Separarea surselor de semnal folosind măști de frecvență

Contents

[1. Introducere 4](#_Toc11505447)

[2. Motivația temei 5](#_Toc11505448)

[3. Tehnologii folosite 6](#_Toc11505449)

[3.1. Python 6](#_Toc11505450)

[3.1.1. Python în domeniul de machine learning 6](#_Toc11505451)

[3.1.2. Distribuția Anaconda 7](#_Toc11505452)

[3.1.3. Jupyter Notebook 7](#_Toc11505453)

[3.1.4. Pachetul “numPy” 8](#_Toc11505454)

[3.1.5. Pachetul “PyTorch” 8](#_Toc11505455)

[3.1.6. Pachetul “LibROSA” 9](#_Toc11505456)

[3.2. Transformata Fourier 10](#_Toc11505457)

[3.2.1. Semnalele și caracteristicile sunetului 10](#_Toc11505458)

[3.2.2. STFT (short-time frequency tranform) 11](#_Toc11505459)

[3.2.3. Spectrograma 14](#_Toc11505460)

[3.2.4. ISTFT (inverse short-time frequency transform) 15](#_Toc11505461)

[3.2.5. Măști de frecvențe în timp 16](#_Toc11505462)

[3.3. Articol îndrumător “Low-latency sound source separation using deep neural networks” 18](#_Toc11505463)

[3.3.1. NMF 19](#_Toc11505464)

[3.3.2. Propunerea articolului 19](#_Toc11505465)

[3.3.3. Masca timp-frecvență 21](#_Toc11505466)

[3.3.4. Reconstruirea surse lor de semnal 21](#_Toc11505467)

[3.4. Deep learning 22](#_Toc11505468)

[3.4.1. Relația dintre dezvoltarea cognitivă și creierul uman 22](#_Toc11505469)

[3.4.2. Cum funcționează o rețea neuronală 23](#_Toc11505470)

[3.4.2.1. Ce este o rețea neuronală? 24](#_Toc11505471)

[3.4.2.2. Cum funcționează o rețea neuronală? 25](#_Toc11505472)

[3.4.2.3. De ce rețelele neurale sunt capabile să învețe? 26](#_Toc11505473)

[3.4.3. Rețele neurale profunde (Deep neural networks) 28](#_Toc11505474)

[3.4.4. Algoritmul de antrenare sau optimizatorul ADAM 29](#_Toc11505475)

[4. Descrierea soluției 30](#_Toc11505476)

[4.1. Obiective 31](#_Toc11505477)

[4.2. Implementare 31](#_Toc11505478)

[4.2.1. Construirea unui set de date de antrenare/validare/testare 32](#_Toc11505479)

[4.2.2. Încărcarea semnalelor și mixarea lor 34](#_Toc11505480)

[4.2.3. Structura rețelelor neurale 36](#_Toc11505481)

[4.2.3.1. Rețeaua formată doar din straturi dens conectate 36](#_Toc11505482)

[4.2.3.2. Rețeaua formtă din straturi convoluționale și dens conectate 37](#_Toc11505483)

[4.2.3.3. Straturile de tip fully connected (dens conectate) 38](#_Toc11505484)

[4.2.3.4. Straturile convoluționale 40](#_Toc11505485)

[4.2.3.5. Batch normalization 40](#_Toc11505486)

[4.2.4. Workflow-ul rețelei neurale 43](#_Toc11505487)

[4.2.5. Funcția de cost (MSE) 44](#_Toc11505488)

[4.2.6. Sigmoida vs ReLU 45](#_Toc11505489)

[4.2.7. Antrenarea rețelei 48](#_Toc11505490)

[4.2.7.1. Pasul de propagare înainte (feedforward) 48](#_Toc11505491)

[4.2.7.2. Pasul de propagare înapoi (backpropagation) 49](#_Toc11505492)

[4.2.8. Testarea rețelei 50](#_Toc11505493)

[4.3. Rezultate și performanțele rețelei 50](#_Toc11505494)

[4.3.1. Performanțe în funcție de funcția de activare 50](#_Toc11505495)

[4.3.2. Performanțe în funcție de împărțirea setului de date 50](#_Toc11505496)

[4.3.3. Performanțe în funcție de numărul de epoci 50](#_Toc11505497)

[5. Concluzii 50](#_Toc11505498)

[6. Index figuri 50](#_Toc11505499)

[7. Bibliografie și webografie 51](#_Toc11505500)

# Introducere

Separarea surselor de semnal folosind măști de frecvență urmărește faimoasa problemă cunoscută sub numele de „cocktail party” sau „blind source separation”.

Efectul de „cocktail party” constă în abilitatea creierului de a se focusa preponderent asupra unei voci, surse de semnal sau conversație, într-o cameră plină de oameni, muzică și alte sunete de fundal, astfel încât să filtreze restul sunetelor care nu reprezintă un interes pentru interlocutor. Ascultătorii au abilitatea de a segrera multiple voci în surse diferite și, subconștiental, să decidă care dintre surse reprezintă interes într-un procentaj mai mare.

Lucrarea este compusă din cinci capitole, care împreună descriu pașii urmați pentru atingerea scopului propus.

Pentru început este prezentat capitolul „Introducere”, în care este descris, pe scurt, în ce constă proiectul, tema propusă și ce conține documentația actuală.

Al doilea capitol, „Motivația temei”, prezintă ce probleme sunt întâmpinate pentru oferirea unei separări cât mai corecte a celor două surse de semnal.

Al treilea capitol, „Tehnologii folosite”, enumeră și detaliază ce tehnologii au ajutat la dezvoltarea proiectului curent. Tot aici este prezentat, pe scurt, articolul suport pe care s-a bazat proiectul.

În capitolul „Descrierea soluției”, este descris modul în care tehnologiile au fost îmbinate cu articolul îndrumător pentru realizarea aplicației finale. Lucrarea urmărește comparația a două rețele neurale din punct de vedere al performanței pentru combaterea fenomenului de „overfitting”. Sunt prezentate și etapele prin care trec sursele de semnal: de la sunetele inițiale, nealterate, se creează o mixtură între cele două semnale (voci), urmând antrenarea rețelei neurale profunde din care va rezulta o mască. Această mască va fi aplicată pe mixtura celor două surse de semnal, rezultând două întregistrări în care cele două surse sunt separate una de cealaltă.

Acestă aplicație a fost realizată cu ajutorul domnului lector universitar doctor Gâlmeanu Honorius.

# Motivația temei

Problema de tip „cocktail” a fost populară din 1953, iar în ultima vreme a fost considerată a fi o mare provocare pentru rețelele neurale. Aceasta constă în filtrarea sunetelor astfel încât să se audă doar ce este de interes pentru interlocutor. Problema, în mod intrinsec, este destul de dificilă, și mereu a fost arătat un interes deosebit de mare pentru modul în care oamenii reușesc să o rezolve, în mod natural, anatomic.

Articolul întrumător al acestei lucrări propune o abordare folosind o rețea neuronală profundă, formată din trei straturi de tip „fully connected”. Pentru propunerea articolului era nevoie de o putere de calcul foarte mare, lucru de care nu beneficiam, de aceea, pentru cercetarea subiectului si temei în sine, am decis elaborarea unei comparații între două structuri diferite de rețele neurale pentru combaterea fenomenului de „overfitting”.

În urma elaborării lucrării, s-a observat că rețeaua propusă de articol nu este una din cele mai puternice pentru obiectivele propuse, dar oferă un rezultat decent pentru surse de semnal înregistrate într-o cameră fără alte sunetele de fundal. De aceea, s-a urmărit și compararea cu o altă structură de rețea, și anume rețeaua neuronală cu două straturi convoluționale pentru extragerea caracteristicilor și două straturi de tip „fully connected”.

La finalul lucrării, s-au comparat rezultatele obținute pentru cele două rețele și pentru două tipuri diferite de creare a setului de date, din cauza insuficienței de putere de calcul necesară pentru implementarea rețelei în modul în care articolul îndrumător propunea.

# Tehnologii folosite

## Python

### Python în domeniul de machine learning

Pythonul este considerat noul Fortran în lumea stiinței. Aparent, este foarte popular în lumea oamenilor de știință, care nu sunt programatori, și oferă utilizatorilor un număr mare de biblioteci pentru orice tip de problemă de programare.

Este foarte ușor de învățat, sau cel puțin lucrurile de bază sunt ușoare de învățat. În acest mod, limbajul a devenit foarte popular și folosit de cei care nu aveau trecut ca programatori. Spre deosebire de limbajele de programare ca Java sau C#, care sunt catalogate ca fiind complicate, grele de învățat, se adaugă și nevoia de a învăța să lucrezi cu medii de lucru sau infrastructuri precum Eclipse, Maven, sau alte sisteme de punere în folosință a programelor.

Un alt altuu este că Pythonul oferă un mix de stiluri, iar pentru a scrie un program nu este nevoie să existe contact cu programarea orientată pe obiecte, așa cum este nevoie în Java sau C#, deoarece sunt incorporate mai multe paradigme de programare precum : funcțional, orientat pe obiecte, orientat pe aspect și programare structurală.

Motivele prezentate mai sus fac parte din lista de motive pentru care Pythonul este un limbaj de programare indicat pentru învățarea de „machine learning”, deoarece oamenii care doresc să exploreze acest domeniu pentru lucrări de cercetare nu sunt împedicați de sintaxa sau infrastructura unui limbaj.

O sarcină importantă a unui programator sau inginer care lucrează cu rețele neurale este să extragă, să proceseze, să definească, să curețe, să aranjeze și să înțeleagă datele pentru dezvoltarea unui algoritm inteligent. Astfel, acest limbaj este o alegere bună deoarece este ușor de înțeles, iar în acest mod, se poate trece la lucru mult mai repede.

Un alt motiv este faptul că Python are un număr mare de biblioteci implementate apriori pentru aproape orice domeniu. Există biblioteci pentru procesare de imagini (numpy, opencv, scikit), procesare de text (nltk, numpy, scikit), procesare audio (librosa), rezolvare de probleme de tip „machine learning” (pandas, scikit), vizualizare a datelor (matplotlib, seaborn, scikit), folosirea învățării profunde sau deep learning (tensorflow, pytorch), calcul științific (scipy), integrare de aplicații web (Django), și așa mai departe. Partea bună a acestor biblioteci este că nu există o curbă de învățare, ci odată ce se înțeleg lucrurile de bază ale limbajului se poate începe implementarea, folosindu-se bibliotecile doar prin importarea lor și apelare funcțiilor necesare.

Motivul principal pentru care Pythonul nu este folosit în absolut orice este faptul că folosește foarte mult spațiu și nu este cel mai rapid limbaj, dar acesta nu a fost creat pentru rapiditate sau menajarea memoriei, ci pentru ușurința cu care poți implementa orice. Dacă se dorește rapiditate și menajarea memoriei într-un mod cât mai eficient se poate folosi C++ sau C.

În majoritatea cazurilor, când se implementează diferiți algoritmi de rețele neurale se folosește Python (tensorflow sau pytorch), dar pentru implementarea în sisteme reale, unde viteza contează, se folosește C sau C++.

### Distribuția Anaconda

Anaconda este o distribuție gratuită și „open – source” pentru programarea în Python sau R a programelor de calcul stiințific ( știința datelor, „machine learning”, procesare de date cu volum mare, și altele), care dorește simplificarea administrării pachetelor și implementării.

Anaconda are peste 1.500 de pachete care ajută programatorul să implementeze ce dorește, fără să se piardă în detalii.

Alt atuu este faptul că Anaconda colectează date din fișiere, baze de date și alte date pe care le are salvate într-un cloud.

### Jupyter Notebook

Jupyter Notebook extinde consola Pythonului pentru un mediu mult mai prietenos de programare. Acesta oferă o aplicație web capabilă să cuprindă tot procesul computațional, și anume dezvoltarea, documentarea, executarea codului și comunicarea rezultatelor aferente programelor. În acest mediu de programare sunt combinate două componente :

* aplicația web, o unealtă interactivă ce poate fi folosită în browser
* documentele Notebook, o reprezentare al întregului conținut vizibil în aplicația web, incluzând intrările și ieșirile calculelor, text explicit, structuri matematice, imagini și reprezentări media ale obiectelor

### Pachetul “numPy”

NumPy este un pachet fundamental pentru calculul științific în Python. Conține printre multe alte lucruri și :

* suport pentru obiecte vectori N – dimensionale puternice
* funcții sofisticate care ajută la vectorizarea calculului
* unelte pentru integrarea codului C, C++ sau Fortran
* algebră liniară folositoare, transformata Fourier, capabilități aleatoare pentru numere

Pe lângă aceste atuuri, NumPy poate fi folosit, de asemenea, ca un container multidimensional eficient pentru datele generice. Astfel, datele pot fi definite în mod arbitrar. Acest lucru permite bibliotecii NumPy să se integreze rapid și fără alte probleme cu o varietate mare de baze de date.

