UNIVERSITATEA POLITEHNICA BUCUREȘTI
FACULTATEA DE ELECTRONICĂ, TELECOMUNICAȚII ȘI TEHNOLOGIA INFORMAȚIEI

RESTAURARE DE DIACRITICE UTILIZÂND TEHNICI BAZATE PE REȚELE NEURALE

LUCRARE DE DIPLOMĂ

Prezentată ca cerință parțială pentru obținerea titlului de *Inginer*
în domeniul *Calculatoare și Tehnologia Informației*
programul de studii *Ingineria Informației*

Profesor coordonator:
Ș.L. Dr. Ing. Horia CUCU

Student:
Florin-Nicolae IORDACHE

București
2019

Universitatea "Politehnica" din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Departamentul **EAII**

Anexa 1

TEMA PROIECTULUI DE DIPLOMĂ
a studentului **IORDACHE C.L. Florin-Nicolae , 444A**

1. Titlul temei: Restaurare de diacritice utilizând tehnici bazate pe rețele neurale

2. Descrierea contribuţiei originale a studentului (în afara părţii de documentare) şi specificaţii de proiectare:
Proiectul urmărește realizarea unui studiu comparativ al algoritmilor de inteligență artificială bazați pe rețele neurale ce pot fi folosiți în restaurarea diacriticelor unui text scris în limba română. Contribuția personală este dată de cercetarea și implementarea mai multor algoritmi asupra cărora se va urmări efectul modificării parametrilor utilizați. Printre parametrii urmăriți se numără arhitectura rețelei, algoritmul de optimizare, algoritmul de regularizare, rata de învățare și modalitatea de preprocesare a setului de date de antrenare. Se vor rula algoritmii de antrenare și de testare, iar în urma lor se vor nota rezultate precum eroarea sau acuratețea și timpul necesar antrenării. Scopul este acela de a analiza performanțele metodelor ce folosesc rețele neruale față de metodele clasice de restaurat diacritice, ce utilizează modele statistice.

3. Resurse folosite la dezvoltarea proiectului:
Limbaj de programare Python, cu librării precum Keras, Tensorflow, NumPy și Matplotlib ce facilitează implementarea algoritmilor. Setul de date de antrenare și testare este oferit de către laboratorul SpeeD.

4. Proiectul se bazează pe cunoştinţe dobîndite în principal la următoarele 3-4 discipline:
PC, POO, DEPI, RFIA

5. Proprietatea intelectuală asupra proiectului aparţine: U.P.B.

6. Data înregistrării temei: 2018-11-28 18:40:49

Conducător(i) lucrare,
Conf. dr. ing. Horia COCU
semnătura: .....

Student,
semnătura:.....

Director departament,
Prof. dr. ing Sever PAȘCA
semnătura:.....

Decan,
Prof. dr. ing. Cristian NEGRESCU
semnătura:.....

Cod Validare: **5a674f90f7**

Declarație de onestitate academică

Prin prezenta declar că lucrarea cu titlul "*Restaurare de diacritice utilizând tehnici bazate pe rețele neurale*", 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 *Calculatoare și Tehnologia Informației*, programul de studii *Ingineria Informației* 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, *25.06.2019*

Florin-Nicolae IORDACHE

CUPRINS

LISTĂ DE FIGURI

LISTĂ DE TABELE

LISTĂ DE ACRONIME

API – Application Programming Interface

BPE – Batches per Epoch

BiLSTM – Bidirectional Long Short-Term Memory

CUDA - Compute Unified Device Architecture

ChER – Character Error Rate

ESP – Early Stopping Patience

GAN – Gradient Accelerat Nesterov

GRU - Gated Recurrent Unit

JSON – Javascript Object Notation

LSTM – Long Short-Term Memory

RLRP – Reduce learning rate patience

RNR – Rețea neurală recurentă

RSS - Really Simple Syndication

SGD – Stochastic Gradient Descent

SSH – Secure Shell

TPE – Tree Parzen Estimator

WER – Word Error Rate

CAPITOLUL 1

INTRODUCERE

1.1 MOTIVAȚIE

Diacriticele reprezintă semne adăugate literelor pentru a da un anumit sens cuvintelor pe care le formează. Aceste semne sunt de obicei adăugate dedesubtul sau deasupra literelor. Limba română folosește 5 litere cu diacritice: ă, â, î, ș, ț. Chiar dacă sunt puține astfel de litere, există totuși o multitudine de cuvinte care le folosesc, aproximativ 40% din cuvintele vocabularului, și multe cuvinte care au aceeași formă, dar au sens diferit, în funcție de diacriticele utilizate, de exemplu *fata*, *fată*, *fața*, *față*.

Neavând la dispoziție forma corectă a cuvântului, cititorul poate doar să presupună ceea ce ar trebui să însemne textul. Scrierea fără diacritice reprezintă ambiguități care pot duce la interpretări greșite. De exemplu, propoziția “am vazut o gasca” are diferite sensuri deoarece ultimul cuvânt poate avea două înțelesuri. Fie cuvântul este de fapt gașcă, adică un grup de oameni, fie este gâscă, o pasăre. De obicei, când vine vorba de comunicarea informală, cum sunt mesajele private între prieteni, aceasta nu este o problemă majoră, chiar dacă uneori apar confuzii, precum cea anterioară. În schimb, în cadrul unor texte precum cele juridice, comunicate oficiale, chiar și știri, cuvintele au o greutate mult mai mare și e nevoie ca ele să fie scrise corect.

Având în vedere aceste aspecte, scrierea corectă cu diacritice este o necesitate. Odată cu dezvoltarea tehnologică, oamenii folosesc tot mai mult metode digitale de scriere și transmitere a textului, care înlocuiesc scrisul de mână. În România la momentul actual sunt folosite cu precădere

tastaturi ce au doar literele alfabetului limbii engleze, iar pentru a introduce și caractere cu diacritice este nevoie de mai mult efort, de exemplu utilizarea unor combinații de taste, este mult mai la îndemână scrierea fără diacritice.

Un sistem de restaurare a diacriticelor are principalul rol de a identifica într-un mod corect și cât se poate de automat caracterele fără diacritice și de a le înlocui, în funcție de cuvântul și contextul în care se află. Totodată, sistemul trebuie împachetat într-o aplicație capabilă să se ocupe de preprocesarea și postprocesarea textului.

În momentul de față există astfel de servicii web, cum sunt „diacritice.opa.ro” și „diacritice.com” care se ocupă cu rezolvarea acestei probleme, însă metoda utilizată de acestea nu este una automată. Ea se bazează pe compararea cuvintelor cu un vocabular, urmată de alegerea diacriticelor corecte de către utilizator în cazul în care există mai multe forme. Există totuși și servicii web ce folosesc metode automate, bazate pe inteligență artificială, de exemplu diacritice.ai, dar care nu este în totalitate gratuit deoarece există o limită a numărului de caractere ce pot fi introduse.

Impactul existenței unui astfel de sistem este unul considerabil din punct de vedere al timpului economisit. În cazul unei persoane care scrie, din diverse motive, un text de dimensiuni mari cu totul fără diacritice, pasul următor ar fi reluarea textului și adăugarea lor de mână, lucru care, în funcție de dimensiunea textului, se face într-un timp considerabil de lung. Folosind un sistem care ia fiecare propoziție din text și pune diacriticele în locurile corecte se poate rezolva problema într-un timp mult mai scurt, de ordinul minutelor.

Aplicabilitatea sistemului există și în cazul platformelor mobile, cum sunt telefoanele. Acestea beneficiază de o putere de calcul suficientă și de suport software astfel încât sistemul să poată fi integrat. Deja există aplicații mobile care corectează textul scris de utilizator, doar că nu iau în considerare și problema diacriticelor, astfel că sistemul ar putea interveni în situația de față. Știm deja că aproape fiecare dintre noi deține câte un telefon mobil și că, datorită posibilității conectării acestora la internet, sunt trimise tot mai multe mesaje scrise. Deoarece unele mesaje sunt de natură importantă, adăugarea sistemului pe o platformă de tip telefon mobil ar avea un impact la scară largă.

În lumina digitalizării vieții oamenilor, dar și a evoluției rapide a tehnologiei, volumul de date existente în format digital a crescut considerabil, la fel și puterea de calcul a dispozitivelor precum plăcile grafice și procesoarele. Acești factori au dus la posibilitatea antrenării cu succes a algoritmilor de învățare automată bazați pe rețele neurale profunde. Problema restaurării de diacritice se poate rezolva cu mare acuratețe folosind astfel de algoritmi.

Dezvoltarea unui astfel de sistem nu se poate face, însă, fără o bază de antrenament destul de mare. Deși există deja corpusuri de text mari în limba română, acestea prezintă două mari probleme, una este lipsa diacriticelor corecte, iar alta este faptul că vocabularul format de acestea nu conține și cuvinte ce au început să fie recent folosite în limba română.

1.2 OBIECTIVE

Având în vedere importanța scrierii cu diacritice dar și necesitatea unei unelte care să ajute la restaurarea acestora în situația în care ele lipsesc, obiectivul principal al lucrării este dezvoltarea unui sistem de restaurare a diacriticelor. Sistemul trebuie să poată restaura cu precizie diacriticele pe propoziții în limba română folosindu-se de context. Pentru a putea obține acest lucru, au fost tratate următoarele obiective specifice:

a. Colectarea unui corpus de text în limba română de dimensiuni mari, cu diacritice puse corect și care să acopere un vocabular cât mai divers.

b. Proiectarea și implementarea unei rețele neurale profunde pentru restaurarea diacriticelor și studiul performanțelor acesteia în funcție de felul în care sunt variate valorile hiperparametrilor și arhitectura.
c. Implementarea unei aplicații de restaurare a diacriticelor utilizând modelul cel mai performant obținut.

1.3 STRUCTURA LUCRĂRII

Capitolul 2 prezintă starea artei în domeniul restaurării de diacritice, metodele utilizate, rezultatele experimentale obținute precum și particularitățile fiecărei lucrări. De asemenea vor fi prezentate și metricile de performanță ce pot fi folosite în evaluarea unui model specializat pe sarcina restaurării diacriticelor.

Capitolul 3 descrie, din punct de vedere teoretic, conceptele care stau la baza metodelor cele mai comune de restaurare de diacritice și de procesare a limbajului natural în general. Acestea sunt metodele statistice bazate pe n-grame și rețelele neurale. Despre modelele statistice se va face o scurtă prezentare, pe când despre rețelele neurale se va intra în mai multe amănunte.

Capitolul 4 este dedicat prezentării tipurilor de tehnologii utilizate în experimente, mai exact limbajul de programare Python, librăriile Keras, Tensorflow, Hyperopt, baza de date MongoDB și librăria CUDA. După fiecare prezentare a tehnologiei se va menționa scopul pentru care a fost utilizată în elaborarea lucrării.

Capitolul 5 se concentrează pe metodele de restaurare de diacritice propuse, arhitecturi de rețele neurale. Acesta va prezenta metoda la modul general, urmând să fie particularizată pentru trei tipuri de rețele: la nivel de caracter, la nivel de cuvânt și hibride. Vor fi menționate detalii precum codificarea datelor și arhitectura rețelei în fiecare caz în parte.

