vocal distorsionat. În consecință nu putem concluziona că avem niște rezultate pentru transcrierea noastră din text în spectrogramă.

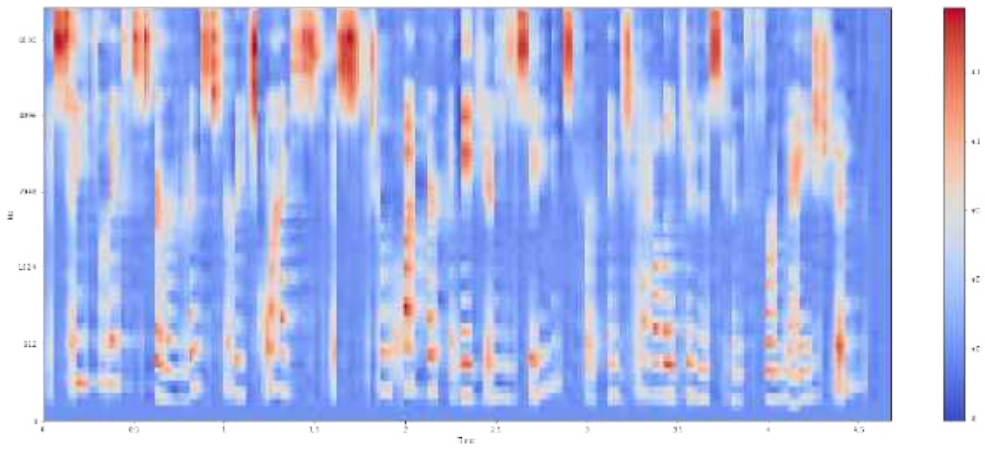

Figura 4.25: Transcriere din text în spectrogramă Mel

Capitolul 5
Concluzii

5.1 Concluzii generale

Proiectul de diplomă a presupus analiza mai multor baze de date, în principal pentru limba română, pentru a observa dacă putem într-adevăr implementa un sistem de transcriere în vorbire și pentru limba română cu tehnici *deep learning*. Analiza ne-a arătat că baza de date RSS poate îndeplini această sarcină. Calitatea audio este una chiar foarte bună, iar textul ce îl poate transcrie este de o lungime semnificativă. Poate deci fi folosit în implementarea pe un robot.

De asemenea, proiectul a mai presupus și încercarea unei implementări proprii. S-a reușit doar conversia spectrogramelor Mel în spectrograme Liniare. Conversia din text în spectrogramă Mel nu a fost un succes însă.

5.2 Contribuții personale

- Realizarea unei baze de date proprii
- Prelucrarea și pregătirea bazei de date pentru antrenare, folosind multiple scripturi în Python
- Evaluarea și compararea mai mult baze de date
- Dezvoltarea modulelor pentru rețeaua de transcriere a textului în vorbire:
  - Codorul
  - Atenția
  - Decodorul

De asemenea, tot codul sursă asociat îl puteți găsi la următorul link: TTS Romanian

5.3 Dezvoltări ulterioare

Posibile dezvoltări implică adăugarea unui modul *Vocoder* ce reușește să adapteze ușor rețeaua la o voce diferită. De asemenea, se dorește introducerea sistemului de transcriere pe un robot *Nao* și se va studia în detaliu motivul pentru care sistemul propriu nu a funcționat pe deplin.

Bibliografie

[1] https://github.com/ttaoREtw/Tacotron-pytorch.

[2] Speech chain. https://www.phon.ucl.ac.uk/courses/spsci/iss/week1.php.

[3] https://foneticalimbiiromane.ro/fonetica/.

[4] https://foneticalimbiiromane.ro/fonologia/.

[5] https://foneticalimbiiromane.ro/fonetica/sunetul/.

[6] https://foneticalimbiiromane.ro/fonetica/sunetul/vocala/.

[7] https://foneticalimbiiromane.ro/fonetica/sunetul/consoana/.

[8] https://foneticalimbiiromane.ro/fonologia/fonemul/.

[9] https://foneticalimbiiromane.ro/fonologia/diftong/.

[10] https://foneticalimbiiromane.ro/fonologia/triftong/.

[11] https://foneticalimbiiromane.ro/fonologia/hiat/.

[12] https://foneticalimbiiromane.ro/fonologia/accentul/.

[13] https://foneticalimbiiromane.ro/fonologia/silaba/.