### Pachetul “PyTorch”

PyTorch este un software, mai specific un pachet sau o bibliotecă pentru probleme de „machine learning” , dezvoltat pentru limbajul Python, bazat pe biblioteca Torch, care este folosită pentru aplicații precum procesarea de limbaj natural.

Pachetul a fost dezvoltat de grupul de cercetare de inteligență artificială de la Facebook.

Tensorii, spre deosebire de înțelesul lor în matematică, sunt diferiți în programare, unde sunt tratați ca simple structuri de date, precum vectorii multidimensionali. Acești tensori, care sunt similari vectorilor NumPy, diferă prin faptul că pot fi folosiți pe plăcile grafice GPU care suportă CUDA.

Acest pachet a fost folosit în aplicație pentru :

* crearea de tensori din liste / vectori de timp numPy
* crearea rețelei neurale :
  + straturile dens conectate
  + straturile convoluționale
  + pentru normalizarea datelor
  + pentru aplicarea funcției de activare ( sigmoida, ReLU)
  + pasul de propagare înainte
  + pasul de propagare înapoi
* salvarea și încărcarea modelelor de rețea după ce procesul de învățare a încetat
* golirea cache-ului din CUDA
* verificarea dacă este disponibilă CUDA

### Pachetul “LibROSA”

LibROSA este un pachet din Python pentru analiză de muzică și audio. Oferă multe funcționalități deja implementate, care sunt necesare pentru dezvoltarea sistemelor care doresc să obțină informații din muzică, înregistrări audio.

Acest pachet a fost folosit în aplicație pentru :

* încărcarea semnalelor audio în program
* salvarea semnalelor audio după ce au fost separate de mix
* calcularea transformatei Fourier pe termen scurt
* calcularea inversei transformatei Fourier pe termen scurt

## Transformata Fourier

Din punct de vedere matematic, transformata Fourier reprezintă operația aplicată asupra unei funcții cu elemente complexe, și returnează tot o funcție cu elemente complexe. Ce este diferit la funcția returnată este că este reorganizată după frecvențele componente.

Analiza Fourier este considerată abecedarul analizei semnalelor. Aceasta tranformă semnalul nostru din domeniul timpului în domeniul frecvențelor. În timp ce domeniul timpului exprimă semnalul într-o secvență de probe/bucăți de semnal, domeniul frecvențelor reprezintă semnalul ca sinusoidale, cu diferite magnitudini, frecvențe, și decalaje de fază a semnalului.

### Semnalele și caracteristicile sunetului

Semnalul în timp este reprezentat printr-o anumită funcție, care de regulă dă forma de dependență a nivelului sau amplitudinii semnalului în raport cu timpul.

Semnalele pot fi de două feluri, periodice și neperiodice. Cele periodice sunt caracterizate printr-o perioada care se tot repetă în decursul semnalului, iar cele neperiodice nu conțin această perioadă de repetiție.

La rândul lor, semnalele perioduce sunt de două feluri, armonice și nearmonice.

Semnalul audio este un semnal neperiodic, de cele mai multe ori, fiind determinat de sunete neperiodice. Din punct de vedere electric, semnalul audio este reprezentat de variația în timp a tensiunii sau curentului electric.

Sunetul este caracterizat de patru elemente:

* amplitudinea – caracteristica undei sonore percepute ca volum
* frecvența/înălțimea sunetului – numărul de perioade ( T -> s) sau oscilații efectuate

( F -> HZ) într-un timp dat

* intensitatea sunetului/sonoră – este măsurată în decibeli
* timbrul – calitatea de bază a fiecărui sunet, determinat de componenta de armonici

În procesare semnalelor se ține foarte mult cont și sunt controlate frecvența și intensitatea sunetelor. În tema aleasă toate înregistrările sunt modificate la 16KHz, iar analiza pentru aflarea măștilor de frecvență se bazează pe ideea de „frecvențe în timp”.

### STFT (short-time frequency tranform)

Transformata Fourier pe termen scurt (STFT) este o transformare de tip Fourier care ajută la determinarea frecvenței și fazei sinusoidalei a secțiunilor locale ale unui semnal, având în vedere schimbarea lui în timp.

Pentru calculul STFT se împarte un semnal mai lung în segmente mai scurte, de dimensiuni egale, iar mai apoi se calculează tranformata Fourier, separat, pe fiecare segment mai scurt. La final, aceasta duce la aflarea spectrului Fourier pentru fiecare segment mai scurt. După se poate arăta spectrul, printr-un desen, ca o funcție a timpului.

Astfel, tranformata Fourier pe termen scurt este o secvență de transformări Fourier a unui semnal ferestruit. Pentru această transformare trebuie setați mai mulți parametrii, și anume:

* fereastra de analiză ( de exemplu 10/20/30 ms)
* valoarea de „overlap”/„hopping”, cât de mult se dorește suprapunerea ferestrelor în mutarea către următoarea fereastră ( de exemplu overlap de 50%, pentru dimensiunea ferestrelor de 10 ms se vor lua ferestre în felul următor:

0 ms → 10 ms, 5 ms → 15 ms, 10 ms → 20 ms, și așa mai departe)

* funcția de ferestruire ( de exemplu Hann, Gaussian, etc. )
* generarea segmentelor ferestruite
* aplicarea tranformatei Fourier pe fiecare segment ferestruit

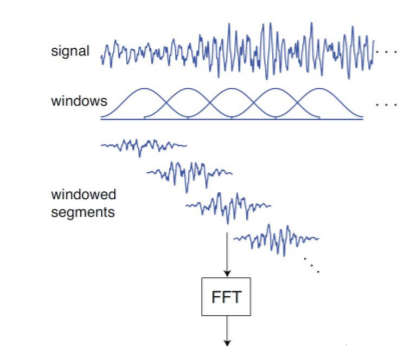


Figura 1 : Pașii aplicării transformatei Fourier pe un semnal

Aplicarea algoritmului de transformare Fourieri pe termen scurt constă în respectarea unor pași, și anume:

1. Se alege un segment de dimensiune mică din întregul semnal:

* se folosește conceptului de „sliding window” (fereastră care se plimbă) pentru a fi mai ușor de vizualizat ce se întâmplă
* datele dinăuntrul ferestrei reprezintă segmentul curent care va fi procesat
* dimensiunea ferestrei rămâne aceeași pe tot parcursul procesării datelor, dar punctul de pornire („offsetul”) se va schimba după fiecare pas din algoritm
* de obicei, schimbarea acestui „offset” constă într-o valoare mai mică decât dimensiunea ferestrei, pentru a face posibil fenomenul de „overlap”, astfel încât fereastra precedentă și cea curentă se vor suprapune

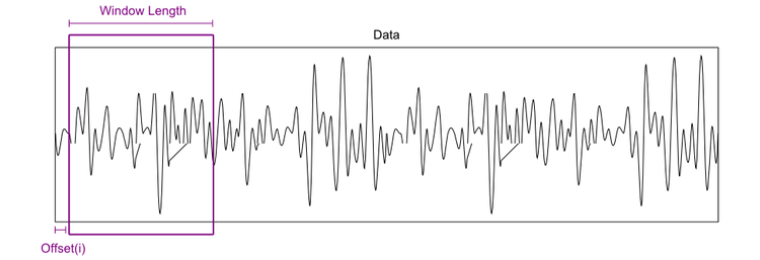


Figura 2 : Fereastră care se plimbă peste un semnal

1. Se înmulțește segmentul cu o funcție de jumătate de cosinusoidă:

* folosită pentru a evita problemele create de segmentarea unui semnal
* în momentul în care tăiem un semnal în bucățele mai mici, marginile creează o tranziție ascuțită care nu exista înainte în semnal
* înmulțind cu această funcție „jumătate cosinusoidală”, ajută la atenuarea semnalului, astfel încât tranzițiile la margini să nu afecteze transformata Fourier a datelor

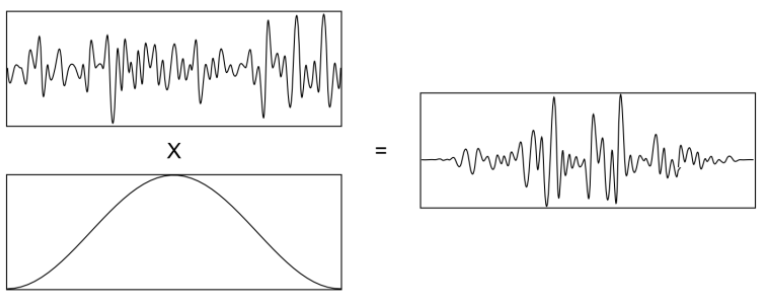


Figura 3 : Datele unui semnal după aplicarea funcției "half-cosine"

* alt atuu al acestei funcții, este că având un overlap de 50%, suma tuturor ferestrelor este o constată egală cu 1; acest lucru înseamnă că înmulțind datele noastre cu o anumită funcție, nu se vor adăuga diferențe în amplitudinea semnalului original

1. se adaugă zerouri la finalul segmentului
2. aplicarea transformatei Fourier asupra segmentului și normalizarea în frecvențe pozitive și negative:

* numărul the bucăți de date care intră este egal cu numărul de bucăți de frecvențe care ies
* datele date de transformarea Fourier trebuie să fie scalate cu numărul de bucăți din transformare pentru a menține egală energia semnalului ( în concordanță cu teorema lui Parseval)
* transformata Fourier returnează atât frecvențele pozitive, cât și cele negative

1. combinarea energiilor frecvențelor pozitive și negative la un loc și crearea unui grafic
2. scalarea spectrului rezultat în dB :

* convetirea datelor în dB are efectul de a lungi vârfurile frecvențelor în jos, spre nivelul mediu al sunetului, și de a aduce valorile foarte joase în sus.
* acest lucru permite compararea conținutului la toate nivelele de amplitudine

1. înlăturarea zgomotului care se află peste o anumită limită:

* tot ce este sub -40 dB este sub limita minimă a frecvențelor, astfel încât se consideră ca aceste frecvențe sunt erori numerice din algoritm

### Spectrograma

„Spectrograma este reprezentarea vizuală a unui spectru (a densității spectrale) de frecvențe a unui sunet în raport cu timpul sau cu alte variabile [[1]](#footnote-1).”

Aceasta este de obicei asociată cu un „heat map”, o imagine cu insitatea dată de varietatea culorilor și a luminozității.

Formatul general este un grafic cu 2 dimensiuni plus una separată pentru amplitudine:

* o axă reprezintă timpului, cea orizontală
* o axă reprezintă frecvențele, cea verticală
* o a treia dimensine ar fi cea a amplitudinilor frecvențelor la un anumit timp, care sunt reprezentate de intensitatea de culoare a acelui punct

Există multe moduri pentru desenarea graficul spectrogramei, uneori axele orizontale și verticale sunt inversate, astfel încât timpul se desfășoară în sus și în jos, alteori graficul este de tip „waterfall” unde amplitudinea este reprezentată de înălțimea unei suprafețe 3D în locul intensității sau culorii. Axele de frecvență și amplitudine pot fi ori liniare, ori logaritmice, în funcție de scopul spectrogramei. Semnalele audio sunt reprezentate de cele mai multe ori cu o axă logaritmică a amplitudinii (decibeli, dB) și frecvențele ar putea fi liniare pentru a menține relația armonică, sau logaritmică pentru a menține relațiile muzicale/tonurile.

În continuare sunt prezentate trei spectrograme, folosite în aplicație. Prima spectrogramă este mixul dintre următoarele două. Spectrograma a doua și a treia sunt două voci diferite, una de femeie și una de bărbat. Cum este specificat mai sus, pe axa orizontală avem timpul, pe cea verticală sunt frecvențele, iar în dreapta avem o scară pentru măsurarea amplitudinii în dB.