Capitolul 6 descrie detaliile de implementare a sistemului. Vor fi listate și prezentate bucăți de cod utilizate în etape precum definirea modelului, antrenare, testare, preprocesare date, optimizare hiperparametri, interfață pentru restaurare și colectare corpus de text.

Capitolul 7 prezintă modul de desfășurare al experimentelor, mai exact configurația experimentală utilizată și experimentele realizate, alături de motivația și rezultatul lor. Configurația experimentală se referă la seturi de date folosite și resurse hardware și software.

CAPITOLUL 2

STAREA ARTEI ÎN RESTAURAREA DE DIACRITICE

2.1 LUCRĂRI PE ACEEAȘI TEMĂ

Problema restaurării de diacritice a mai fost abordată, atât ca și cercetare, dar și într-un mod pragmatic. Există lucrări în literatura de specialitate pe această temă care tratează problema în diferite feluri, unele folosind modele statistice, iar altele folosind rețele neurale și există și implementări de rețele neurale puse la dispoziție pe internet. Un scurt rezumat al acestora, al felului de desfășurare al experimentelor, precum și rezultatele experimentale sunt prezentate în acest capitol. De observat este faptul că în fiecare lucrare sunt folosite seturi de antrenare și evaluare diferite, iar rezultatele nu sunt cu adevărat comparabile. Totuși, ele pot da o idee asupra performanțelor sistemelor deoarece se folosesc aceiași indici de performanță.

- În lucrarea Tufiș, 2008 [1] se folosesc înlocuiri cu cuvinte dintr-un dicționar, mai exact cu acele cuvinte care au probabilitate maximă de apariție și, în cazul cuvintelor necunoscute, se calculează la nivel de caracter șirul de caractere cu probabilitate maximă de apariție în acea situație. Cu alte cuvinte, se folosește un model n-gram la nivel de cuvânt și unul la nivel de caracter. Setul de date utilizat este reprezentat de un corpus format din texte dintr-o revistă săptămânală și texte juridice.
- Petrică, 2014 [2] folosește tot un model statistic, bazat pe n-grame la nivel de cuvânt. Datele de antrenare sunt obținute în urma extragerii conținutului articolelor de pe site-uri

de știri în limba română prin intermediul unui program ce face acest lucru într-un mod automat. Se ridică și ipoteza faptului că textul este predispus la greșeli, iar în acest sens este trecut printr-o aplicație de curățare și prelucrare.

- Rușeți, 2018 [3] tratează sarcina folosind rețele neurale recurente. Ca o privire de ansamblu, rețeaua este compusă din trei căi: una pentru fereastra curentă de caractere, una pentru cuvântul curent și una pentru cuvintele propoziției curente. Fereastra de caractere și cuvintele propoziției sunt trecute fiecare printr-o rețea recurentă cu celule LSTM, bidirecțională, iar apoi rezultatele tuturor celor trei căi sunt concatenate. La ieșire, rețeaua clasifică, la nivel de caracter, ce fel de diacritică trebuie adăugată sau dacă nu este nevoie de nicio modificare. Setul de date de antrenare este format dintr-un corpus care conține transcrieri ale dezbaterilor din Parlamentul României și totalitatea articolelor de pe site-ul Wikipedia, la momentul respectiv, care sunt scrise în limba română.
- Cristescu, 2018 [4] utilizează rețele neurale atât recurente cât și convoluționale pentru rezolvarea sarcinii. Utilizează ideea de căi separate pentru caractere și pentru cuvinte. Calea de caractere trece prin straturi convoluționale, iar cea de cuvinte este trecută printr-o rețea recurentă. Calea cuvintelor este mapată în spațiul caracterelor printr-o hartă de proiecție, ce este dată la intrare rețelei și este obținută în pasul de preprocesare a datelor. După proiecție, cele două căi sunt concatenate, rezultatul este trecut mai departe prin alte straturi convoluționale, iar la ieșire rețeaua face clasificarea la nivel de caracter. În funcție de clasa prezisă, se decide dacă este corect caracterul sau nu. În cazul în care nu este corect, clasa determină tipul de diacritică ce trebuie să îi fie pusă. Pentru antrenare au fost folosite articolele scrise în limba română pe Wikipedia și baza de date OpenCrawl.

Rezultatele experimentelor sunt prezentate în Tabel 2.1.

2.2 INDICI DE PERFORMANȚĂ

În proiectarea și dezvoltarea unui orice fel de sistem este nevoie de anumiți indici cu ajutorul cărora să se poată face evaluarea performanțelor sistemului. Acești indici sunt de cele mai multe ori niște valori scalare și pot reprezenta măsuri precum eroarea sau precizia sistemului. În lucrările științifice studiate legate de restaurarea de diacritice se pot remarca anumite măsuri ce se repetă și care pot fi folosite cu ușurință și cu succes în evaluarea sistemului. De menționat este faptul că acești indici de performanță se calculează folosind atât ceea ce a prezis sistemul cât și ceea ce ar fi trebuit să prezică sistemul, adică evaluarea trebuie făcută tot pe un set de date etichetat.

- Eroarea la nivel de cuvânt – este un indice de performanță standard utilizat de obicei în sistemele automate de recunoaștere de vorbire. Se calculează drept numărul de cuvinte inserate, șterse sau substituite eronat obținute din sistem raportat la numărul total de cuvinte obținute din sistem. În caz particular, pentru această sarcină este vorba doar de cuvinte substituite greșit. Poate fi folosit în evaluarea sistemului de restaurare a diacriticelor pentru a observa cantitatea de cuvinte greșite existentă în text. În mod ideal aceasta este 0.

$$WER\ [\%] = \frac{Număr\ cuvinte\ greşite}{Număr\ cuvinte} \tag{2.1}$$

- Eroarea la nivel de caracter – un indice de performanță asemănător WER cu diferența că numărătoarea se face la nivel de caracter. De asemenea, în mod ideal este 0.

$$ChER\ [\%] = \frac{Număr\ caractere\ greşite}{Număr\ caractere} \tag{2.2}$$

- Precizie, reamintire și scorul F – indici întâlniți în problemele de predicție binară, în regăsirea informației. Pornesc de la ideea împărțirii prezicerilor sistemului în pozitiv, sistemul a pus diacritică, și negativ, sistemul nu apus diacritică. Mai departe, acestea se împart în adevărate și false, adică detecții:
    - Adevărat pozitive, sistemul a pus în mod corect o diacritică.
    - Fals pozitive, sistemul a pus o diacritică greșită.
    - Adevărat negative, sistemul nu a pus diacritică acolo unde nu trebuia.
    - Fals negative, sistemul nu a pus diacritică acolo unde trebuia.

  Precizia este raportul dintre numărul de diacritice introduse corect și numărul de diacritice din textul prezis, iar reamintirea este raportul dintre numărul de diacritice introduse corect și numărul de diacritice din textul original. Scorul F este media armonică a preciziei și reamintirii. [Cucu, 2013] [5]

$$Precizia = \frac{Adevărat\ pozitive}{Adevărat\ pozitive + Fals\ pozitive} \tag{2.3}$$

$$Reamintirea = \frac{Adevărat\ pozitive}{Adevărat\ pozitive + Fals\ negative} \tag{2.4}$$

$$F_1 = 2 \cdot \frac{Precizia \cdot Reamintirea}{Prezicia + Reamintirea} \tag{2.5}$$

Lucrare	WER [%]	ChER [%]
[Tufiș, 2008]	2.25	0.60
[Petrică, 2014]	0.52	0.11
[Rușeți, 2018][1]	0.99	0.21
[Cristescu, 2018]	-	0.14

Tabel 2.1 - Rezultate experimentale

[1] Lucrarea utilizează o metrică diferită, prin care WER este calculat raportat la numărul total de cuvinte care acceptă diacritice. Indicele din tabel folosește totuși aceeași metrică pe care o folosesc și celelalte lucrări. Pentru a ajunge la aceasta, s-a înmulțit rezultatul cu raportul dintre numărul de cuvinte care acceptă diacritice și numărul total de cuvinte, oferite în articol. Același lucru este și în cazul ChER.

CAPITOLUL 3

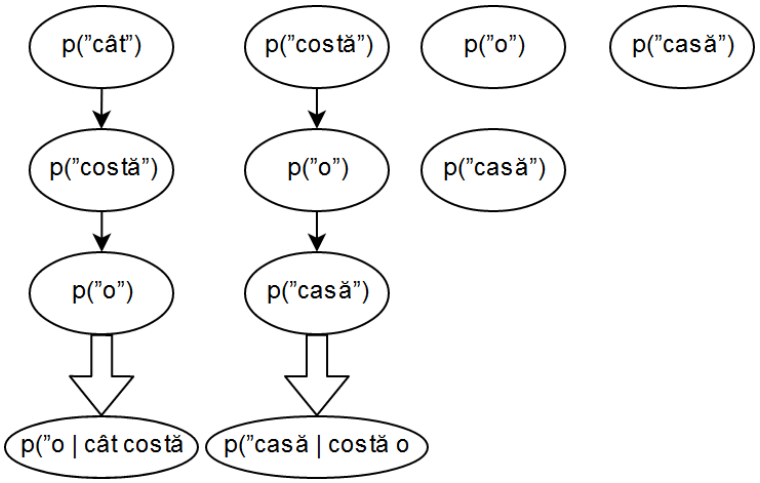
NOȚIUNI TEORETICE DESPRE METODE

3.1 MODELE STATISTICE DE TIP N-GRAM

Modelele statistice folosesc teoria probabilităților pentru a extrage informații din seturile de date. Un tip de model statistic des folosit în domeniul procesării de text se bazează pe n-grame.

Un n-gram reprezintă o secvență de n cuvinte. Modelele statistice ce utilizează n-gram estimează probabilitatea celui de al n-lea cuvânt având în vedere cele n-1 cuvinte anterioare acestuia din secvență. Având la dispoziție secvența de n-1 cuvinte și o listă de cuvinte care ar putea lua poziția n, se pot genera propoziții luând cuvântul care are cea mai mare probabilitate să apară după secvența de n-1 cuvinte [6].

Estimarea probabilităților se realizează parcurgând un corpus de text și, pentru fiecare secvență de n cuvinte întâlnită, se ține cont de câte ori apare aceasta. Mai departe, se reia acest procedeu pentru secvențele de lungime tot mai mică până se ajunge la numărul de apariții doar ale cuvintelor. În urma calculului acestor probabilități se obține un model de limbă, capabil să poată fi folosit în rezolvarea unor sarcini precum recunoașterea vorbirii, traducerea automată sau chiar și restaurarea de diacritice.



Figură 3.1 - Obținerea probabilităților unui model 3-gram

În Figură 3.1 este exemplificat felul în care este tratat calculul unor n-grame dintr-o propoziție, în cazul particular în care $n = 3$, iar propoziția de intrare este ”cât costă o casă”.

Pentru a putea vizualiza ideea de extras secvențe de 3 cuvinte și calculat probabilitatea lor, în figură este considerat că din propoziția inițială se obține, prin deplasarea ei la stânga cu un cuvânt, o altă propoziție, iar aceasta este din nou deplasată la stânga cu un cuvânt și se obține o altă propoziție. Cuvintele aflate pe aceleași poziții în toate aceste propoziții sunt puse împreună și sunt obținute secvențele de lungime 3 din propoziția inițială.