[14] Jürgen Handke. *The structure of the Lexicon: human versus machine*, volume 5. Walter de Gruyter, 2012.

[15] https://wstyler.ucsd.edu/posts/scuse_me.html.

[16] https://ro.wikipedia.org/wiki/Deschidere_vocalic%C4%83.

[17] Leendert van Beek. Ling 201: Introduction to linguistics. 2010.

[18] D Stanomir. Semnale şi sisteme discrete. *Editura Athena*, 1997.

[19] Serbanescu Alexandru, Serban Gheorghe, Iana Vasile Gabriel, Oroian Teofil, and Rincu Iulian. Prelucrarea digitală a semnalelor.

[20] Michael Portnoff. Time-frequency representation of digital signals and systems based on short-time fourier analysis. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(1):55–69, 1980.

[21] Hristo Zhivomirov et al. On the development of stft-analysis and istft-synthesis routines and their practical implementation. *TEM Journal*, 8(1):56–64, 2019.

[22] https://raw.githubusercontent.com/musikalkemist/AudioSignalProcessingForML/master/15%20-%20Short-Time%20Fourier%20Transform%20explained%20easily/Short-Time%20Fourier%20Transform%20Explained%20Easily.pdf.

[23] https://raw.githubusercontent.com/musikalkemist/AudioSignalProcessingForML/master/17%20-%20Mel%20Spectrogram%20Explained%20Easily/Mel%20Spectrograms%20Explained%20Easily.pdf.

[24] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[25] http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture01-wordvecs1.pdf.

[26] http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture04-dep-parsing.pdf.

[27] Joakim Nivre. Incrementality in deterministic dependency parsing. In *Proceedings of the workshop on incremental parsing: Bringing engineering and cognition together*, pages 50–57, 2004.

[28] https://web.stanford.edu/~jurafsky/slp3/3.pdf.

[29] http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture05-rnnlm.pdf.

[30] Philipp Koehn. *Statistical machine translation*. Cambridge University Press, 2009.

[31] http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture07-nmt.pdf.

[32] https://nptel.ac.in/courses/117/105/117105145/.

[33] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[34] Sercan O. Arik, Mike Chrzanowski, Adam Coates, Gregory Diamos, Andrew Gibiansky, Yongguo Kang, Xian Li, John Miller, Andrew Ng, Jonathan Raiman, Shubho Sengupta, and Mohammad Shoeybi. Deep voice: Real-time neural text-to-speech, 2017.

[35] https://paperswithcode.com/method/griffin-lim-algorithm.

[36] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[37] Yuxuan Wang, RJ Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, et al. Tacotron: Towards end-to-end speech synthesis. *arXiv preprint arXiv:1703.10135*, 2017.

[38] Ovidiu Grigore. Curs rețele neuronale și sisteme fuzzy. http://ai.pub.ro/content/RNSF.htm.

[39] http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture03-neuralnets.pdf.

[40] Danqi Chen and Christopher D Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 740–750, 2014.

[41] http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture05-rnnlm.pdf.

[42] http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture06-fancy-rnn.pdf.

[43] https://www.deeplearningbook.org/contents/rnn.html.

[44] http://web.stanford.edu/class/cs224n/index.html#schedule.

[45] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

[46] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*, 2014.

[47] http://web.stanford.edu/class/cs224n/slides/cs224n-2021-lecture07-nmt.pdf.

[48] https://keithito.com/LJ-Speech-Dataset/.

[49] Alexandru-Lucian Georgescu, Horia Cucu, Andi Buzo, and Corneliu Burileanu. Rsc: A romanian read speech corpus for automatic speech recognition. In *Proceedings of The 12th Language Resources and Evaluation Conference*, pages 6606–6612, 2020.

[50] Adriana Stan, Junichi Yamagishi, Simon King, and Matthew Aylett. The romanian speech synthesis (rss) corpus: Building a high quality hmm-based speech synthesis system using a high sampling rate. *Speech Communication*, 53(3):442–450, 2011.

[51] https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly.

[52] https://www.linkedin.com/pulse/explanation-attention-based-encoder-decoder-deep-keshav-bhandari.

[53] https://medium.com/analytics-vidhya/neural-machine-translation-using-bahdanau-attention-mechanism-d496c9be30c3.

[54] https://machinelearningmastery.com/how-to-avoid-exploding-gradients-in-neural-networks-with-gradient-clipping/.