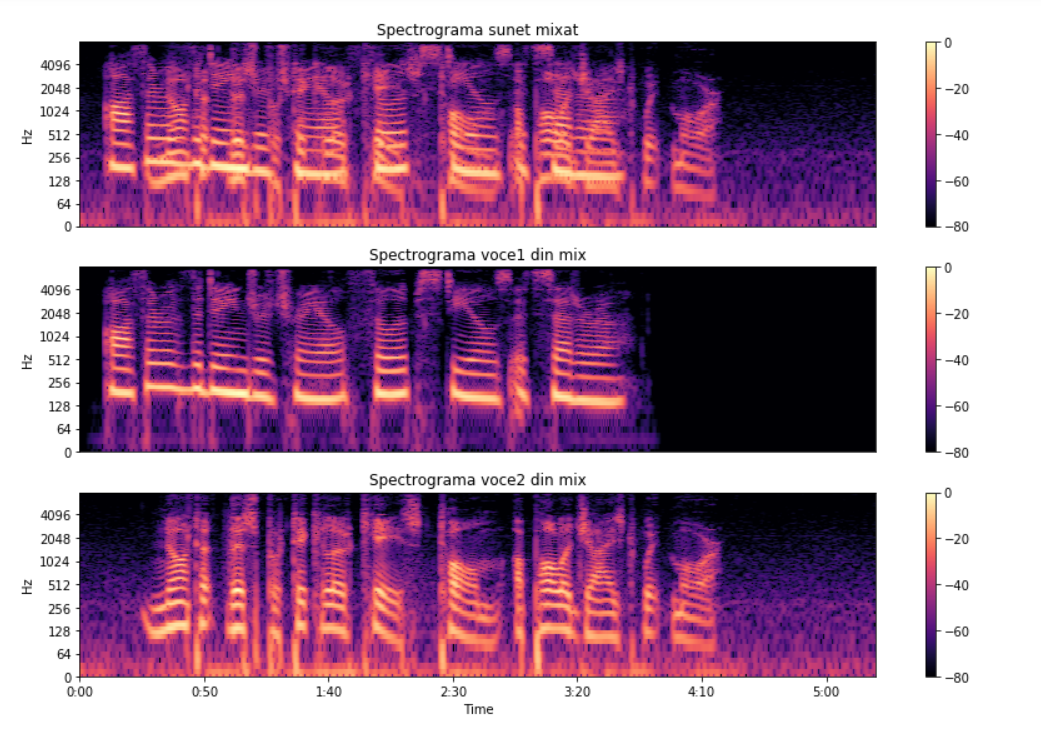


Figura 4 : Spectrograme folosite în aplicație pentru două voci și mixul lor

Pentru calculul spectrogramei se va calcula magnitudinea la pătrat, unde magnitudinea este valoarea absolută: spectrogram(f, t) = | STFT(t, f) |2.

### ISTFT (inverse short-time frequency transform)

Transformata Fourieri pe termen scurt este o operație inversabilă, însemnând că după aplicarea transformării Fourier pe termen scurt asupra semnalului audio, acesta poate fi recuperat folosind operația de transformare inversă Fourier, dar doar atâta timp cât matricea respectivă mai conține elementele complexe (faza semnalului). Dacă se face transformarea matricii către spectrogramă/magnitudine, din acel punct transformarea inversă Fourier nu mai are niciun efect, deoarece elementele complexe ale semnalului s-au pierdut în momentul aflării valorii absolute pentru magnitudine.

Dacă se dorește să se modifice spectrul complex al semnalului, acesta mai poate fi recuperat folosind metoda „overlap – add (OLA)”.

### Măști de frecvențe în timp

Se presupune că avem două surse, un semnal pentru voce și un semnal pentru muzică. Presupunem că mixul celor două surse este risipit, adică majoritatea punctelor de tip frecvență – timp sunt nule.

Presupunem că primul semnal, muzica, are majoritatea punctelor de energie joasă, iar al doilea semnal are majoritatea punctelor de energie înaltă.

Se mai presupune și că cele două semnale sunt disjuncte, adică punctele lor de tip frecvență – timp nu se suprapun, sau se suprapun foarte puțin.

Având mixul risipit și disjuncția, cele două surse din mix se pot discrimina destul de ușor. Punctele care au probabilitatea să aparțină de o sursă sunt marcate cu valoarea constată „1”, iar restul cu „0”. Astfel rezultă o mască binară (varianta mai riguroasă a algoritmului).

După calcularea măștii, prin înmulțirea măștii cu sursa mixată, se va obține sursa pentru care s-a calculat masca.

Figura 5 : Calcularea măștii de frecvență cu respect pentru prima sursă de semnal

În majoritatea cazurilor, mixurile nu sunt foarte risipite și nici sursele disjuncte. Punctele din mix care sunt probabil să aparțină unei surse au valori aproape de valoarea 1, iar restul aproape de valoarea 0 ( varianta de „soft – masking”). Ecuația generală, ecuația cu respect pentru prima sursă de semnal din mix, cât și ecuația cu respect pentru a doua sunt următoarele :

Figura 6 : Calcularea măștii de frecvență cu respect pentru ambele surse de semnal

În formule, „t” reprezintă o fereastră de procesare anume (vezi secțiunea [„3.3.2. Propunerea articolului](#_Propunerea_articolului)”), iar „f” este indexul blocului din transformata discretă Fourier (STFT).

„” și „” sunt vectorii transformatei Fourieri pe termen scurt cu caracteristici, ale celor două surse de semnal mixate. Valorile măștii sunt în intervalul [0,1], asigurând astfel stabilitatea numerică și ajutând pasul de propagare înapoi al rețelei.

Având masca pentru o sursă, pentru cealaltă sursă masca se poate calcula fie în același mod, dar înlocuind numărătorul cu sursa pentru al doilea semnal, fie scăzând masca dintr-o matrice construită, care va avea aceeași dimensiune și va conține doar valori de 1.

## Articol îndrumător “Low-latency sound source separation using deep neural networks”

Problema de tip „cocktail” a fost populară din 1953, iar în ultima vreme a fost considerată a fi o mare provocare pentru rețelele neurale. Aceasta constă în filtrarea sunetelor astfel încât să se audă doar ce este de interes pentru interlocutor. Problema, în mod intrinsec, este destul de dificilă, și mereu a fost arătat un interes deosebit de mare pentru modul în care oamenii reușesc să o rezolve, în mod natural, anatomic.

Problema nu este unică pentru oameni, și alte specii se confruntă cu probleme similare. De exemplu, multe specii de broaște folosesc vocalizări, specifice speciei lor, pentru a-și localiza perechea noaptea, când vizibilitatea este scăzută, cu foarte puține repere vizuale, și multe alte broaște în jur.

Există două provocări distincte, din punct de vedere conceptual, pentru un ascultător în această situație.

Prima provocare este segregarea sunetului. Sunetele într-o scenă, toate unele peste altele, generează semnalul care intră mai departe în ureche. Totuși, această mixtură oferă în sine puțină informație pentru organism, care în sine este interesat într-o sursă de semnal particulară. Sistemul auditiv trebuie să derive proprietățile sunetelor individuale din mixtura care intra în ureche.

A doua provocare este alocarea atenției doar asupra sursei de semnal care prezintă interes, ignorând celelalte, și schimbarea atenției între surse, în timp ce se urmărește o conversație cu două surse de semnal diferite. Majoritatea proceselor noastre cognitive pot opera doar pe un singur lucru o data, în consecință alegem o singură sursă de semnal pe care să ne putem focusa.

Urechea umană face o descompunere a frecvențelor pe sursele de semnal, similar unei transformate Fourier, cu ajutorul căreia se poate verifica spectrograma frecvențelor sunetului.

Separarea sursei de semnal la o latență scăzută necesită ca fiecare input, o fereastră audio, să fie procesată la o temporizare cât mai mică, iar outputul să fie generat cât mai repede.

Articolul propune o rețea neuronală profundă, deep neural network (DNN), cu latență scăzută. Această metodă are avantajul de a folosi un context extins „din trecut”, mai apoi generându-se filtre/măști de tipul „soft time frequencies masks”, care sunt aplicate peste sunetul mixat, astfel încât să fie oferită o performanță de separare mai bună, comparând cu „NMF baseline”.

### NMF

Separarea surselor poate fi văzută ca o problemă de factorizare a matricei, unde sursa mixtură este modelată ca o matrice care conține reprezentarea spectrogramei aferente.

Spectrogramele sunt de obicei folosite pentru vizualizarea timpului cu variația spectrală

a densității unei înregistrări audio. Semnalele audio pot, astfel, să fie reprezentate în întregime de o matrice cu :

* linii, axa orizontală ( reprezentarea timpul)
* coloane, axa verticală ( reprezentarea frecvențelor)
* valori ale elementelor, care corespund intensitatății sau culorii fiecărui punct din imagine ( indicând amplitudinea unei frecvențe particulare la un anumit moment „t”).

Spectrograma unui semnal x(t) poate fi estimată prin calculul magnitudinii la pătrat a transformatei Fourier (prezentată anterior la secțiunea [3.2.3 Spectrograma](#_Spectrograma)) a semnalului x(t), astfel încât x(t) poate fi reconstruit din spectrogramă prin inversa transformatei Fourier (prezentată de asemenea în secțiunea [3.2.4 ISTFT (inverse short-time frequency transform)](#_ISTFT_(inverse_short-time) ), după procesarea semnalului în domeniu spectral.

### Propunerea articolului

Separarea surselor de semnal țintește recuperarea sunetelor individuale dintr-un mix, compus din cele două sunete originale ale surselor diferite. Unele domenii care folosesc această tehnologie sunt : recunoașterea vocală, transpunerea fonetică a muzicii, scoaterea zgomotelor de fundal dintr-un discurs și multe alte aplicații care implică auzul și sursele de semnal audio.

Sunt două metode populare de a aborda separarea surselor de semnal, și anume abordarea modelelor compoziționale, precum factorizarea matricelor negative sau, cumva echivalent, analiza componentelor probabilistic latente și metodele care includ rețelele neurale profunde.

Metodele compoziționale descompun mixuri acustice complexe în mixuri liniare ale unor subunități mai simple sau componente. Pe de altă parte, rețelele neurale profunde sunt modele esențiale liniare, capabile să învețe mapări de tipul intrare – ieșire neliniare complexe, cu relația dintre cele două, intrarea și ieșirea, fiind mapată în ponderile păstrate în straturile ascunse. Tehnicile bazate pe rețelele neurale profunde se extind tot mai mult și s-a descoperit că au performanțe mult mai bune decât modelele compoziționale.

Pentru această separare s-a folosit o abordare care include învățarea supervizată, în care ferestre scurte de mască au fost generate bazându-se pe factorizarea contextului din trecut pentru prezicerea ponderilor și rezultând într-un filtru de separarea cu un singur canal.

În acest mod s-au generat seturile de antrenare, validare și testare (prezentare mai explicită la secțiunea [4.2.1 Construirea unui set de date deantrenare/validare/testare](#_Construirea_unui_set)).

Această tehnică are rezultate în momentul în care fereastra de procesare este foarte mică, de exemplu 5/10/20 milisecunde, nu mai mult.

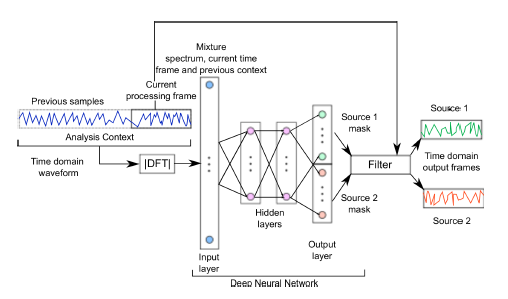


Figura 7 : Ilustrare schematică a rețelei neurale profunde propuse

Caracteristicile spectrale (spectrogramele) derivate din mixul acustic sunt folosite ca date de intrare pentru rețeaua neuronală profunda (DNN – deep neural network). La final măștile de timp – frecvență voi fi prezise de rețea ca date de ieșire. Aceste măști de timp – frecvență sunt aplicate, mai apoi, pe spectrul mixului pentru a obține spectre individuale pentru reconstruirea estimativă a surselor de semnal.

Articolul propune împărțirea semnalului în blocuri pentru procesare. Pentru a păstra latența joasă, blocul de semnal este foarte scurt, referit mai departe ca fereastră de procesare (PF – processing frame). Latența este determinată de dimensiunea acestui bloc de procesare, având în vedere că toate probele sunt păstrate în memorie înainte să poată fi aplicată transformata Fourier pentru obținerea reprezentării spectrale.

Se propune utilizarea unui context temporal anterior mai lung pentru generarea datelor de intrare pentru rețea, care sunt corespunzătoarea ferestrei de procesare curente. Acest context temporal anterior este referit ca fereastra de analiză ( AF – analysis frame). Mai departe, caracteristicile spectrale rezultate din aceste ferestre de analizare vor reprezenta datele de intrare pentru rețea, pentru ca rețeaua să prezică masca de separare pentru fereastra de procesare curentă, așa cum este prezentat și în figura 7.