Din punct de vedere matematic, relația de calcul a probabilităților în raport cu numărul de apariții poate fi scris sub forma următoare: [7]

$$p(”cât\ costă\ o”) = \frac{N(”cât\ costă\ o”)}{N(”cât\ costă”)} \tag{3.1}$$

După obținerea probabilităților secvențelor de lungime 3 se calculează probabilitățile secvențelor de lungime 2, urmată de calculul probabilităților cuvintelor. Astfel, cu cât se dorește un model de limbă ce cuprinde secvențe de lungimi mari cu atât volumul de calcul crește.

Folosind aceste probabilități se poate calcula probabilitatea de apariție a unei secvențe de lungime mai mare decât 3. Acest lucru se realizează cu ajutorul regulii de înlănțuire a probabilităților.

$$\begin{gathered} p(”cât\ costă\ o\ casă”) = \\ = p(”cât”, ”costă”, ”o”, ”casă”) = \\ = p(”cât”) \cdot p(”costă”|”cât”) \cdot p(”o”|”cât”, ”costă”) \cdot p(”casă”|”cât”, ”costă”, ”o”) \end{gathered} \tag{3.2}$$

Modelul de limbă n-gram poate fi implementat cu ușurință iar utilizarea lui în sarcini precum restaurarea de diacritice poate da rezultate foarte bune însă complexitatea de calcul crește odată cu creșterea dimensiunii maxime a secvențelor de cuvinte, iar flexibilitatea modelului este scăzută, deoarece el se bazează foarte mult pe ideea prezicerii unor construcții și cuvinte ce au mai fost întâlnite în setul de antrenare.

3.2 Rețele neurale artificiale

3.2.1 Generalități

O rețea neurală poate fi descrisă printr-un graf în care fiecare nod corespunde unui neuron iar fiecare muchie corespunde unei ponderi. Fiecare neuron calculează suma ponderată a ieșirilor neuronilor conectați la acesta și, în unele cazuri, adună un termen de polarizare. Numărul rezultat

este apoi trecut printr-o funcție de activare și este astfel obținută ieșirea neuronului curent [8]. Din punct de vedere statistic, neuronul modelează o regresie liniară, la care se adaugă activarea, adică partea de neliniaritate.

Ecuația (3.3) prezintă formula matematică de calcul a ieșirii unui neuron, pentru care intrarea este un vector X. W este vectorul ponderilor neuronului, iar B este vectorul termenilor de polarizare a neuronului, cu ajutorul cărora se modelează o funcție liniară. Funcția aplicată acestora este activarea, care, în acest exemplu, este funcția tanh, dar pot fi folosite și alte variante.

$$Z = \tanh(W^T \cdot X + B) \tag{3.3}$$

La intrarea rețelei sunt aduse exemple de antrenare, reprezentate, de obicei, prin vectori de numere reale. În contextul unei antrenări supervizate este pusă la dispoziția rețelei ieșirea ce ar trebui obținută. Aceasta poate fi un vector sau un scalar, în funcție de problema ce se dorește a fi rezolvată.

Trecerea unui exemplu printr-o rețea și obținerea unei ieșiri se numește propagare în față. În ideea reglării ponderilor această etapă este urmată de o etapă de propagare în spate. Folosind numerele obținute la ieșirea rețelei se calculează eroarea acesteia cu ajutorul unei anumite funcții, aleasă în funcție de tipul de problemă ce se dorește rezolvată. În contextul unei învățări supervizate, eroarea dată de ieșirea rețelei este calculată relativ la ieșirea pe care ar fi trebui să o dea rețeaua. Folosind valoarea erorii rețeaua va începe procesul propagării în spate. Acesta presupune utilizarea unui algoritm de optimizare a funcției cost iar variabilele de stare sunt ponderile și polarizările neuronilor. De obicei se utilizează algoritmi de optimizare multi-dimensionali derivativi precum gradientul negativ.

Un ciclu complet propagare față-spate a unui exemplu de antrenare se numește o iterație. Trecerea tuturor exemplelor de antrenare se numește epocă. De obicei antrenarea se face în mai multe epoci, în ideea obținerii unei erori cât mai mici. Totuși, nu se poate antrena cu oricât de multe epoci, deoarece există riscul unei supraantrenări, care reduce posibilitatea de generalizare, iar rețeaua va avea rezultate foarte bune doar pe exemplele de antrenare, nu și pe alte exemple noi.

Principalele categorii de rețele neurale sunt cele de tip unidirecționale și cele recurente [9]. Diferența dintre ele este reprezentată de felul prin care este trecută informația prin rețea. În prima situație informația circulă într-un singur sens, doar de la intrare spre ieșire, iar în a doua situație rețeaua transmite informații despre momentul curent către momentul următor sau către momentul trecut.

3.2.2 Rețele neurale unidirecționale

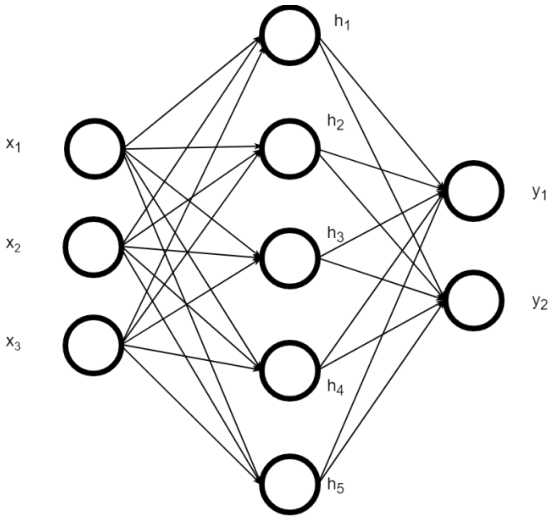
Principalul scop al acestui tip de rețea este aproximarea unei funcții obiectiv f^*, utilizând mai mulți neuroni ce modelează regresia liniară [10]

Există un strat de neuroni la intrare, de dimensiune egală cu dimensiunea spațiului vectorial din care provin informațiile, în componența căruia nu sunt neuroni propriu-ziși, care antrenează niște parametrii, ci este doar calea de intrare a informației în rețea, nealterată.

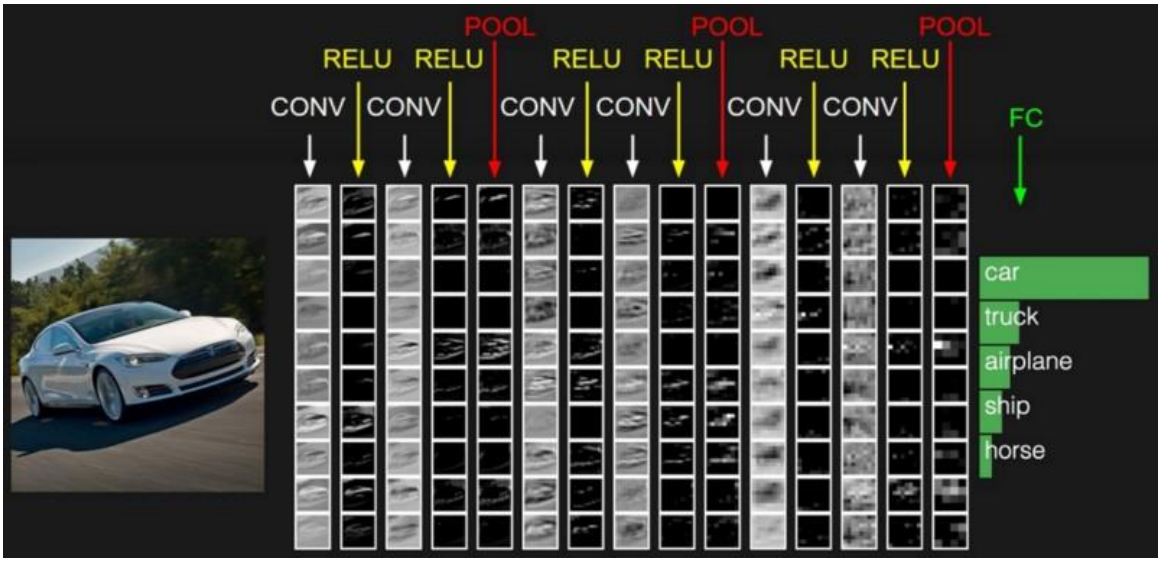
Apoi urmează un număr de straturi ascunse, responsabile de învățarea funcției de aproximat, cu cât numărul de straturi ascunse e mai mare cu atât rețeaua este considerată mai profundă.

La ieșire există un alt strat de neuroni, de dimensiune egală cu numărul claselor ce se doresc a fi obținute, a căror activare este aleasă în așa fel încât să reiasă date despre clasa din care face parte informația de la intrarea rețelei, cu ajutorul a ceea ce au învățat straturile ascunse.

Deși în domeniul învățării automate există deja modele liniare, ele sunt limitate doar la funcții liniare, pe când rețelele neurale nu sunt. Acestea nu au dificultăți în învățarea unei probleme în care datele nu sunt separabile liniar. Straturile ascunse sunt folosite pentru a crește non-liniaritatea și a schimba reprezentarea datelor pentru a putea generaliza mai bine funcția.



Figură 3.2 – Perceptronul multi-strat



Figură 3.3 - Extragere de atribute și clasificare într-o rețea convoluțională [28]

Cel mai simplu model de rețea neurală este perceptronul multi-strat, în care este folosit un singur strat ascuns. Acesta se descurcă foarte bine pe un volum relativ mic de date. Modelele mai profunde folosesc un număr mai mare de straturi ascunse cu care reușesc să învețe probleme complicate dintr-un volum mare de date. În Figură 3.2 se poate observa un perceptron multi-strat cu 3 neuroni la intrare, reprezentând dimensionalitatea acceptată a vectorilor de la intrare, un strat ascuns cu 5 neuroni și 2 neuroni la ieșire, reprezentând numărul de clase posibile la ieșire.

Un model particular de rețea unidirecțională este cea convoluțională. Acest tip de rețea este folosită cu precădere în domeniul prelucrării imaginilor, dar poate avea și alte aplicații. În situații în care spațiul de intrare este prea mare, cum este cazul imaginilor, numărul de neuroni de la intrare este și el mult prea mare, de exemplu 65536 neuroni în cazul unei imagini 256x256 pixeli. Această problemă se rezolvă prin extragerea de caracteristici, care reduce dimensionalitatea vectorilor de la intrare.

Extragerea de caracteristici se face folosind de mai multe ori un șir de operații – convoluție, aplicarea unei funcții de activare și sondarea. Convoluția cu nuclee este folosită și în domeniul procesării imaginilor. Diferența esențială este faptul că, în mod tradițional, ponderile nucleelor sunt fixe și determină rolul lor, iar în contextul învățării automate ponderile nucleelor sunt antrenabile, adică acestea se modifică asemeni ponderilor neuronilor. În urma unei antrenări, straturile convoluționale învață să extragă anumite tipuri de caracteristici, considerate relevante

pentru rezolvarea sarcinii. Straturile de sondare sunt cele responsabile de reducerea dimensionalității.