### Masca timp-frecvență

Metoda supervizată propusă de articol, descrisă anterior, țintește estimarea unor măști timp – frecvențe, care pot crește inteligibilitatea separării semnalelor din mix.

Pentru fiecare fereastră de procesare, ieșirea dorită (targetul) este calculat cu ecuația :

În timpul antrenării rețelei, ponderile sunt actualizate prezentând rețelei datele de intrare, adica fereastra de analizare ( care conține fereastra de procesare curentă), împreună cu datele de ieșire dorite (măștile calculate pentru toate ferestrele de procesare). Astfel, se vor capta caracteristicile relevante din setul de antrenare, pentru producerea măștilor corecte pe exemple pe care rețeaua neuronală profundă nu le-a mai văzut.

### Reconstruirea surse lor de semnal

Spectrul complex STFT al surselor de separare și din mixul Y(t, f) sunt obținute prin înmulțirea măștilor prezise de rețea cu ferestrele de procesare ale mixului. După înmulțirea de tip „element – wise” (sau produs Hadamard) se aplică procesul de transformată Fourier doar că de această data este inversă, pentru recuperarea sunetului din mix.

## Deep learning

„Deep learning”-ul (învățarea produndă), cunoscută și ca învățare ierarhică sau învățare profundă structurată, face parte din familia „machine learningului”, care se bazează pe rețele neurale artificiale. Învățarea poate fi nesupervizată, supervizată sau semi-supervizată.

Arhitecturi ale învățării profunde, precum rețelele neurale profunde, rețelele neurale recurente sau rețelele neurale convoluționale au fost aplicate de-a lungul timpului pentru recunoaștere vocală, procesare de limbaj natural, recunoaștere audio, filtrare a rețelelor sociale, traduceri, analiză medicală imagistică, inspecția meterialelor, și așa mai departe. Rețelele neurale au fost inspirate de procesarea informațiilor și comunicarea distribuită ale nodurilor din sistemele biologice. În mod natural, rețelele neurale tind să fie statice și simbolice, în timp ce creierul biologic al diferitelor organisme în viață sunt dinamice și analogice.

„Deep learning”-ul este o clasă de algoritmi ai „machine learningului”, care folosesc mai multe straturi progresive pentru extragerea caracteristicilor de nivel tot mai înalt, din intrări de tip „raw”, adică neprelucrate. De exemplu, în procesarea de imagini, straturile de la început, sau de nivel mai jos, vor extrage margini sau colțuri, pe când straturile de nivel mai înalt vor extrage caracteristici care au o însemnătate pentru om, precum litere, cifre sau fețe.

### Relația dintre dezvoltarea cognitivă și creierul uman

„Deep learning” este în strânsă legătură cu o clasă de teorii a dezvoltării creierului (în special, dezvoltarea neocortexului) propusă de neurologi cognitivi în anii 1990. Aceste teorii ale dezvoltării au fost folosite în modele computaționale, creând astfel predecesorii sistemelor de „deep learning”.

Precum neocortexul, rețelele neurale implică o ierarhie de filtre în straturi în care fiecare strat ia în considerare informația de la un strat anterior, iar mai apoi îl pasează mai departe ca rezultat la celelalte straturi. Acest proces este o stivă care se organizează singură.

O multitudine de moduri au fost folosite pentru investigarea plauzibilității modelelor de învățare profundă dintr-o perspectivă neurobiologică. Pe de o parte, câteva variante de algoritmi de „backpropagation” au fost propuși în așa fel încât să crească procesarea realistă a informației. Anumiți oameni de știință au argumentat că formele de învățare nesupervizată a învățării profunde, ca cele bazate pe modele generative ierarhice și grafice, sunt mai aproape de realitatea biologică.

Deși o comparație sistematică între organizația creierului uman și codificarea neuronală în rețele profunde nu a fost stabilită, câteva analogii au fost raportate. De exemplu, computația realizată de unitățile învățării profunde pot fi similar cu cea a neuronilor și populațiilor de neuroni. În mod similar, reprezentări dezvoltate de modele de învățare profundă sunt similare cu cele măsurate în sistemul vizual primar, atât la nivel de unitate cât și la nivel de populație.

### Cum funcționează o rețea neuronală

În primă instanță, rețelele neurale par un domeniu misterios. În timp ce există o cale intuitivă de a înțelege modelele liniare sau arborii de decizie, rețelele neurale nu au explicații chiar atât de clare. La prima vedere, rețelele neurale au un strat de intrare, care transferă datele către straturile ascunse, iar după puțină magie se poate observa informația dată de stratul de ieșire. În principiu, înțelegerea straturilor ascunse și a ceea ce ele realizează este cheia către implementarea și optimizarea rețelelor neurale.

Rețelele neurale reprezintă unii dintre cei mai puternici și folosiți algoritmi când vine vorba despre subdomeniul numit „deep learning” care aparține de „machine learning”.

Mai jos este prezentat arborele tuturor domeniilor și subdomeniilor ai inteligenței artificiale.

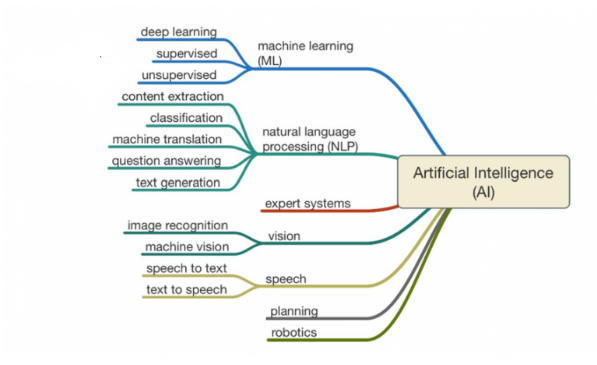


Figura 8 : Arbore structural al inteligenței artificiale și a tuturor subdomeniilor sale

#### Ce este o rețea neuronală?

Pe scurt, o rețea neuronală este un sistem computațional care constă într-un număr de elemente sau noduri simple dar cu un nivel de interconectibilitate foarte ridicat, numiți și neuroni, organizați pe straturi, acestea procesând informația și dând răspunsuri în stări dinamice (care se pot schimba în funcție de datele primite la intrare).

Algoritmul este extrem de folositor în aflarea modelelor caracteristice care sunt prea complexe pentru extragerea lor manuală, pentru ca mai târziu să fie date mașinii să învețe să recunoască diferențele dintre ele.

În contextul acestei structuri prezentate mai sus, șabloanele sau modelele caracteristice sunt introduse în rețeaua neuronală de stratul de intrare și sunt transmise în unul sau mai multe straturi ascunse prezente în rețea; straturile sunt numite „ascunse” doar pentru că nu constituie stratul de intrare sau de ieșire. Totuși, în aceste straturi ascunse, are loc toată procesarea datelor, printr-un sistem de conexiuni, caracterizat de ponderi („weights”) și valori de tip bias („biases”).

Datele de intrare sunt primite, neuronul calculează o sumă ponderată, adăugând și valoarea biasului, și în concordanță cu rezultatul și o funcție de activare ( cea mai comună este sigmoida, chiar dacă nu prea mai este folosită acum și există altele mai bune, precum ReLU, care vor fi prezentate în secțiunea [4.2.6 „Descrierea soluției”](#_Sigmoida_vs_ReLU)) se decide dacă neuronul trebuie să fie „fired” sau activat. După aceea, neronul transmite informația în continuare celorlalți neuroni conectați cu el, într-un proces numit propagare înainte ( „forward pass”). La finalul acestui proces, ultimul strat ascuns este conectat cu stratul de ieșire, care are câte un neuron pentru fiecare posibil rezultat dorit.

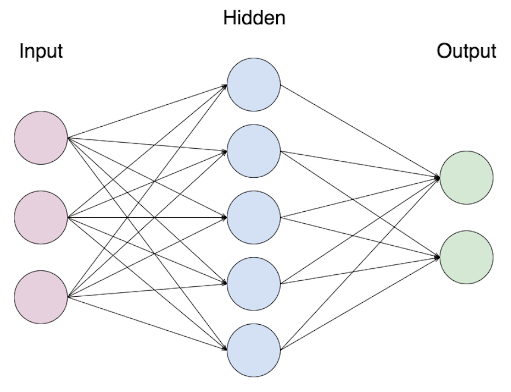


Figura 9 : Structura unei rețele neurale

#### Cum funcționează o rețea neuronală?

Pentru a înțelege cum funcționează rețeaua trebuie înțeles mai întâi cum funcționează diferite tipuri de neuroni. Putem face o paralelă cu perceptronul, chiar dacă folosirea acestuia a scăzut în ultima perioadă, deoarece oferă o idee bine conturată despre cum funcționează neuronii mai moderni.

Un perceptron folosește o funcție pentru a învăța un clasificator binar, mapând un vector de valori binare cu o singură valoare binară de ieșire și poate fi folosit de asemenea în învățarea supervizată. În acest context, perceptronul urmărește următorii pași:

* multiplică toate intrările cu ponderile lor, numere reale care exprimă importanța corespondenței dintre datele de intrare si cele de ieșire
* le adună într-o variabilă numită „sumă ponderată”
* se aplică o funcție de activare, în altă ordine de idei, se determină dacă suma ponderată este mai mare decât o valore prag impusă, unde acest prag este echivalent cu valoarea de bias, și i se asignează valoarea 1 sau 0 ca dată de ieșire

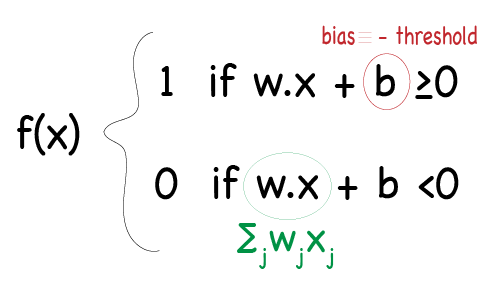


Figura 10 : Funcția perceptronul

Cel mai important lucru din acest algoritm este acela că putem varia ponderile și valoare biasului pentru a obține modele diferite de tipul „decision – making”. Putem asigna mai multă pondere (mai puternică) acestor intrări astfel încât daca ele sunt pozitive, vor favoriza ieșirea dorită de noi. De asemenea, pentru că valoarea de bias poate fi înțeleasă ca o măsură a nivelului de dificultate în aflarea ieșirii L, putem scădea sau crește valoare acestuia pentru a ni se oferi rezultatul dorit ca ieșire.

În consecință, un perceptron poate analiza diferite evidențe sau date și să ia o decizie având în vedere preferințele setului de date. Este posibilă crearea unor rețele mult mai complexe, incluzând mai multe straturi de perceptroni, unde fiecare strat primește ca intrare ieșirea stratului anterior, îl ponderează și reușeste să ia decizii din ce în ce mai complexe.

Unul dintre dezavantajele unei rețele care conține perceptroni este faptul că schimbări minore ale ponderilor sau valorii de bias, chiar dacă este doar într-un singur perceptron, poate schimba radical rezultatul, schimbând din valoarea 1 în 0 și vice versa.

Ce se dorește în domeniu este să se poată face schimbări minore asupra ponderilor sau valorii de bias, iar aici intervin tipurile moderne de neuroni, de exemplu neuronii de tip sigmoidă. Diferența majoră dintre un neuron sigmoidă și un perceptron este faptul că intrarea și ieșirea pot fi orice valoare continuă aflată în intervalul deschis [0,1]. Ieșirea este obținută după aplicarea funcției sigmoide de activare asupra intrărilor, considerând ponderile și valoarea biasului.

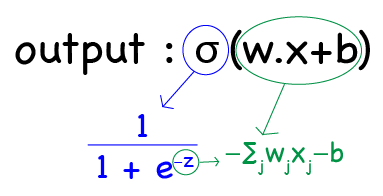


Figura 11 : Formula pentru aflarea valorii de ieșire

#### De ce rețelele neurale sunt capabile să învețe?

Atuul principal al algoritmilor de „machine learning” este abilitatea lor de a învăța și a se îmbunătăți de fiecare dată când prezic o ieșire. În contextul rețelelor neurale, faptul că reușesc să învețe implică faptul că ponderile și valoarea de bias, care definesc conexiunea dintre neuroni, devin tot mai precise, adică ponderile și valoarea biasului sunt selectate în așa fel încât ieșirea din rețea aproximează valoarea reală a y(x) pentru toate intrările de antrenare.