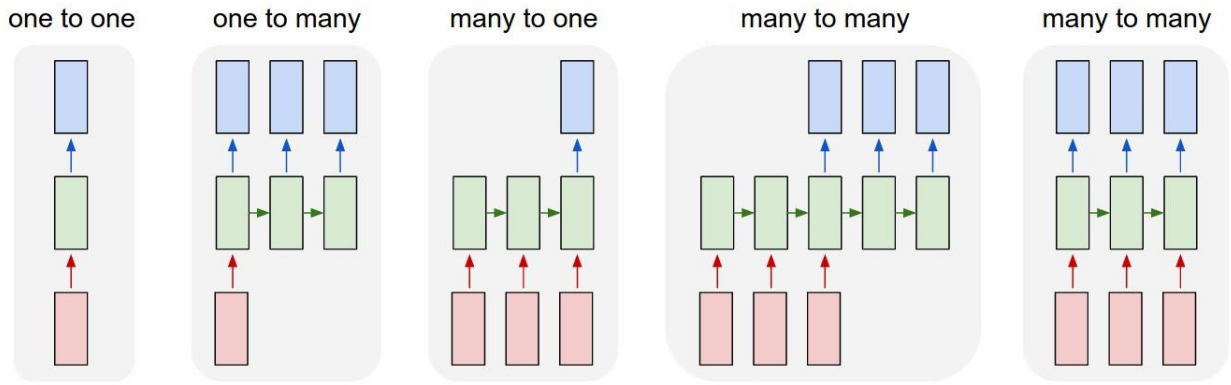
În Figură 3.3 pot fi vizualizate atributele care sunt extrase în straturile convoluționale ale unei astfel de rețele, dar se observă și faptul că după aceste straturi atributele sunt trecute printr-o rețea de tipul perceptronului multi-strat, ce are rol de clasificare.

3.2.3 Rețele neurale recurente

Rețelele neurale recurente (RNR) au două mari avantaje față de rețelele unidirecționale

i. RNR au noțiunea ordinii temporale și iau în considerare și alte informații decât cele date de exemplul de la momentul curent.
ii. Dimensiunile vectorilor de la intrare și de la ieșire pot fi variate.

RNR pot fi folosite cu succes în domenii precum prelucrarea semnalelor și procesarea limbajului natural. [11]



Figură 3.4 - Tipuri de RNR în funcție de numărul de intrări și ieșiri [29]

RNR pot avea dimensiunile vectorilor de intrare și de ieșire variabile, ca în Figură 3.4.

În funcție de aceste dimensiuni se deosebesc mai multe tipuri de aplicații:

- Unu-la-unu – cazul obișnuit, unidirecțional
- O intrare, mai multe ieșiri – de exemplu descrierea într-o propoziție a unei imagini
- Mai multe intrări, o singură ieșire – analiza sentimentului transmis de o propoziție
- Mai multe intrări, mai multe ieșiri după terminarea intrărilor – traducere automată
- Mai multe intrări, mai multe ieșiri în același timp – restaurare de diacritice

RNR iau în considerare atât exemplul la momentul curent cât și ieșirea rețelei pentru exemplu la momentul anterior. Astfel, ieșirea la un anumit moment de timp depinde de restul ieșirilor momentelor anterioare. Acest lucru se obține prin implementarea unei variabile de stare care se obține din valoarea ei anterioară și valoarea exemplului la momentul curent. Mai departe această variabilă de stare ia parte în calculul ieșirii rețelei la momentul curent.

Din punct de vedere matematic procedeul de transport al memoriei poate fi descris astfel:

$$h_t = a(W \cdot x_t + U h_{t-1}) \tag{3.4}$$

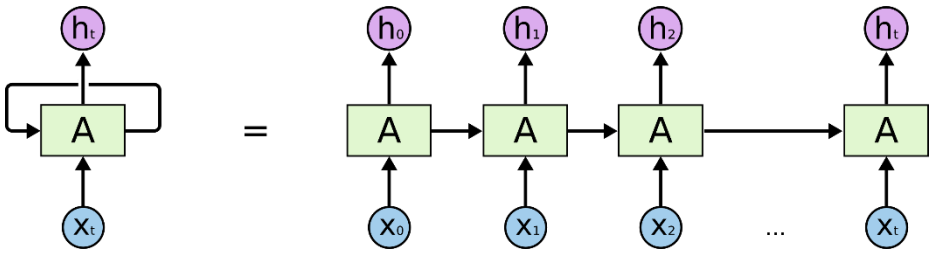
Variabilele utilizate sunt

- h_t ieșirea stratului ascuns la momentul de timp curent t
- x_t intrarea rețelei la momentul de timp curent t
- h_{t-1} ieșirea stratului ascuns la momentul de timp anterior t-1
- U matrice de tranziții între straturi ascunse
- W matrice de ponderi

Desfășurarea temporală a unei astfel de rețele este prezentată în Figură 3.5.

Se observă felul în care, la fiecare moment de timp, ieșirea celulei recurente este trimisă către celula corespunzătoare momentului următor de timp.

Dacă întreaga secvență este pusă la dispoziția rețelei, se mai poate adăuga un strat de RNR care, pentru momentul curent, să aibă în vedere informație legată de momentele ulterioare celui curent. Astfel de model, în care la fiecare moment avem informații atât din trecut cât și din viitor, se numește rețea recurentă bidirecțională.



Figură 3.5 - Desfășurarea în timp a unei RNR [30]

Asemeni rețelelor unidirecționale, ieșirea celulelor unei RNR pot fi puse la intrarea unei alte rețele recurente, obținându-se astfel un model de rețele stivuite.

3.2.3.1 Rețele neurale recurente cu celule de memorie

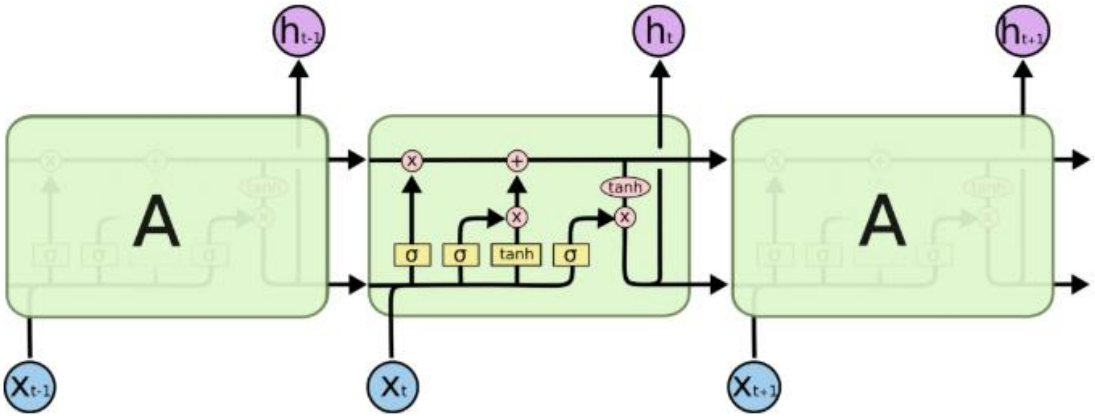
O problemă importantă prezentă la acest tip de rețele este dispariția gradienților, adică informația se pierde pe măsură ce se avansează temporal. Cu alte cuvinte, rețeaua uită, iar acest lucru se poate rezolva prin adăugarea unei memorii. Astfel s-au dezvoltat, ca extensii ale RNR obișnuite, rețelele cu celule LSTM și celule GRU. LSTM sunt cele mai populare și des folosite. Celulele de tip LSTM îi permit RNR-ului să țină minte informația pentru perioade lungi de timp, deoarece o țin într-o memorie asemănătoare celei din sistemele de calcul, prin faptul că asupra ei se pot face operații de citire, scriere și ștergere.

În funcție de importanța informației, celula LSTM poate decide să o păstreze sau să o șteargă. Procesul de decizie se bazează pe calcule ce folosesc anumite ponderi, care pot fi învățate de algoritm. Prin posibilitatea antrenării acestor ponderi se poate învăța ce este important și ce nu.

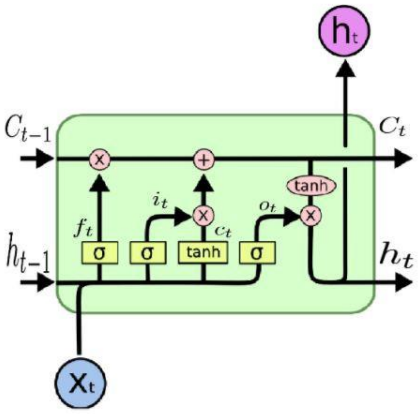
Celula LSTM se folosește de 3 porți, intrare, uitare și ieșire pentru a decide dacă să stocheze informația, să o șteargă sau să o lase să aibă un impact asupra ieșirii la momentul curent. Aceste porți se folosesc de funcția sigmoidă, astfel încât există posibilitatea derivării acesteia pentru a putea lua parte la procesul de propagare în spate.

În Figură 3.7 este descris principiul de funcționare a celulei LSTM. Sunt folosite notațiile

- f_t pentru poarta de uitare – determină dacă se șterge conținutul celulei;
- i_t pentru poarta de intrare – determină dacă se scrie în celulă;
- o_t pentru poarta de ieșire – determină ieșirea celulei.



Figură 3.6 - Rețea recurentă cu celule de tip LSTM [30]



Figură 3.7 - Celula LSTM [30]

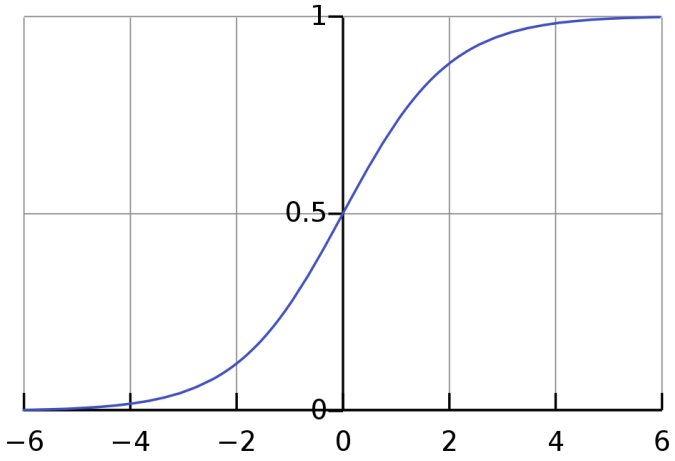
3.2.4 Funcția de activare

Este o funcție neliniară aplicată de un neuron pentru a introduce proprietăți de neliniaritate în rețea. Există mai multe variante de funcții de activare utilizate de către rețele, alegerea lor făcându-se atât pe baza performanțelor oferite, cât și pe baza sarcinii ce se dorește a fi rezolvată. Principalele funcții de activare sunt prezentate în cele ce urmează. [12]

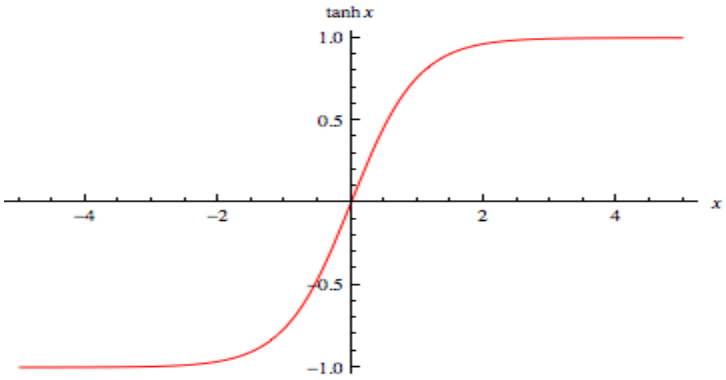
3.2.4.1 Sigmoid

Este o funcție neliniară ce poate lua valori în intervalul [0, 1]. Poate fi folosită în aproximarea unor situații complexe datorită caracterului neliniar. Însă aceasta suferă de problema dispariției gradienților datorită faptului că pe măsură ce variabila de intrare se îndepărtează de 0 valorile funcției tind doar la 0 sau la 1, iar panta este aproape dreaptă. Funcția este și costisitoare din punct de vedere computațional.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.5}$$



Figură 3.8 - Graficul funcției sigmoid



Figură 3.9 - Graficul funcției tangent hiperbolică

3.2.4.2 Tangentă hiperbolică

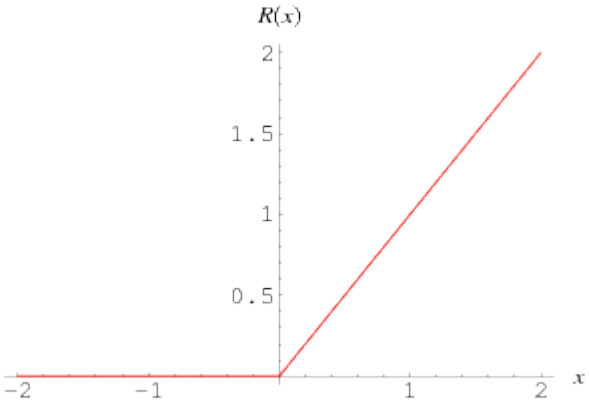
Poate lua valori în intervalul [-1, 1]. Asemănător funcției sigmoid, are un caracter neliniar prin care ajută la rezolvarea problemelor complexe, dar apare problema dispariției gradienților. Are totuși avantajul că este centrată în 0 și poate modela cu ușurință valori cu caracter puternic pozitiv, negativ sau neutru.

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.6}$$

3.2.4.3 ReLU

Funcție de rectificare liniară, sau funcție rampă, care poate lua valori foarte mari deoarece nu este limitată. Este eficientă din punct de vedere computațional și, deși nu pare, este neliniară. Nu poate lua valori negative. Este cea mai populară alegere de funcție de activare pentru rețele neurale profunde datorită faptului că nu mai apare problema dispariției gradienților.

$$f(x) = \max(0, x) \tag{3.7}$$

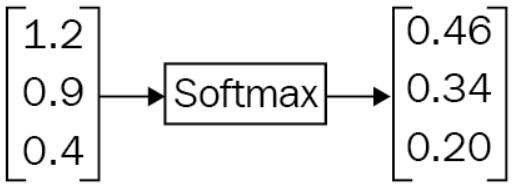


Figură 3.10 – Graficul funcției ReLU

3.2.4.4 Softmax

Poate lua valori de la 0 la 1. Este utilizată în straturile de ieșire pentru a oferi probabilitățile de apartenență la clasele urmărite de rețea.

$$softmax(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{3.7}$$



Figură 3.11 - Transformarea valorilor folosind Softmax [13]

3.2.5 Funcția cost

Scopul unei rețele neurale este acela de a găsi ponderile adecvate astfel încât la ieșirea ei să se regăsească o predicție bună. Acest lucru se obține prin obținerea unei erori mici, deci se dorește minimizarea erorii [14]. Un aspect esențial al erorii este funcția prin care aceasta este calculată. Funcția aceasta se numește funcție cost sau funcție eroare. Funcția cost reduce un sistem complex la o singură valoare scalară care permite soluțiilor să fie comparate și ordonate [15].

Alegerea funcției cost poate fi o problemă complicată deoarece aceasta trebuie să capteze proprietățile problemei și să fie motivată de aspectele importante ale proiectului.

În funcție de ieșirea dorită se pot folosi diferite tipuri de funcții cost. Vor fi prezentate cele mai frecvent utilizate.

- În cazul unei regresii, în care rețeaua are un singur număr scalar la ieșire, se poate folosi eroarea pătratică medie. Formula acesteia este una simplă care se găsește în relația (3.8).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2 \tag{3.8}$$

- Pentru rezolvarea unei sarcini de tip clasificare binară sau clasificare pe mai multe clase, se folosește entropia încrucișată.

$$J(w) = -\frac{1}{N}\sum_{n=1}^{N}[y_n log\hat{y}_n + (1 - y_n)\log(1 - \hat{y}_n)] \tag{3.9}$$

3.2.6 Algoritmul de optimizare

Sunt utilizați algoritmi de optimizare multidimensională, ce pot fi împărțiți în mai multe categorii [16].

- Derivativi de ordinul întâi, care optimizează folosind gradientul funcției. Cel mai des folosit algoritm este cel al gradientului negativ.
- Derivativi de ordinul doi, care optimizează folosind matricea Hesiană. Acest tip de algoritm este mai rapid doar dacă este cunoscută derivata a doua. Altfel, calculul acesteia este costisitor din punct de vedere al memoriei și al timpului și sunt preferați algoritmii de ordinul întâi.
- Nederivativi, care nu folosesc informații despre gradientul funcției.

3.2.6.1 Gradientul negativ

Este un algoritm de optimizare multidimensional derivativ, care se folosește de gradientul funcției cost pentru a obține noi valori ale ponderilor, relativ la valorile curente, utilizând un pas constant.

Formula gradientului negativ este scrisă la punctul (3.10). În ea se poate vedea cum vectorul de ponderi w este actualizat folosind gradientul funcției cost J în valoarea curentă a ponderii. α este rata de învățare care transmite cât de mare este saltul făcut. Această valoare este constantă, fapt pentru care există șansa ca algoritmul să se deplaseze fie prea mult și să sară peste minim, fie prea încet și să nu ajungă la minim într-un timp rezonabil.

$$w_{k+1} = w_k - \alpha \nabla \mathrm{J}(w_k) \tag{3.10}$$

Gradientul negativ poate să proceseze fie câte un exemplu de antrenare per iterație, fie poate să proceseze setul de antrenare pe loturi, adică să actualizeze ponderile conform mai multor exemple deodată.

În cazul utilizării algoritmului cu câte un exemplu de antrenare, acesta se numește gradient negativ stocastic. El converge la minime bune, dar are fluctuații foarte mari ale valorii costului în urma fiecărei actualizări și are un timp mare de execuție. Procesarea pe loturi oferă o convergență mai stabilă și mai rapidă.

Pentru utilizarea cu succes a acestui algoritm au fost aduse modificări menite să îmbunătățească punctele mai slabe ale sale.

3.2.6.2 Impulsul

La fiecare pas de optimizare se adaugă un nou termen care ia în considerare o fracțiune din pasul anterior de optimizare. Scopul impulsului este acela de a accelera SGD în navigarea pe direcția relevantă și a reduce din fluctuații atunci când se pornește în altă direcție. Se obține o convergență stabilă și oscilații reduse.

Vectorul de actualizare este calculat separat prin formula de la (3.11) și ia în calcul valoarea vectorului de la pasul anterior. Termenul γ se numește termen de impuls și exprimă în ce măsură este luat în considerare vectorul de actualizare anterior. De obicei se alege $\gamma = 0.9$.

$$V(t) = \gamma V(t-1) + \alpha \nabla \mathrm{J}(w_k) \tag{3.11}$$

Folosind această notație, formula de actualizare a ponderilor este (3.12)

$$w_{k+1} = w_k - V(t) \tag{3.12}$$

3.2.6.3 Gradientul accelerat Nesterov

O problemă care apare în metoda utilizării impulsului este faptul că acumularea acestuia pe măsură ce funcția descrește duce la posibilitatea ratării minimului. Acest lucru se întâmplă

deoarece unele funcții pot avea minimul într-o zonă foarte abruptă, foarte ușor de sărit peste. Este nevoie de o metodă prin care algoritmul să nu înainteze prea rapid cât să rateze minimul și prin care acesta să fie mai flexibil în cazul schimbărilor funcției.

GAN rezolvă acest lucru prin aproximarea poziției viitoare în care va ajunge algoritmul și ce valori vor avea ponderile. Astfel formula vectorului de actualizare va fi (3.13).

$$V(t) = \gamma V(t-1) + \alpha \nabla \mathrm{J}\big(w_k - \gamma V(t-1)\big) \tag{3.13}$$

3.2.6.4 Adam

Este o prescurtare pentru estimare adaptivă de moment. Acesta este un algoritm de optimizare care urmărește existența unei rate de învățare separate pentru fiecare parametru în parte. De asemenea, rata de învățare este calculată într-un mod adaptiv, în funcție de gradienții calculați în valorile anterioare ale ponderilor. m_t și v_t sunt momentele statistice ale gradienților și reprezintă media și varianța necentrată.

$$\hat{m}_t = \frac{m_t}{1-\beta_1} \tag{3.14}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2} \tag{3.15}$$

$$w_{k+1} = w_k - \frac{\alpha}{\sqrt{\hat{v}_t + \varepsilon}} \hat{m}_t \tag{3.16}$$

Valorile de obicei alese sunt $\beta = 0.9_1, \beta_2 = 0.999$ și $\varepsilon = 10 \cdot e^{-8}$

Adam funcționează bine în practică în comparație cu alte metode adaptive și converge repede. Algoritmul nu întâmpină probleme precum dispariția ratei de învățare, întâlnită la alți algoritmi de optimizare cu rată de învățare adaptivă, și fluctuațiile funcției cost.

3.2.7 Metode de regularizare

Algoritmii de învățare automată bazați pe rețele neurale au principalul avantaj faptul că reușesc să învețe chiar și funcții cu grad de neliniaritate ridicat. Totuși acest lucru poate duce la situația în care funcția învățată se potrivește foarte bine doar pe exemplele folosite în setul de antrenare, dar nu mai reușește să generalizeze alte exemple din alt set de date. Acest fenomen se numește supraantrenare și reprezintă o problemă care poate fi adresată din mai multe puncte de vedere.

3.2.7.1 Regularizarea funcției cost

Cea mai cunoscută tehnică de regularizarea constă în modificarea funcției cost cu scopul trunchierii valorilor prea mari ale ponderilor. Modificarea este dată de introducerea în funcția cost a unui termen de regularizare care este folosit în combinație cu matricea ponderilor prin diferite feluri. [17]

O metodă este utilizarea normei L2 a matricei de ponderi.

$$J_{nou}(W) = J(W) + \lambda \sum_{i=1}^{n} w_i^2 \tag{3.17}$$

Alta este utilizarea normei L1.

$$J_{nou}(W) = J(W) + \lambda \sum_{i=1}^{n} |w_i| \tag{3.18}$$

O altă metodă folosește ambele norme și fiecare este folosită în combinație cu un alt coeficient de regularizare.

$$J_{nou}(W) = J(W) + \lambda_1 \sum_{i=1}^{n} |w_i| + \lambda_2 \sum_{i=1}^{n} w_i^2 \quad (3.19)$$

3.2.7.2 Regularizarea neuronilor

O altă tehnică de regularizare foarte cunoscută este renunțarea [18]. Renunțarea este o metodă de regularizare utilizată în rețelele neurale care ajută învățarea prin determinarea neuronilor să fie mult mai adaptivi și să nu mai depindă de alți neuroni. Ea constă în tăierea anumitor legături între neuroni la momentul antrenării. Tăierea se face doar pentru anumite legături, alese în mod aleatoriu. Fiecare legătură poate fi tăiată, cu o anumită probabilitate, care se numește rată de renunțare. Astfel se introduce existența unui nou hiperparametru ce influențează performanțele rețelei și care trebuie ales într-un mod optim.