Pașii prin care se trece într-o rețea neuronală sunt următorii:

* alegerea datelor de intrare : se ia un set de antrenare și se setează funcția de activarea „a” care corespunde stratului de intrare
* pasul de „feedforward” : pentru fiecare strat, se calculează funcția z = w \* a + b, unde a = σ(z)
* calculul erorii : se calculează eroarea folosindu-se formula :



Figura 12 : Formula de calcul al erorii

* pasul de „backpropagation” : aici se propagă înapoi eroarea; pentru fiecare strat, se calculează formula

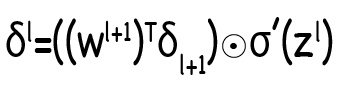


Figura 13 : Formula de calcul pentru propagarea înapoi

* afișarea rezultatelor : se calculează gradientul descent cu respect pentru fiecare pondere și bias folosindu-se formulele :

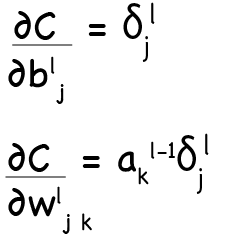


Figura 14 : Formulele de calcul pentru actualizarea ponderilor și a biasului

Desigur, există mult mai multe concepte, implementări și îmbunătățiri care pot fi făcute asupra rețelelor neurale, care sunt din ce în ce mai folosite pentru mai multe domenii și mai puternice decât în anii precedenți.

În [4. „Descrierea soluției”,](#_Descrierea_soluției) se va prezenta structura rețelei neurale folosite pentru implementarea temei și vor fi explicate, pas cu pas, toate conceptele folosite în cadrul dezvoltării temei.

### Rețele neurale profunde (Deep neural networks)

O rețea neuronală profundă (DNN) este o rețea neuronală aritificială cu multiple straturi aflate între straturile de intrare și ieșire. Rețeaua neuronală profundă găsește manipularea corectă matematică pentru a transforma intrarea în ieșire, indiferent dacă este o relație liniară sau non – liniară. Rețeaua se plimbă printre straturi, calculând probabilitatea fiecărei ieșiri. De exemplu, o rețea neuronală profundă care este antrenată să recunoască rasele de câini va primi o imagine, și va calcula probabilitatea ca acel câine din imagine să aparțină unei anumite rase. Utilizatorul poate să vadă rezultatele și să selecteze ce probabilități ar trebui ca rețeaua să arate (mai mare de un anumit prag, etc.) și să returneze o etichetă propusă.

Fiecare manipulare matematică este considerată a fi un strat, iar rețelele neurale profunde complexe au foarte multe straturi, de unde și numele de „profunde”. Scopul este ca, eventual, rețeaua să fie antrenată să descompună datele de intrare în caracteristici, să identifice trendurile care există printre toate mostrele și să clasifice imagini noi/ să ofere datele de ieșire dorite folosindu-se de similaritățile dintre ele, fără să aibă nevoie de ajutor uman.

Rețelele neurale profunde pot modela relații non – liniare complexe. Arhitecturile rețelelor neurale profunde generează modele compoziționale, unde obiectul este expus ca o compoziție stratificată de primitive. Straturile suplimentare permit compunerea de caracteristici din straturile inferioare, care pot modela date complexe cu mai puține unități decât o rețea superficială cu o performanță similară.

Arhitecturile profunde includ multiple variante. Fiecare arhitectură are perfomanțe bune în anumite domenii. Nu este întotdeauna posibilă compararea performanțelor a mai multor arhitecturi , decât atunci când ele au fost evaluate pe același set de date.

Rețelele neurale profunde sunt, de obicei, rețele de tip “feedforward” (cu propagare înainte), în care datele sunt transmise de la stratul de intrare către stratul de ieșire, fără nicio buclă în sens invers. În primă instanță, rețeaua neuronală profundă creează o hartă a neuronilor virtuali și le asignează valori numerice aleatoare, sau așa numitele ponderi, pentru conexiunea dintre neuroni. Ponderile și valorile de intrare sunt multiplicate și returnează o valoare de ieșire în intervalul [0,1]. Dacă rețeaua nu recunoaște un anumit tipar în mod corect, un algoritm ar ajusta ponderile. Astfel, algoritmul poate face anumiți parametrii mai influenți, astfel încât manipularea matematică să fie cea corectă pentru procesarea datelor.

Rețelele neurale recurente, în care datele se pot transfera în orice direcție, sunt folosite pentru aplicații precum modelarea limbajului. Pentru aceasta, foarte folositor este algoritmul “long short – term memory”.

Rețelele neurale profunde convoluționale sunt folosite în “computer vision”. Aceste rețele au fost de asemenea aplicabile pe modelarea acustică pentru recunoaștere vocală.

### Algoritmul de antrenare sau optimizatorul ADAM

Algoritmul Adam de optimizare este o extensie a gradientului descent stohastic. Acesta a început să fie folosit din ce în ce mai des în „deep learning” pentru „computer vision” și procesarea limbajului natural.[[2]](#footnote-2)

Acesta adaugă anumite corecții termenilor calculați pentru a scădea subiectivitatea cu care învață rețeaua neuronală.

Numele vine de la „adaptive moment estimation”[[3]](#footnote-3). Beneficiile folosirii algoritmului Adam de optimizare, prezentate de autori, pentru problemele de optimizare care nu sunt convexe sunt următoarele :

* nu este foarte complicat de implementat
* eficient din punct de vedere computațional
* cererile pentru memorie sunt mici
* invariant la reducerile corecte la scară ale gradienților
* este foarte bun pentru probleme care necesită volum mare de date sau parametrii
* eficient pentru obiective non – staționare
* eficient pentru probleme cu gradienți puțini/rari sau perturbați
* hiper – parametrii au interpretare intuitivă și de obicei au nevoie de foarte puțină atenție din partea utilizatorului[[4]](#footnote-4)

Algoritmul Adam este mai complex față de algoritmul de gradient descent clasic, deoarece algoritmul clasic necesită ajustarea unui singure rate de învățare (alpha). Doar cu ajutorul acestui termen se actualizează ponderile și valoarea ei nu se schimbă niciodată pe parcursul antrenării.

Adam este rezultatul combinării avantajelor a altor doi algoritmi, extensii ale gradientului descent stohastic, și anume:

* AdaGrad („Adaptive Gradient Algorithm”), care are o rată de învățare pentru fiecare parametru, acest lucru îmbunătățind performanțele problemelor cu gradienți puțini/rari (procesarea imaginilor și procesarea limbajului natural)
* RMSProp („Root Mean Square Propagation”), are de asemenea câte o rată de învățare pentru fiecare parametru, dar care se adaptează luând în considerare magnitudinea recentă a gradienților pentru ponderi ( cât de repede se schimbă valoarea acestora), adică algoritmul se va descurca foarte bine pentru problemele nestaționare cu gradienți perturbați

Adam combină eficiențele celor doi algoritmi menționați mai sus. Acesta adaptează rata de învățare folosindu-se atât de media primului moment (ca cel folosit de algoritmul RSMProp), cât și de media pentru al doilea moment al gradienților (varianța necentrată).

Mai specific, algoritmul calculează media exponențială a gradientului și gradientul la pătrat, iar parametrii și sunt folosiți pentru controlul ratelor de scădere ae acestor medii.

și au inițial valori aproape de 1.0, sau cel puțin așa este recomandat să fie.

Parametrii de configurare ai algoritmului sunt următorii :

* alpha, sau rata de învățare (dimensiunea pasului) ; aceasta reprezintă cât de repede sau încet se vor actualiza ponderile rețelei neurale și de obicei are valori mici, de exemplu 10E-3, 10E-4 (pentru o învățare mai înceată) sau 0.3 (pentru o învățare mai rapidă la început, până când algoritmul actualizează valoarea ei)
* , rata de scădere exponențială pentru estimarea primului moment ( cu valoarea 0.9 de exemplu, de cele mai multe ori)
* , rata de scădere exponențială pentru estimarea celui de-al doilea moment (valoarea 0.999 de exemplu, de cele mai multe ori) ; valoarea aceasta ar trebui setată cât mai aproape de 1 pentru problemele cu gradienți puțini/rari
* epsilon, o valoare foarte mică pentru a evita împărțirea la 0 din implementare (de exemplu 10E-8)

Autorii au spus „*Good default settings for the tested machine learning problems are alpha=0.001, beta1=0.9, beta2=0.999 and epsilon=10−8”[[5]](#footnote-5).*

# Descrierea soluției

## Obiective

Obiectivul principal al aplicației este separarea a două surse de semnal, care sunt mixate anterior. Am abordat două metode pentru combaterea „overfitting”-ului.

Prima este separarea unui mix sub forma de 80% pentru setul de antrenare, 10% pentru setul de validare și 10% pentru setul de testare, altfel încât rețeaua, având un context trecut din mixul respectiv, să învețe să separe singură ultimele 20% din ferestre din audio.

A doua este separarea vocilor dintr-un mix, care la rândul lui este suprapus peste o înregistrare cu alt tip de zgomot, precum pianul, aplauzele unei mulțimi etc., prin folosirea permutărilor mixului în setul de antrenare.

Rețeaua va primi un vector (tensor) alcătuit din mai multe ferestre de analizare de tip STFT (Short Time Fourier Transformed). Aceste ferestre conțin date și din ferestrele de procesare anterioare, în acest mod oferindu-se o îmbunătățire considerabilă a performanței în procesarea ferestrelor de dimensiuni mici (exemplu 20/10/5 milisecunde pe fereastră curentă, adică fără contextul din trecut incorporat, iar prin adăugarea contextul din trecut se va obține fereastra de analizare, care va conține de 2 – 4 ori mai multe înregistrări decât cele din fereastra curentă).

Un alt obiectiv este testarea diferitelor valori date rețelei, astfel încât să se găsească rețeaua care se descurca cel mai bine, având acuratețea cea mai mare, fără a face „overfitting”, adică rețeaua să învețe atât de bine setul de antrenare încât să nu știe deloc să prezică valorile pentru setul de validare sau testare.

Pentru testarea rețelei s-au urmărit efectele modificării parametrilor precum marja de eroare între masca prezisă de rețea și masca direct calculată cu formula pentru „soft time frequency masking”, numărul de epoci, rata de învățare, algoritmul de optimizare ( Adam, SGD), conținutul seturilor de antrenare, validare, testare.

În continuare vor fi prezentate modurile în care au fost construite seturile de date, fiecare în parte ce anume conține, cum au fost încărcate acestea și mixate, structura programului și a rețelei, ce se întamplă cu datele când trec prin rețea de la stratul de intrare, până la stratul de ieșire și performanțele rețelei.

## Implementare

### Construirea unui set de date de antrenare/validare/testare

Caracteristicile spectrale corespunzătoare ferestrei de analiză (prezentare mai amănunțită în secțiunea [3.3.2 Propunerea articolului](#_Propunerea_articolului)) sunt generate folsind transformata Fourier (STFT – Short Time Fourier Transform). Se folosește o lungime de fereastră egală cu lungimea ferestrei de procesare și 50% suprapunere (prezentată în figura 15). Cum fereastra de analizare este mai lungă decât fereastra de procesare, se produce un set de caracteristici care sunt mai apoi concatenate pentru a da un vector de caracteristici de analiză mai lung pentru fiecare fereastră de procesare.

Fiecărei ferestre de analizare îi corespunde o mască de separare. Acestă mască de separare este calculată folosindu-se fereastra curentă de procesare din fereastra de analizare.

Setul de date folosit este CMU ARCTIC Database[[6]](#footnote-6), unde se găsesc atât voci de femei, cât și de bărbați. Acestea au fost înregistrate la 16 bit 32 kHz, într-o cameră cu izolare fonică, stereo, iar un singur canal a fost sub formă de undă, iar celălalt este de tip EGG.

Pentru crearea seturilor s-a ținut cont de două abordări, fiecare dintre ele folosind o singură înregistrare audio cu voce de femeie, care va fi mixată cu o singură înregistrare cu voce de bărbat, rezultând astfel un singur mix:

1. Împărțirea 80-10-10
   1. setul de antrenare va conține 80% din întregul mix, se va face împărțirea pe ferestre de procesare și mai apoi pe ferestre de analizare; pe setul de antrenare s-au aplicat diferite permutări ale valorilor pe segmente
   2. setul de validare va conține 10% din mix, după cei 80% alocați setului de antrenare; nu este aplicată nici o permutare aici
   3. setul de testare va conține 10% din mix, după cei 10% alocați setului de validare; asemănător setului de validare, nu se aplică permutări
2. Folosirea mixului întreg, peste care se suprapune un zgomot diferit, de pian, aplauze, etc.
   1. setul de antrenare va conține mixul original și un număr de permutări ale setului
   2. setul de validare va fi format dintr-o permutare diferită de cele folosite în setul de antrenare
   3. setul de testare va fi format din mixul întreg și zgomotul adițional

Înregistrărilor mai scurte de 5 secunde li se adăugau zerouri la final, până când deveneau de 5 secunde, iar cele mai lungi de 5 secunde li se păstrau doar primele 5 secunde (idem și pentru testările când se foloseau 10 secunde).