3.2.8 Reprezentarea vectorială a limbajului natural

Într-o aplicație de inteligență artificială ce are ca scop procesarea limbajului natural este nevoie de existența unei mapări din spațiul discret al cuvintelor într-un spațiu vectorial multidimensional, în așa fel încât atât sensul cuvintelor, cât și relația dintre ele să poată fi păstrată. Această mapare va reprezenta un strat suplimentar de intrare în rețeaua neurală. În limba engleză, termenul consacrat pentru reprezentarea vectorială a cuvintelor este *embedding.*

Pentru a realiza acest lucru se folosește un corpus de text din care sunt obținute pozițiile cuvintelor în spațiul vectorial, în funcție de celelalte cuvinte. Există astfel de algoritmi, care primesc un corpus de text la intrare și din care rezultă o matrice formată din reprezentările vectoriale ale tuturor cuvintelor din corpusul de text. Câteva exemple de algoritmi sunt Word2Vec, GloVe și FastText.

Word2Vec este printre cele mai populare tehnici. Prin aceasta se realizează antrenarea matricei folosind o rețea neurală simplă unidirecțională cu un strat, într-un mod auto-supervizat, adică nu este nevoie de intervenția utilizatorului pentru a eticheta datele, algoritmul le generează din setul de date. Acest algoritm poate calcula reprezentările cuvintelor în două moduri. Primul mod preia, pentru fiecare cuvânt, contextul acestuia, îl trece prin rețea și prezice ce cuvânt ar trebui să urmeze. Acest mod este rapid și are reprezentări vectoriale bune pentru cuvintele cele mai frecvente. Cel de-al doilea mod funcționează în mod diferit, fiecare cuvânt este trecut prin rețea și se încearcă prezicerea contextului său. Această metodă funcționează bine cu cantități mici de date și face o reprezentare bună a cuvintelor mai rare. [19] [20]

Totuși, dacă se dorește utilizarea unui strat de reprezentare vectorială dar nu și antrenarea acesteia în prealabil, există o variantă alternativă. Într-o arhitectură de rețea neurală profundă, matricea aceasta poate fi învățată odată cu rețeaua iar reprezentările vectoriale vor avea mai mult sens pentru sarcina de rezolvat a rețelei.

3.2.9 Problemele rețelelor

Rețele neurale profunde sunt o unealtă puternică ce pot rezolva probleme extrem de complexe, cu o capacitate de generalizare și de adaptare superioară altor algoritmi de învățare automată. Totuși ele vin cu anumite obstacole, principalele fiind nevoia existenței unui set de antrenare consistent și problema proiectării propriu-zise a rețelei,

Având în vedere toate componentele rețelei, problema proiectării devine una foarte complexă. Este nevoie în primul rând de stabilirea unei arhitecturi, ce presupune mai multe alegeri precum tipul de rețea, unidirecțională sau recurentă, caz în care se alege tipul de celulă de memorie, numărul de straturi, folosirea de straturi convoluționale, caz în care e nevoie și de proiectarea arhitecturii lor. Apoi e nevoie de alegerea funcțiilor de activare, funcției cost, algoritmului de optimizare și a algoritmului de optimizare. Acestea, la rândul lor determină apariția unor variabile ce necesită reglaj.

Aceste variabile se numesc hiperparametri și ele trebuie, de asemenea, alese. Într-o rețea apar ca hiperparametri variabile precum numărul de epoci, dimensiunea straturilor, rata de învățare sau rata de renunțare.

O astfel de structură complexă poate obține performanțe foarte bune, doar că apare un compromis mare, resursele utilizate. Din punct de vedere spațial, o rețea neurală profundă poate ajunge la zeci de milioane de ponderi antrenabile, lucru care duce la necesitatea unei memorii suficient de mari. Din punct de vedere temporal, antrenarea unui astfel de model poate dura ore sau chiar zile iar, în unele cazuri, chiar și inferența poate dura destul de mult timp. Totuși antrenarea este un proces ce se poate paraleliza foarte bine, fapt pentru care s-a dezvoltat suport pentru realizarea calculelor necesare antrenării pe plăci grafice, pentru scăderea timpului.

Aceste aspecte, a resurselor utilizate, alături de metricile de performanță, trebuie luate în vedere atunci când se fac modificări la arhitectura sau la hiperparametrii unei rețele neurale profunde.

CAPITOLUL 4

TEHNOLOGII UTILIZATE

4.1 LIMBAJUL DE PROGRAMARE PYTHON

Este principalul limbaj de programare utilizat în elaborarea lucrării. Motivația principală pentru folosirea lui este existența librăriilor gratuite ce pot fi folosite în acest limbaj care fac ușoară implementarea algoritmilor de învățare automată, cum sunt rețelele neurale. Câteva exemple ar fi Tensorflow, Keras, PyTorch, Caffe și Scikit-learn.

Un alt motiv este sintaxa de nivel înalt, ușor de înțeles și de utilizat, dat fiind faptul că Python are și un caracter de limbaj scriptic. Chiar și o persoană începătoare fără experiență în programare, care totuși dorește să lucreze cu algoritmi de inteligență artificială poate învăța într-un timp scurt și cu ușurință programare în Python. De asemenea se pot folosi multiple paradigme de programare, precum orientarea pe obiecte sau programarea funcțională.

Gestionarea librăriilor utilizate se poate face cu ușurință prin utilizarea unor aplicații precum Pip sau Conda. Acestea se ocupă atât de descărcarea și instalarea librăriilor, cât și de versionarea lor și asigurarea existenței dependințelor necesare. Fără a utiliza librării suplimentare, Python oferă o librărie standard generoasă.

Python oferă de asemenea posibilitatea utilizării de medii virtuale. Fiecare mediu virtual este constituit din versiunea de Python utilizată și librăriile necesare. Utilizarea mediilor virtuale este folositoare în situația în care programatorul lucrează la mai multe proiecte, iar fiecare proiect folosește o versiune specifică de Python și librării specifice. Un alt avantaj al mediilor virtuale este dat de portabilitatea oferită. Astfel cineva care lucrează la un proiect folosind un mediu virtual îi poate permite altcuiva să lucreze pe același proiect fără ca această persoană să își instaleze manual fiecare librărie. Tot ce are nevoie este fișierul de descriere a mediului virtual, care, folosit împreună cu un program precum Pip, va face instalarea librăriilor necesare proiectului foarte ușoară.

Domeniul de aplicabilitate a limbajului se poate extinde și în alte ramuri precum jocuri, explorarea datelor sau tehnologii web. Având în vedere faptul că în momentul curent cele mai la modă domenii din tehnologia informației sunt aplicațiile web și inteligența artificială, o persoană care cunoaște acest limbaj poate îmbina aceste două domenii și poate dezvolta o aplicație web care folosește și algoritmi de inteligență artificială.

În cadrul lucrării, limbajul de programare Python a fost folosit în scrierea și evaluarea modelului, preprocesarea și postprocesarea datelor, toate acestea folosind atât librării speciale pentru rețele neurale, cât și librăria standard oferită de Python. Pentru a putea rula antrenarea pe mai multe stații de lucru s-au folosit medii virtuale pentru transferarea librăriilor necesare și administratorul Pip pentru instalarea lor.

Descriem în continuare cele mai importante librării de învățare automată disponibile pentru Python și folosite în elaborarea lucrării.

4.2 KERAS ȘI TENSORFLOW

Tensorflow este o librărie gratuită dezvoltată de Google Brain pentru implementarea algoritmilor de calcul numeric utilizați în aplicații de învățare automată. Scopurile principale al librăriei sunt facilitarea implementării modelelor de învățare automată și optimizarea timpului de antrenare. [21]

Metoda de lucru a librăriei este definirea tuturor operațiunilor matematice prin grafuri de computație. Graful are ca noduri operațiuni matematice iar ca muchii valori numerice, care de cele mai multe ori sunt transmise ca vectori multidimensionali, denumiți tensori.

Operațiunile matematice de bază sunt realizate într-un mod cât mai optimizat folosind un nucleu scris într-un limbaj de nivel mai scăzut, C++, în care astfel de operațiuni sunt executate foarte rapid. La dispoziția programatorului este pus la dispoziție un API scris în limbajul Python, astfel oferind o metodă ușoară la nivel înalt de a proiecta un model.

Un mare avantaj al acestei librării este faptul că permite accelerarea operațiilor prin utilizarea procesoarelor grafice, care au o putere de calcul paralel mult mai mare decât procesoarele de uz general. De asemenea, pentru o mai mare accelerare a antrenărilor, Tensorflow oferă posibilitatea antrenării în mod distribuit, printr-un sistem bazat pe transmitere de mesaje.

Deși Tensorflow este o librărie de nivel înalt, deasupra sa se mai poate adăuga un strat de abstractizare, reprezentat de librăria Keras. Dacă în Tensorflow se facilitează definirea operațiilor utilizate într-o rețea neurală printr-un graf, în Keras se poate defini efectiv rețeaua printr-un graf, în foarte puține linii de cod. Mai departe librăria definește operațiunile necesare print intermediul unui nucleu, care poate fi Tensorflow, Theano, CNTK sau chiar și unul definit de programator [22].

Pentru dezvoltare rapidă de prototipuri este o unealtă excelentă, dar dacă se dorește eficiență maximă și mai mult control este recomandată scrierea modelului folosind Tensorflow sau direct într-un limbaj de nivel scăzut.

Astfel, el poate fi folosit pentru a defini rețeaua, strat cu strat, cu posibilitatea de a defini pentru fiecare strat în parte dimensiunile și caracteristicile acestuia. Pe lângă ajutor în definirea arhitecturii rețelei, Keras pune la dispoziție și alte unelte, utilizate în diferite scopuri.

În primul rând pentru ajutor în cadrul procesului de antrenare sunt oferite mai multe utilități.

- Afișarea unei interfețe în linia de comandă care oferă informații despre starea curentă a antrenării, mai exact indicele exemplului curent de antrenare raportat numărul total de exemple, numărul epocii curente raportat la numărul total de epoci, eroarea de antrenare și eventual eroarea de validare.
- Apelarea unor funcții într-un mod automat în anumite stadii ale antrenării. Aceste funcții pot fi definite de programator, deși unele sunt deja sunt oferite de librărie, iar comportamentul lor poate fi suprascris. Ele pot fi folosite în diferite scopuri, cel mai des folosite fiind cele care salvează modelul împreună cu ponderile modificate, cele care realizează un jurnal al evoluției modelului pe parcursul antrenării și cele care opresc antrenarea modelului sau scad rata de învățare în situația în care eroarea nu mai scade.
- Posibilitatea antrenării folosind un obiect de tip generator, prin intermediul căruia se aleg exemplele de antrenare adăugate la începutul fiecărei epoci. Astfel se poate realiza antrenarea fără parcurgerea întregului set de date, ci doar cele selectate de generator, după o regulă care de obicei este o distribuție aleatorie uniformă. Prin această metodă se reduce în mod drastic timpul de antrenare păstrându-se totuși diversitatea exemplelor trecute prin rețea.

În al doilea rând Keras oferă diferite unelte spre a fi utilizate în preprocesarea și postprocesarea datelor.

- Tokenizer, o unealtă utilă în procesarea limbajului natural, cu care se realizează codificarea și decodificarea textelor la nivel de cuvânt.
- Pentru secvențe se pot folosi unelte care se ocupă de aducerea lor la aceeași lungime, prin umplerea acestora cu numere, cum ar fi 0, înainte sau după.
- Pentru preprocesarea etichetelor este pusă la dispoziție o funcție prin care eticheta este transformată dintr-un număr întreg, scalar, într-un vector coloană de dimensiune egală cu numărul claselor. Acest vector are 0 pe toate pozițiile mai puțin pe cea corespunzătoare numărului clasei, unde se regăsește 1.

Atât Keras cât și Tensorflow vin la pachet cu arhitecturi de rețele neurale profunde cu ponderi preantrenate. Ele pot fi folosite fie pentru simplul scop de a face predicții sau pentru a le reantrena. Reantrenarea lor este o tehnică numită învățare prin transfer prin care se poate antrena într-un timp mult mai scurt un model pentru o sarcină nouă folosind un model antrenat mult timp pentru o altă sarcină, asemănătoare. Acest lucru se realizează prin blocarea modificării ponderilor anumitor straturi și permiterea antrenării celorlalte straturi. De obicei doar ultimului strat îi este permis să își modifice ponderile în astfel de situații.

Keras a avut un rol esențial în elaborarea lucrării datorită ușurinței cu care s-au putut implementa proiectarea, antrenarea, evaluarea și observarea evoluției modelului, precum și pentru ajutorul oferit în preprocesarea datelor. Tensorflow a fost folosit ca și nucleu pentru Keras.

4.3 HYPEROPT

O rețea neurală poate atinge performanțe bune cu ajutorul alegerii într-un mod optim a arhitecturii, setului de antrenare și a valorilor hiperparametrilor. Hyperopt este o librărie care își propune să rezolve problema alegerii valorilor hiperparametrilor. Metoda de rezolvare este constituită din găsirea hiperparametrilor rezolvând o problemă de optimizare. [23]

Ea consideră rețeaua ca fiind funcția obiectiv și un indice de performanță, cum ar fi eroarea de antrenament sau de validare, valoarea funcției. Folosind un spațiu de căutare definit de programator, găsește variabilele de control, reprezentate de hiperparametri, care dau rezultatele cele mai bune. În acest proces funcția este considerată o cutie neagră despre care nu avem detalii precum derivatele acesteia.

Evaluarea funcției obiectiv folosind variabilele de control de la un moment dat este reprezentată de antrenarea rețelei cu acei hiperparametri, deci un pas de optimizare este de durată egală cu durata de antrenare a rețelei.

Spațiul de căutare al hiperparametrilor poate să fie discret sau continuu. În cazul celui continuu, alegerea variabilelor de către algoritm se face după anumite legi de distribuție, ale căror momente statistice pot fi setate.

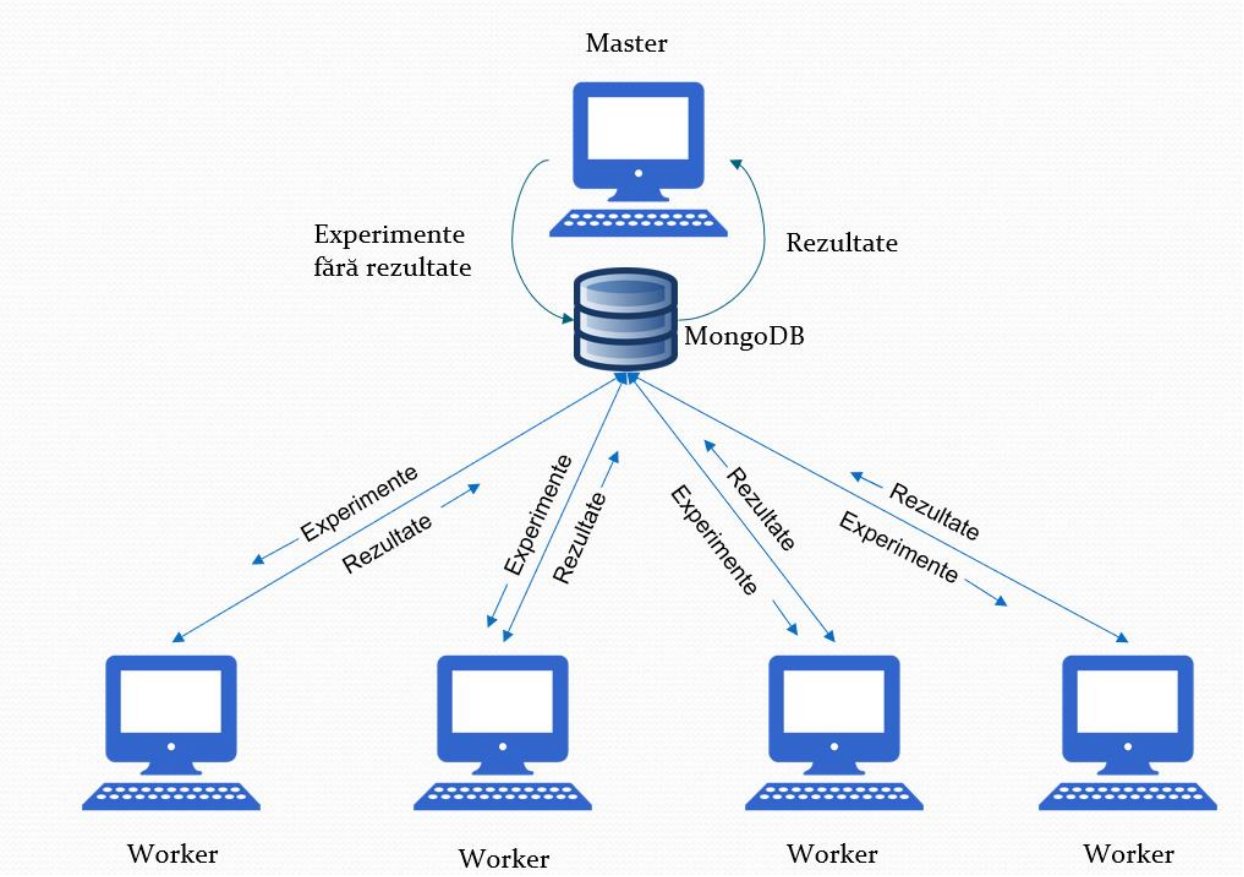
Unul dintre avantajele principale oferite de librăria Hyperopt este capabilitatea algoritmului de optimizare de a alege noi variabile de control într-un mod cât mai inteligent, folosindu-se de rezultatele experimentale anterioare. Pentru a obține astfel de sugestii utilizează algoritmul Estimatorului Parzen Arborescent (TPE). Totuși există și posibilitatea de a nu folosi această metodă și de a face alegerea pur aleatorie, lucru care tot va duce la o soluție, doar că în mai mulți pași de optimizare. Având în vedere că un pas de optimizare este o întreagă antrenare de rețea, alegerea în mod aleatoriu a variabilelor de control duce la un proces de optimizare de hiperparametri foarte costisitor și lent, datorită necesității unui număr mare de pași de antrenare.

Un alt avantaj foarte important al librăriei este faptul că optimizarea hiperparametrilor se poate face cu ușurință într-un mod distribuit, folosind mai multe stații de lucru. Pentru utilizarea în modul distribuit trebuie ca Hyperopt să fie instalat pe toate stațiile de lucru iar funcția de optimizat, adică modelul rețelei, trebuie și ea să fie pusă pe toate stațiile de lucru. Între toate stațiile implicate va exista un intermediar, o bază de date MongoDB, în care vor fi inserate experimente de antrenare dar și rezultatele acestora.

Un singur calculator este cel care creează sarcini și acela este cel master. El este cel care creează sarcini în baza de date. O sarcină conține valorile variabilelor de control de încercat și alte informații precum starea ei, data și ora la care a fost creată și un identificator unic al optimizării curente. Restul calculatoarelor citesc sarcinile din baza de date și își atribuie câte o sarcină care nu a fost încă atribuită. Mai departe execută antrenarea folosind hiperparametrii din sarcină și returnează, în interiorul sarcinii din baza de date, rezultatul. Mai departe calculatorul master va citi din baza de date sarcinile terminate și crea noi sarcini în funcție de ele, iar la final master-ul va afișa hiperparametrii cu care s-a obținut cel mai bun rezultat, în funcție de criteriul stabilit. Acest mod de lucru este descris în Figură 4.1.

Posibilitatea rulării distribuite are un impact destul de puternic având în vedere numărul de hiperparametri ce necesită antrenați și timpul de antrenare necesar unei rețele astfel încât să se obțină rezultate relevante.

În elaborarea lucrării Hyperopt a fost folosit cu scopul optimizării de hiperparametri precum dimensiunea unor straturi sau rata de renunțare și realizarea de experimente legate de performanțele sistemului cu și fără hiperparametri optimizați.



Figură 4.1 - Optimizare de hiperparametri în mod distribuit

4.4 MONGODB

Este o bază de date non-relațională, orientată pe documente. Celelalte tipuri de baze de date non-relaționale sunt cele cheie-valoare (de exemplu Redis), orientate pe grafuri (Neo4j) și cele orientate pe coloane (HBase). [24]

Dintre toate, cele orientate pe documentate sunt cele mai populare și mai adesea folosite. Acestea, față de modelul relațional, au redefinit conceptele precum tabela, aici este o colecție iar rândul are ca echivalent documentul, care este o structură mult mai flexibilă. Colecția poate conține documente cu aceeași structură sau cu structuri diferite.

Datele sunt stocate în format JSON binar, prescurtat BSON. Deoarece se folosește acest format, documentele sunt stocate ca perechi cheie-valoare într-un mod flexibil astfel încât fiecare document să poată avea doar câmpurile de care are nevoie. Un alt avantaj al utilizării JSON apare la implementarea aplicațiilor care se conectează la MongoDB. Se realizează maparea înregistrărilor din baza de date la obiecte din cod într-un mod mult mai ușor, fără a mai fi nevoie de un strat suplimentar pentru acest lucru.

Astfel un avantaj principal este flexibilitatea datelor. Alte avantaje ar mai fi posibilitatea căutării documentelor mai ușor deoarece pot fi folosite interogări după numere într-o gamă sau interogări după text folosind expresii regulare, indexarea datelor pentru o căutare mai rapidă, scalabilitatea verticală și orizontală, posibilitatea stocării de fișiere și posibilitatea replicării bazei de date.

O diferență importantă față de modelul relațional este faptul că nu mai există noțiunea de operații de join. Aici fiecare document poate avea unul sau mai multe documente imbricate. Acest lucru duce la creșterea performanței deoarece operația de join din modelul relațional este una costisitoare din punct de vedere al timpului. Totuși acest lucru duce redundanță și la un consum mai mare al memoriei interne. [25]

Pe lângă consumul mare de memorie există și alte limitări ale bazei de date MongoDB. Dacă se dorește implementarea de logică la nivelul bazei de date acest lucru nu este posibil deoarece nu se pot crea proceduri stocate sau alte tipuri de funcții.