Toate înregistrările au fost încărcate folosind o rată de 16KHz, folosind biblioteca „LibRosa” din Python. Fiecare înregistrare va avea fix 5 secunde după procesare, și anume 80.000 de valori în vectorul de valori ale înregistrării, respectiv fix 10 secunde după procesare, și anume 160.000 de valori în vectorul de valori ale înregistrării.

Mixul se generează adunând efectiv valorile celor doi vectori ai înregistrărilor.

Fiecare înregistrare pentru mix, vocea 1 și vocea 2, sunt împărțite în ferestre mai mici de câte 20/10/5 milisecunde fiecare, aceasta reprezentând fereastra curentă (PF sau „processing frame”) de acum încolo.

În rețeaua neuronală va intra un tensor, vector, care va conține un număr de ferestre de analizare (AF, „analysis frame”). Această fereastră de analizare este de 2-4 ori mai mare decât fereastra curentă. Ea este formată din fereastra curentă și 2/3 ferestrele precendente. Pentru generarea ferestrei de analizare se folosește suprapunere de 50%. Acest proces este prezentat în Figura 14.

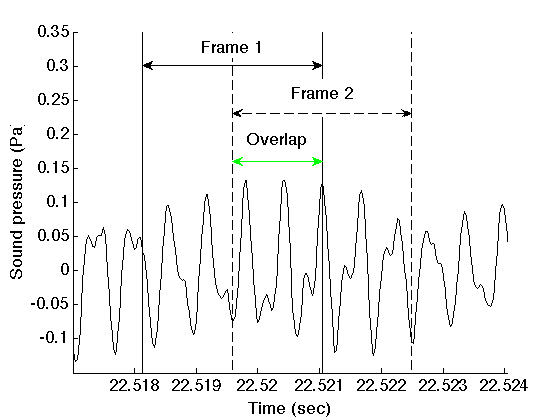


Figura 15 : Procesul de suprapunere a ferestrelor

După ce s-a creat ferestra de analizare, folosindu-se procesul de suprapunere, se aplică transformata Fourier pe termen scurt asupra AF-ului („analysis frame”-ului), implementată în biblioteca LibRosa. Ca parametrii, funcția pentru STFT ( short time Fourier transform) primește fereastra pentru care se face transformarea, se alege metoda de „windowing” (în cazul nostru se folosește fereastra Hanning), dimensiunea ferestrei pentru transformata Fourier ( 1024 este valoarea folosită în aplicație), și valoarea săriturii, adică numărul de coloane peste care sare pentru a trece la următoarea fereastră ( setat cu valoarea 8).

După calculul transformatei Fourier pe termen scurt, se calculează masca ferestrei curente și se salvează în lista de targeturi. Masca este calculată în felul următor:

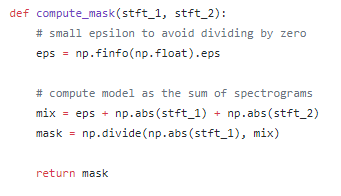


Figura 16 : Modul de calcul al măștii de frecvență pentru fereastra de procesare curentă

### Încărcarea semnalelor și mixarea lor

Semnalele sunt încărcate folosind biblioteca Python numită LibRosa, în felul următor :

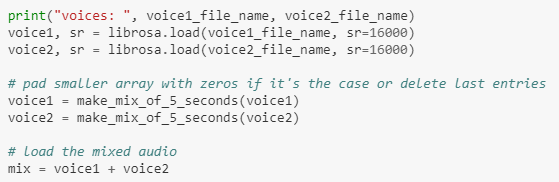


Figura 17 : Încărcarea semnalelor și mixarea lor

Fiecare înregistrare audio este încărcată folosind funcția „load”, căreia trebuie să îi dăm ca parametrii numele fișierului pe care dorim să îl încărcăm și rata de eșantionare a fișierului.

Mai departe, am ales ca fișierul audio care va fi prelucrat mai departe și dat ca dată de intrare pentru rețeaua neuronală profundă să aibă dimensiunea fixă de 5 secunde (sau 10 secunde). S-a ajuns la această decizie din cauza faptului că pentru un mix de 5 secunde, se vor genera 498 de ferestre de analizare, sau există în acel mix 498 de ferestre de procesare, ceea ce este mult pentru o mașină cu putere de calcul mediocră în momentul când apar și permutările mixului original.

Valoarea de 5 secunde s-a dovedit a fi un număr bun pentru a avea un mix de dimensiune cât mai mică dar și pentru a se înțelege propoziția spusă în fișierele audio. Astfel, dacă fișierul audio are mai mult de 5 secunde, acesta va fi tăiat astfel încât să fie de 5 secunde, iar dacă fișierul audio este mai mic de 5 secunde, i se va adăuga zerouri până când vectorul reprezentativ fișierului să fie de dimensiunea dorită.

Mixul celor două voci se va obține prin simpla adunare a celor doi vectori, aferenți celor două voci.

Mai departe, putem observa spectrogramele celor două voci și a mixului format din ele :

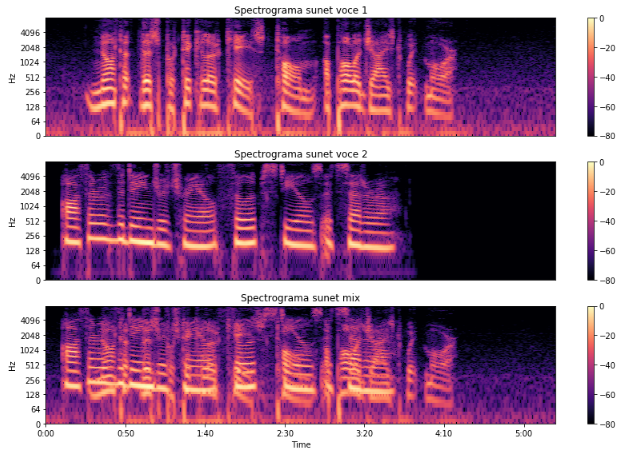


Figura 18 : Spectrogramele calculate pentru cele două surse de semnal și mixul format din ele

În figura precendentă se observă o reprezentare a spectrogramelor celor două voci și a mixului format din cele două voci.

După cum se poate observa, pe axa OX avem timpul, cum am precizat și mai sus, se ajunge pană la 5 secunde pe fiecare înregistrare audio.

Pe axa OY avem frecvențele, cât sunt de înalte sau joase în funcție de vocea respectivă.

Mai avem o a treia dimensiune, cea reprezentată in dreapta fiecărui grafic, și anume intensitatea amplitudinii fiecărui sunet. Acesta este reprezentat pe o scară de la -80 la 0. Valoarea de 0 este pragul auditiv al omului, tot ce este sub, nu poate fi auzit de noi. Tot ce este peste 110, deja este foarte zgomotos, iar la 140 se ajunge la pragul dureros. În cazul nostru nu este nevoie, deoarece mai sus sunt folosite înregistrări din camere antifonate.

### Structura rețelelor neurale

#### Rețeaua formată doar din straturi dens conectate

Rețeaua neuronală are trei straturi de tip „fully connected” ( dens conectate), iar după fiecare strat dens conectat se face o normalizare a datelor.

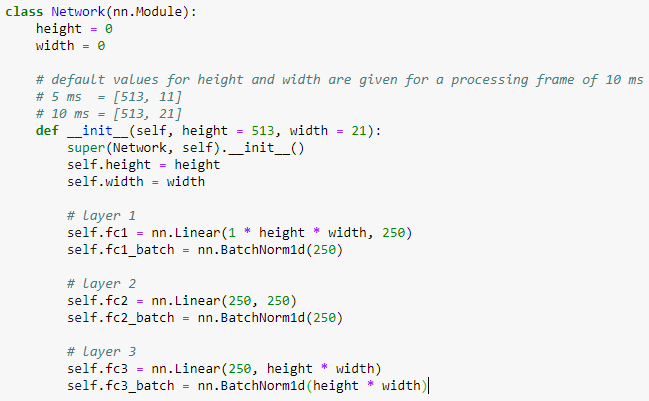


Figura 19 : Structura rețelei neurale dens conectate pe straturi

Valorile standard pentru numărul de linii și coloane sunt 513, respectiv 21, pentru o fereastra de procesare curentă de 10 milisecunde și o fereastră de analiză de 4 ori mai mare decât fereastra de procesare.

#### Rețeaua formtă din straturi convoluționale și dens conectate

Rețeaua neuronală are două straturi convoluționale și două straturi de tip „fully connected” ( dens conectate), iar după fiecare strat se face o normalizare a datelor.

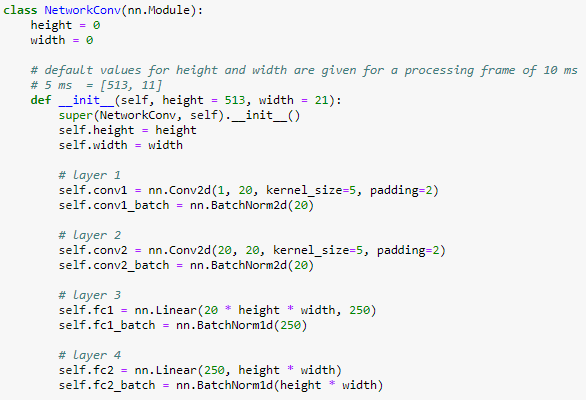


Figura 20 : Structura rețelei neurale cu straturi convoluționale și dens conectate

Valorile standard pentru numărul de linii și coloane sunt 513, respectiv 21, asemănător structurii doar cu straturi dens conectate.

#### Straturile de tip fully connected (dens conectate)

O rețea neuronală profundă dens conectată constă într-o serie de straturi dens conectate. Aceste straturi dens conectate sunt reprezentate printr-o funcție de la ℝ (m) la ℝ (n). Fiecare dimensiune de ieșire depinde de dimensiunea intrării. Un strat dens conectat este reprezentat astfe:

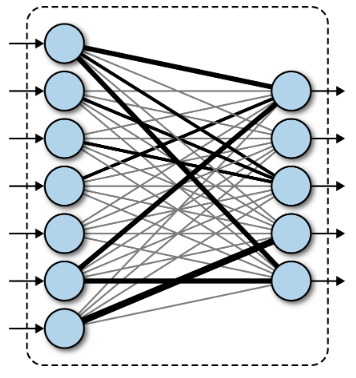


Figura 21 : Structrura unui strat dens conectat

Fie x ∈ ℝ cel care reprezintă intrarea pentru un strat dens conectat, iar y ∈ ℝ să fie a i-a ieșire din stratul dens conectat. Atunci ∈ ℝ este calculat astfel :

Aici, σ este o funcție neliniară ( sigmoida, care va fi prezentată mai târziu), iar sunt parametrii care învață din rețea. Ieșirea y este calculată astfel :

Este posibil să se stivuiască rețele dens conectate. O rețea cu multiple rețele dens conectate este de obicei numită „rețea profundă” și arată ca în Figura 20.

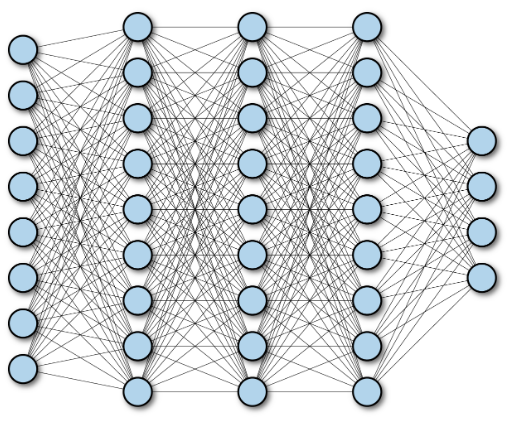


Figura 22 : Rețea neuronală profundă cu rețele dens conectate

Nodurile din rețelele dens conectate sunt numite de obicei neuroni. În consecință, oriunde în literatura de specialitate, rețelele dens conectate vor fi referite des ca fiind rețele neurale.

În acest tip de strat, fiecare neuron din stratul curent este conectat cu fiecare neuron din stratul anterior și fiecare conexiune are propria pondere.

Aceste straturi dens conectate sunt cele care fac învățarea discriminantă într-o rețea neuronală profundă.

#### Straturile convoluționale

Straturile convoluționale vor primi intotdeauna un tensor cu dimensiunea (număr imagini) x (lățime imagine) x (înalțime imagine) x (adâncime imagine).

Rețelele neurale profunde formate doar din straturi dens conectate pot și ele să extragă caracteristici din input și să clasifice, dar pe imagini nu este întocmai practică această arhitectură. Straturile convoluționale sunt în principiu folosite pentru extragerea de trăsături din input.

Parametrii stratului sunt un set de filtre care se adaptează și învață din fiecare input. În timpul propagării înainte, fiecare filtru se „plimbă” de-a lungul imaginii/inputului pe lățime și înălțime. Ca rezultat, rețeaua învață filtre care se activează când acestea detectează anumite tipuri de caracteristici la o anumită poziție din datele de intrare.

#### Batch normalization

Normalizarea este o metodă foarte cunoscută în antrenarea rețelelor neurale profunde. Aceasta a fost introdusă de Sergey Ioffe și Christian Szegedy de la Google. Metoda se referă la normalizarea valorilor de activare ale unităților ascunse, astfel încât distrubuția acestor valori de activare să rămână aceleași pe timpul antrenării.

În timpul antrenării oricărei rețele neurale profunde, dacă activarea ascunsă a distribuției se schimbă din cauza schimbărilor ponderilor și biasului în acel strat, ele pot cauza foarte repede schimbări în stratul următor, iar acest lucru încetinește foarte mult antrenarea.

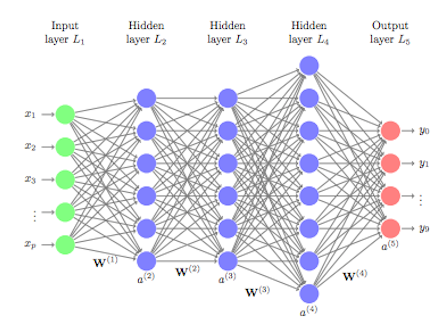


Figura 23 : Structura unei rețele neurale profunde cu 3 straturi ascunse

În figura 21 avem o rețea neuronală profundă cu 3 straturi ascunse, pe lângă care un strat de intrare și încă unul de ieșire. Fiecare strat ascuns are propriile matrici de ponderi și vectori de bias. Intrarea la fiecare nivel trece prin transformări folosind matricea de ponderi și vectorul de bias. De exemplu, ieșirea din stratul ascuns L2 se comportă ca intrare pentru stratul ascuns L3. Valorile de activare ascunse ale stratului 2 (L2) sunt transformate înmulțind matricea de ponderi și adăungând valorile din vectorul de bias. Această ieșire este trecută mai departe printr-o funcție de activare precum sigmoida, reLU sau tanh, iar rezultatul pentru stratul ascuns 3 (L3) este obținut.

Acest proces se repetă la fiecare strat ascuns. Așa cum am observat, valorile de activare ale stratului 3 sunt de fapt afectate, influențate de valorile de activare ale stratului 2. Dacă distribuția valorilor de activare ale stratului 2 se schimbă rapid, atunci acest lucru va afecta eficiența în antrenarea rețelei neurale profunde.

Mai departe se va descrie cum se realizează această normalizare asupra unui set de date. Normalizarea oricărei date se referă la aflarea mediei și varianței datelor și normalizarea lor, astfel încât datele să aibă 0 media și varianță unitară.

Considerăm că avem „d” unități ascunse într-un strat ascuns în rețeaua neuronală ascunsă. Se pot reprezenta valorile de activare ale acestui strat ca fiind . O unitate din aceast vector de valori de activare se poate normaliza folosind următoarea formulă :

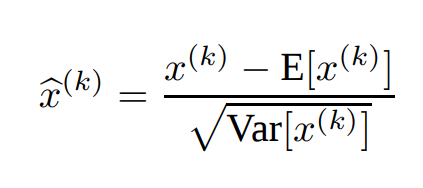


Figura 24 : Formula de normalizare a unei unități din stratul ascuns „d”

este media valorilor,iar este varianța unității k din stratul ascuns. După normalizare, fiecare unitate ascunsă va avea după media 0 și varianță unitară, dar de obicei nu se dorește media 0 și varianța unitară. În schimb, se dorește ca rețeaua să învețe și să adapteze valorile mediei și varianței. Pentru acest lucru, se introduc 2 variabile noi, una pentru învățarea mediei și alta pentru varianță. Acești doi parametrii sunt învățați și actualizați împreună cu ponderile și biasul, în timpul antrenării. Versiunea finală normalizată, scalată și orientate a valorilor ascunse de activare pentru a k-a unitate ascunsă este dată de formula următoare :

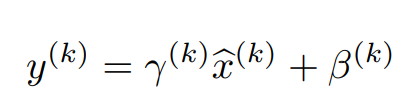


Figura 25 : Versiunea finală a valorii de activare a unei unități după normalizare

De obicei, când antrenăm orice rețea neuronală profundă, nu se dau toate datele deodată pentru că ar crește foarte mult complexitatea calculului. În schimb, rețelele neurale se antrenează cu tehnici stohastice de optimizare, unde mici seturi de date este dată din întregul set de date, iar parametrii rețelei sunt actualizați luând în considerare valorile de eroare ale acelui set mai mic de date.

### Workflow-ul rețelei neurale

Cum este descris și în [figura 7](#_Propunerea_articolului), în rețeaua neuronală profundă se urmează următorii pași, pe parcursul unei epoci:

1. se creează seturile de antrenare, validare și testare (vezi secțiunea [4.2.1 Construirea unui set de date de antrenare/validare/testare](#_Construirea_unui_set))
2. se face pasul de propagare înainte (prezentat în secțiunea „[4.2.7.1 Pasul de propagare înainte”](#_Pasul_de_propagare) ):
   1. datele de intrare intră în rețeaua neuronală profundă, iar matricea 2D de dimensiune n x m este transformată într-un vector cu n x m elemente, 1D
   2. vectorul n x m intră în primul strat de tip dens conectat, care va conține 250 de neuroni, astfel vectorul care pleacă din acest strat are 250 de elemente
   3. valorile din vector sunt normalizate cu batch normalization, dimensiunea vectorului se păstrează tot la 250 de elemente
   4. se aplică funcția de activare, și anume sigmoida sau ReLU (ambele au fost folosite în testarea și găsirea structurii rețelei care este cea mai eficientă pentru obiectivul urmărit)
   5. pașii b,c, și d se repetă și pentru stratul al doilea dens conectat
   6. pașii b,c, și d se repetă și pentru stratul al doilea dens conectat
   7. la finalul stratului al trilea, rezultatul prezis de rețea este tranformat din vectorul de 250 de elemente într-o matrice 2D de dimensiune n x m, cum era inițial
3. se calculează funcția de cost sau eroarea dată de diferența dintre ce a prezis rețeaua și care este răspunsul target dorit (se folosește MSELoss)
4. se face pasul de propagare înapoi (implementat în biblioteca din pyTorch, și prezentat în secțiunea „[4.2.7.2. Pasul de propagare înapoi](#_Pasul_de_propagare_1)”)
5. se actualizează ponderile
6. se salvează valorile erorilor dintre preziceri și valorile target pentru a se putea vedea cum și cât scade eroare, sau în cazurile mai puțin fericite, crește

Pentru rețeaua care conține două straturi convoluționale și două straturi dens conectate pașii de trecere prin rețea sunt similari, diferă doar tipul stratului și numărul total de straturi.

### Funcția de cost (MSE)

În aplicație este folosită ca funcție de cost media erorii pătratice (Mean Square Error, MSE) din biblioteca PyTroch și este salvată într-o funcție proprie :



Figura 26 : Funcția de cost MSE

Ecuația în python pentru această funcție este de fapt :

,

iar funcția matematică este :

, unde :

* C = funcția de cost
* w = colecția tuturor valorilor de ponderi
* b = vectorul de bias
* n = numărul total de date de intrare doar pentru antrenare pentru rețea
* y(x) = valorile de tip target sau rezultatul calculat apriori și corect
* x = data de intrare x
* a = valoare prezisă de rețea ca rezultat pentru data de intrare x

Funcția mediei erorii pătratice măsoară valoarea medie în care valorile predicțiilor modelului variază, deci se poate vedea ca o măsurare a performanței modelului pe setul de antrenare.

Costul este mai mare când modelul nu are performanțe prea bune pe setul de antrenare, iar obiectivul algoritmului de învățare este să găsească acele preziceri, sau mai degraba ponderile cu ajutorul cărora modelul face prezicerile, astfel încât costul să fie minimum.

### Sigmoida vs ReLU

Forma matematică a sigmoidei este :

Aceasta are valorile cuprinse în intervalul deschis [0,1], și este reprezentată astfel :

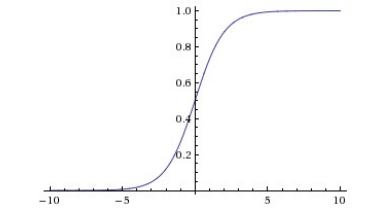


Figura 27 : Reprezentarea funcției sigmoide

Funcția sigmoidă preia un număr real și îl scalează în intervalul [0,1]. Dacă există cazuri în care avem numere negative foarte mari, acelea sunt înlocuite cu 0, identic si pentru numerele pozitive foarte mari, care devin 1.

Sigmoida a pierdut teren în cadrul funcțiilor de activare din cauza a două deazvantaje:

* se ajunge la distrugerea gradientului, adică neuronul ajunge la saturație; dacă de exemplu ponderile inițiale sunt foarte mari, neuronii se vor satura, iar rețeaua nu va mai fi capabilă să învețe
* valorile de ieșire ale funcției nu sunt centrate în 0 (spre deosebire de ReLU), astfel gradientul poate deveni fie complet pozitiv, fie complet negativ, iar acest lucru are complicații asupra dinamicii dintre neuroni.

Funcția de activare ReLU are formula :

, unde a = Wx + b

ReLU returnează ca ieșire valoarea x, dacă x este număr pozitiv, sau 0 dacă este negativ, așadar are valori în intervalul [0, ∞) . Reprezentarea funcției ReLU arată astfel :

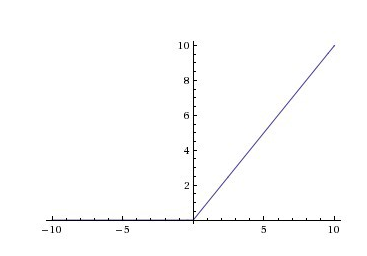


Figura 28 : Reprezentarea funcției ReLU

ReLU este cea mai folosită funcție de activare din lume, la acest moment, deoarece este folosită în aproape toate rețelele neurale convoluționale sau în deep learning.

Pentru a vedea diferențele de reprezentare și evoluție dintre cele două funcții de activare avem figura următoare:

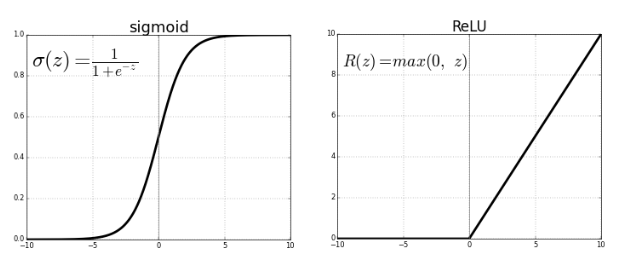


Figura 29 : Reprezentarea sigmoidă vs ReLU

Când se cunoaște faptul că funcția care se dorește a se aproxima are anumite caracteristici, se poate alege funcția de activare care va ajuta antrenarea cât mai bună și rapidă a rețelei. De exemplu, sigmoida funcționează foarte bine pentru probleme de clasificare, deoarece aproximarea unei funcții de clasificare ca o combinație de sigmoide este mai ușor, iar astfel se va ajunge mult mai repede la convergență.

Funcția ReLU are atât avantaje, cât și dezavantaje:

* avantaj : oferă o convergență mai bună a gradientului, are formă liniară care nu se saturează (spre deosebire de sigmoidă)
* avantaj : se implementează prin prăguirea matricii de activare în 0, spre deosebire de sigmoidă sau tangenta hiperbolică, a căror neuroni presupun operații complexe
* dezavantaj : neuronii ReLU sunt „fragili” și pot deveni inactivi/muri foarte repede; dacă într-un neuron intră o valoare foarte mare, acesta va propaga către restul unităților ponderile într-un mod în care el însuși nu se va mai putea activa vreodată. Acest lucru se poate întâmpla dacă rata de învățare este prea mare

### Antrenarea rețelei

Pentru o pereche de instruire se face pasul de propagare înainte, se obţine vectorul de ieşire, iar pentru fiecare strat de neuroni l, începând de la ultimul, se calculează un termen de eroare folosindu-se integralele parțiale în pasul de propagare înapoi.

#### Pasul de propagare înainte (feedforward)

În pasul de propagare înainte datele de intrare trec prin rețeaua neuronală și la final rețeaua prezice o ieșire în funcție de data de intrare primită.

Primul strat nu are rol computațional, iar valoarea de ieșire este întocmai vectorul/tensorul de intrare.

Ultimul strat are rolul de prezicere, adică valorile ieșirilor din ultimul strat sunt folosite pentru a prezice masca aferentă ferestrei de analizare dată ca intrare.

„Propagarea înainte preia un vector și modifică starea neuronilor din fiecare strat.”[[7]](#footnote-7)

Valorile de intrare din primul strat, intră în stratul dens conectat sau convoluțional, se calculează ponderile și biasul, se normalizează datele, se aplică funcția de activare, în cazul proiectului funcția ReLU și se trece la următorul strat.

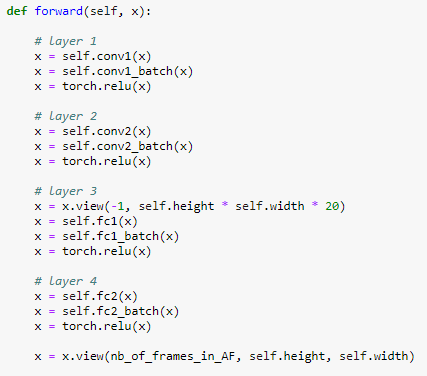


Figura 30 : Pasul de propagare înainte

#### Pasul de propagare înapoi (backpropagation)

În pasul de propagare înapoi trebuie propagată eroarea din funcția de cost (calculată făcându-se diferența dintre ieșirea prezisă de rețea și valorea măștii corecte, calculată apriori) înapoi către fiecare strat. În această propagare înapoi a erorii către fiecare strat trebuie actualizate valorile ponderilor în concordanță cu mesajul erorii.

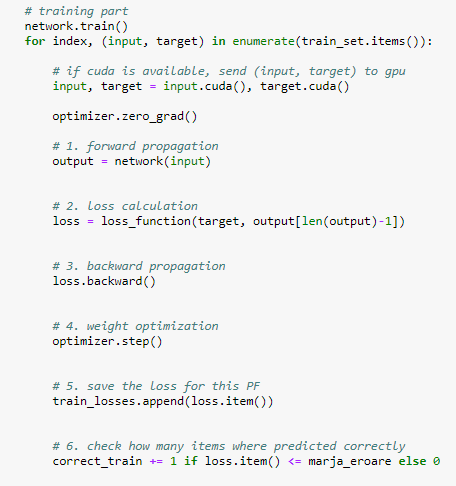


Figura 31 : Pasul de propagare înapoi

### Testarea rețelei

Pentru a nu ajunge în situația de „overfitting”, trebuia să se ajusteze numărul de epoci și alți parametrii astfel încât curba erorilor pentru test să nu înceapă să urce după ce a coborât împreună cu cea de antrenare, ca în figura de mai jos:

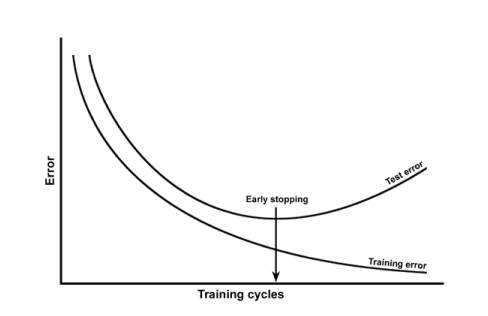


Figura 32 : Curbele de învățare pentru antrenare și testare

În aplicație, pasul de testare al rețelei arată în felul următor :

* pentru fiecare fereastră de analiză, se trimit datele către rețea, iar rețeaua returnează o mască
* se calculează eroarea pentru a putea vedea cât de mare este discrepanța dintre prezicere si masca inițială calculată corect
* masca se salvează într-o matrice pentru a putea fi aplicată asupra întregului mix la final
* dacă eroare este sub o marjă de eroare aleasă, atunci se acceptă ca fiind corectă prezicerea
* se calculează acuratețea pentru setul de testare



Figura 33 : Pasul de evaluare pe setul de testare

## Rezultate și performanțele rețelei

### Performanțe în funcție de funcția de activare

S-au urmărit performanțele pentru două funcții de activare, și anume funcția sigmoidă și reLU. Deoarece sigmoida are domeniul de definire în intervalul deschis [0,1], la un moment dat, ea tindea să se plafoneze și acuratețea să crească ori foarte greu, ori mai deloc.

În schimb, reLU are domeniul de definire [0, ∞], oferă o convergență mai bună a gradientului și are formă liniară care nu se saturează spre deosebire de sigmoidă.

### Performanțe în funcție de structura rețelei neurale

### Performanțe în funcție de împărțirea setului de date

S-au urmărit două metode diferite, și anume:

* modelul 80-10-10 pe un singur mix, unde rețeaua se concentrează pe separarea surselor de semnal pe ultima bucată de 10% din mix
  + rețeaua primea ca set de antrenare primele 80% valori din întregul mix, se creau ferestrele de analizare, iar pentru validare și testare rețeua prezicea măștile pentru ultimele 20% ferestre
  + abordarea nu a fost una reușită deloc, validarea și testarea pe cele două bucăți de câte 10% avea procentajul de acuratețe de 0%
* modelul în care se dădea mixul întreg ca dată de intrare, iar la testare rețeaua trebuia să știe să separe vocile, indiferent de ce alt zgomot se adaugă pe fundal, precum un pian, aplauze etc.
  + ca set de antrenare se dădea mixul original și un număr de permutări ale setului, astfel încât rețeaua să nu ajungă în „overfitting”
  + setul de validare consta într-o permutare a mixului original, diferită de permutările create pentru setul de antrenare
  + setul de testare este format din mixul original și un sunet de fundal adăugat
  + aici s-au obținut procentaje între 25-35%

# Concluzii

În concluzie, s-a atins un prag maximum de TODO % acuratețe, folosindu-se metoda în care rețeaua primea mixul întreg în diferite combinații, și anume setul de antrenare format din mixul original și un număr de permutări ale vocilor și mixului, setul de validare format dintr-o permutare diferită de cele din setul de antrenare, iar setul de testare format din mixul original suprapus peste un zgomot de fundal, precum sunet de pian, lătrat de câine, zgomot de mașini, etc.

Ca dorință de viitor, rămasă neimplementată, este folosirea unui număr mare de mixuri întregi pentru setul de antrenare, aproximativ 1000 mixuri, și câte 100 mixuri atât pentru setul de validare, cât și pentru testare. Rezultatul din această abordare ar fi faptul că rețeaua ar ști să prezică măștile de separare a vocilor celor două persoane, femeia și bărbatul, din unele mixuri și înregistrări pentru fiecare persoană pe care nu le-a mai văzut în antrenare.

# Index figuri

[Figura 1 : Pașii aplicării transformatei Fourier pe un semnal 11](#_Toc11508529)

[Figura 2 : Fereastră care se plimbă peste un semnal 12](#_Toc11508530)

[Figura 3 : Datele unui semnal după aplicarea funcției "half-cosine" 13](#_Toc11508531)

[Figura 4 : Spectrograme folosite în aplicație pentru două voci și mixul lor 15](#_Toc11508532)

[Figura 5 : Calcularea măștii de frecvență cu respect pentru prima sursă de semnal 16](file:///C:\Users\Kate\Desktop\disertatie\sound-source-separation\Disertatie-doc.docx#_Toc11508533)

[Figura 6 : Calcularea măștii de frecvență cu respect pentru ambele surse de semnal 17](file:///C:\Users\Kate\Desktop\disertatie\sound-source-separation\Disertatie-doc.docx#_Toc11508534)

[Figura 7 : Ilustrare schematică a rețelei neurale profunde propuse 20](#_Toc11508535)

[Figura 8 : Arbore structural al inteligenței artificiale și a tuturor subdomeniilor sale 23](#_Toc11508536)

[Figura 9 : Structura unei rețele neurale 24](#_Toc11508537)

[Figura 10 : Funcția perceptronul 25](#_Toc11508538)

[Figura 11 : Formula pentru aflarea valorii de ieșire 26](#_Toc11508539)

[Figura 12 : Formula de calcul al erorii 27](#_Toc11508540)

[Figura 13 : Formula de calcul pentru propagarea înapoi 27](#_Toc11508541)

[Figura 14 : Formulele de calcul pentru actualizarea ponderilor și a biasului 27](#_Toc11508542)

[Figura 15 : Procesul de suprapunere a ferestrelor 33](#_Toc11508543)

[Figura 16 : Modul de calcul al măștii de frecvență pentru fereastra de procesare curentă 34](#_Toc11508544)

[Figura 17 : Încărcarea semnalelor și mixarea lor 34](#_Toc11508545)

[Figura 18 : Spectrogramele calculate pentru cele două surse de semnal și mixul format din ele 35](#_Toc11508546)

[Figura 19 : Structura rețelei neurale dens conectate pe straturi 36](#_Toc11508547)

[Figura 20 : Structura rețelei neurale cu straturi convoluționale și dens conectate 37](#_Toc11508548)

[Figura 21 : Structrura unui strat dens conectat 38](#_Toc11508549)

[Figura 22 : Rețea neuronală profundă cu rețele dens conectate 39](#_Toc11508550)

[Figura 23 : Structura unei rețele neurale profunde cu 3 straturi ascunse 41](#_Toc11508551)

[Figura 24 : Formula de normalizare a unei unități din stratul ascuns „d” 42](#_Toc11508552)

[Figura 25 : Versiunea finală a valorii de activare a unei unități după normalizare 42](#_Toc11508553)

[Figura 26 : Funcția de cost MSE 44](#_Toc11508554)

[Figura 27 : Reprezentarea funcției sigmoide 45](#_Toc11508555)

[Figura 28 : Reprezentarea funcției ReLU 46](#_Toc11508556)

[Figura 29 : Reprezentarea sigmoidă vs ReLU 47](#_Toc11508557)

[Figura 30 : Pasul de propagare înainte 48](#_Toc11508558)

[Figura 31 : Pasul de propagare înapoi 49](#_Toc11508559)

[Figura 32 : Curbele de învățare pentru antrenare și testare 50](#_Toc11508560)

[Figura 33 : Pasul de evaluare pe setul de testare 51](#_Toc11508561)

# Bibliografie și webografie

* „Blind Audio Source Separation Pipeline and Algorithm Evaluation” - Wisam Reid, Kai-Chieh Huang & Doron Roberts-Kedes

(<http://cs229.stanford.edu/proj2015/124_report.pdf>)

* <https://musicinformationretrieval.com/stft.html>
* https://en.wikipedia.org
* [https://librosa.github.io/librosa](https://librosa.github.io/librosa/)
* https://docs.scipy.org/doc/numpy
* “Time frequency masking”, Machine Perception of Music & Audio - Zafar Rafii, 2014 (<http://users.cs.northwestern.edu/~pardo/courses/eecs352/lectures/MPM14-Time-Frequency-Masking.pdf>)
* “Inteligență artificială” – Lucian M. Sasu
* <https://towardsdatascience.com/understanding-neural-networks-what-how-and-why-18ec703ebd31>
* https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning
* A Method for Stochastic Optimization”, Diederik Kingma și Jimmy Ba
* https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning, Jason Brownlee, 2017
* <https://www.deeplearning.ai/deep-learning-specializatio>
* <http://www.numerical-tours.com/matlab/audio_1_processing>

1. [https://ro.wikipedia.org/wiki/Spectrogram%C4%83](https://ro.wikipedia.org/wiki/Spectrogramă) [↑](#footnote-ref-1)
2. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning, Jason Brownlee, 2017 [↑](#footnote-ref-2)
3. “Adam: A Method for Stochastic Optimization”, Diederik Kingma și Jimmy Ba [↑](#footnote-ref-3)
4. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning, Jason Brownlee, 2017 [↑](#footnote-ref-4)
5. “Adam: A Method for Stochastic Optimization”, Diederik Kingma și Jimmy Ba [↑](#footnote-ref-5)
6. Setul de date poate fi găsit la : <http://www.festvox.org/cmu_arctic/> [↑](#footnote-ref-6)
7. “Inteligență artificială” – Lucian M. Sasu , versiunea iunie 2019 [↑](#footnote-ref-7)