În elaborarea lucrării am utilizat MongoDB împreună cu Hyperopt pentru a putea realiza optimizarea de hiperparametri. Baza de date a fost instalată pe o stație de lucru folosită doar pentru acest lucru, pe care nu s-a rulat nicio antrenare. Pentru a putea analiza experimentele a fost folosit și un script de Python care a reușit să se conecteze la baza de date, să citească datele experimentelor, să organizeze într-un mod corelat valorile hiperparametrilor cu eroarea de validare obținută și să le afișeze pe un grafic.

4.5 CUDA

Este o platformă de calcul paralel dezvoltată pentru plăcile grafice Nvidia. Rolul acesteia este de a permite dezvoltatorilor software să accelereze execuția programelor intensiv-computaționale prin utilizarea plăcilor grafice.

Combinația dintre plăcile grafice Nvidia și CUDA a devenit folosită în multe domenii care au nevoie de performanțe înalte de calcul, cum ar fi învățarea automată, data science, analiza numerică, imagistică medicală sau bioinformatică. [26]

În antrenarea rețelelor neurale profunde este de asemenea folosită CUDA împreună cu librării specializate ce oferă suport pentru această platformă, cum sunt Keras si Tensorflow. În acest caz specific este dezvoltată o librărie specială, cuDNN, care este specializată în realizarea operațiunilor matematice specifice rețelelor neurale profunde.

Pe lângă cuDNN, CUDA oferă și alte librării pentru rețele profunde, precum TensorRT și DeepStream. De asemenea există librării CUDA specializate și pentru alte domenii, cum sunt cele pentru algebră liniară, matematică, procesare de semnale și algoritmi paraleli.

Dacă nu se dorește utilizarea unor librării, există posibilitatea programării operațiunilor realizate pe GPU utilizând CUDA la nivelul cel mai scăzut prin limbajul C++.

CAPITOLUL 5

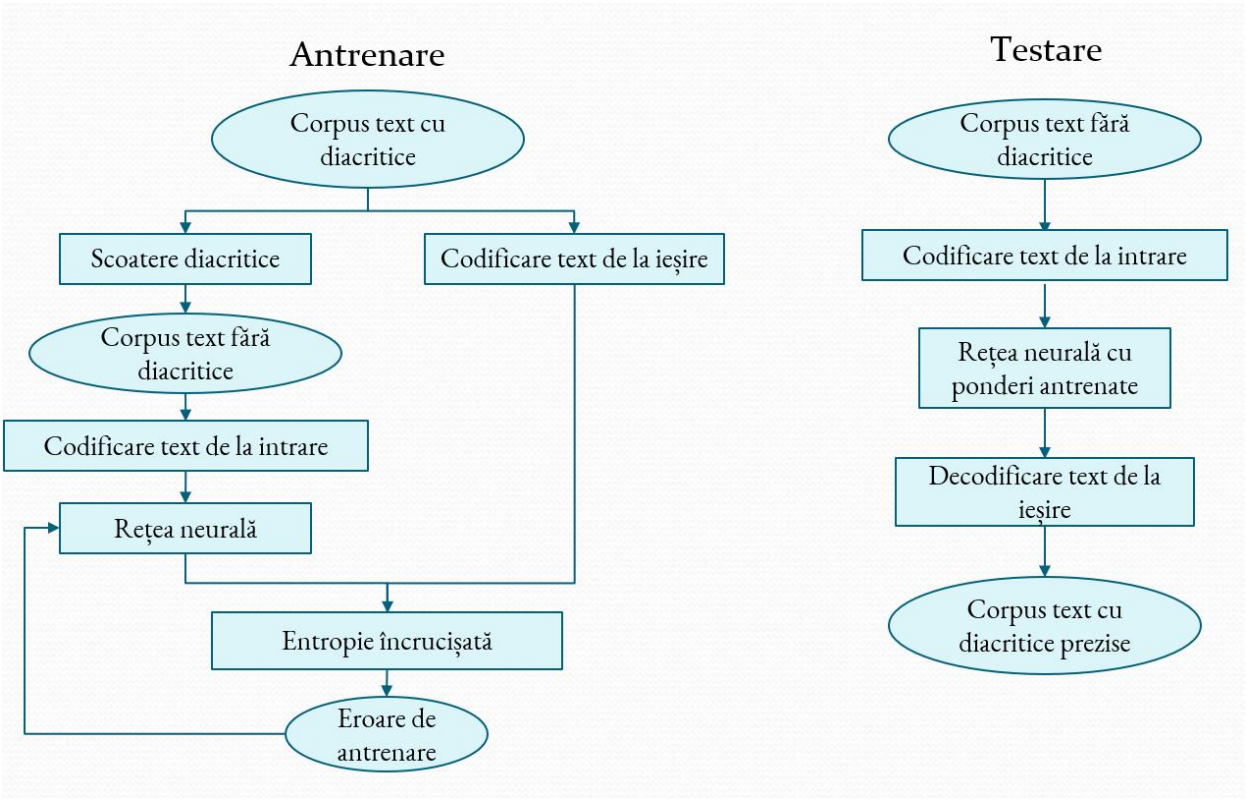
METODE PROPUSE

Ideea principală este folosirea unui corpus de text cu diacritice pentru antrenarea unei RNR. Aceasta trebuie să învețe ca, pentru propozițiile de la intrare fără diacritice, să scoată aceleași propoziții, cu diacritice.

Rolul RNR este acela de a procesa secvența de text, la nivel de cuvânt, caracter sau ambele, și de a extrage informații utile despre fiecare parte a vorbirii în parte. Aceste informații vor fi trimise mai departe către un strat neural dens de ieșire cu rol de clasificare.

La antrenare, fiecărei propoziții din corpusul de text de antrenare îi este făcută o copie, căreia îi sunt scoase diacriticele și sunt puse într-un alt corpus, care va folosit la intrarea rețelei, iar propozițiile din corpusul cu diacritice vor fi folosite la ieșire, drept etichete. Intrarea rețelei este astfel formată din propoziții fără diacritice, codificate la nivel de cuvânt sau la nivel de caracter, iar ieșirea este un clasificator care prezice forma corectă a cuvintelor sau caracterelor de la intrare. Pentru antrenarea rețelei este calculată eroarea dintre preziceri și diacriticele corecte din corpus. De asemenea și etichetele de antrenare sunt codificate, astfel încât eroarea este calculată relativ la codificare.

La testare sunt disponibile doar propoziții fără diacritice. Ele sunt codificate și trecute prin rețeaua cu ponderi antrenate, care va prezice, pentru fiecare caracter, o listă de probabilități de apartenență la fiecare clasă din cele posibile. Se aplică funcția argmax pentru a afla poziția din listă la care se



Figură 5.1 - Metoda utilizată

află probabilitatea cea mai mare. Acea poziție reprezintă codificarea de la ieșire. Mai departe ea este decodificată pentru a se obține caracterul prezis de rețea.

Metoda este descrisă grafic în Figură 5.1.

5.1 ARHITECTURA LA NIVEL DE CARACTER

Fiecare exemplu de antrenare este codificat la nivel de caracter, prin asocierea câte unui număr întreg fiecăruia, folosind o lista de caractere posibile la intrare, formată din litere fără diacritice, cifre și simbolurile spațiu, cratimă și sfârșit de linie.

Mai departe caracterele trec prin stratul de mapare la reprezentare vectorială, cu ponderi antrenabile pe măsură ce rețeaua se antrenează. Vectorii asociați caracterelor intră în RNR propriu zisă, fiecare vector fiind corespunzător unei celule recurente. Astfel, în mod indirect, fiecărui caracter de la intrare i se asociază o celulă recurentă.

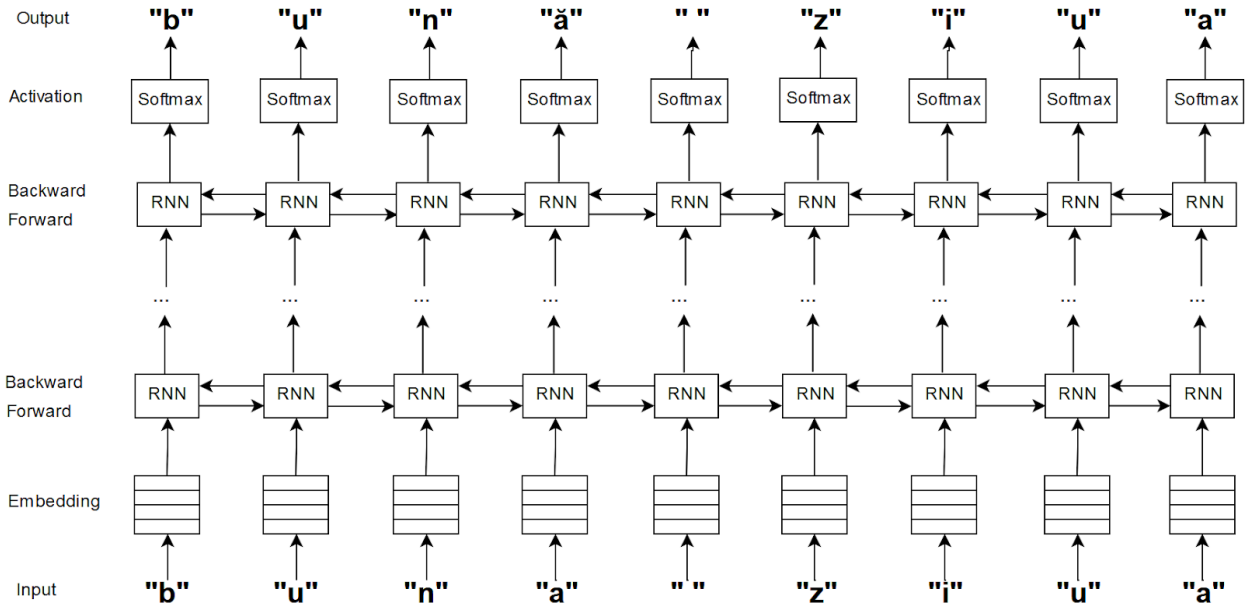
După trecerea prin RNR, informația ajunge într-un strat de tip dense, a cărui activare este funcția softmax. Vor fi utilizate două metode de codificare a datelor de la ieșire:

i. Asocierea fiecărui caracter un cod care corespunde indicelui său în lista de caractere posibile la ieșire. Deoarece la ieșire pot fi 44 de caractere (litere fără diacritice și cu diacritice, cifre, caractere speciale spațiu, newline și cratimă), atunci stratul de ieșire va realiza clasificarea în 44 de clase. Acest lucru poate duce la existența posibilității ca un caracter să fie înlocuit cu un alt caracter greșit, chiar și în cazul caracterelor ce nu pot avea diacritice.
ii. Împărțirea caracterelor de la ieșire în 3 clase corespunzătoare.
    a. 0 – nu e nevoie de diacritică
    b. 1 – „î”, „ș” , „ț”, „â”
    c. 2 – „ă”

Acest tip de împărțire duce la nevoia existenței unui pas suplimentar de preprocesare și postprocesare a datelor, dar asigură consistența caracterelor. Dimensiunea stratului de ieșire va fi 3.

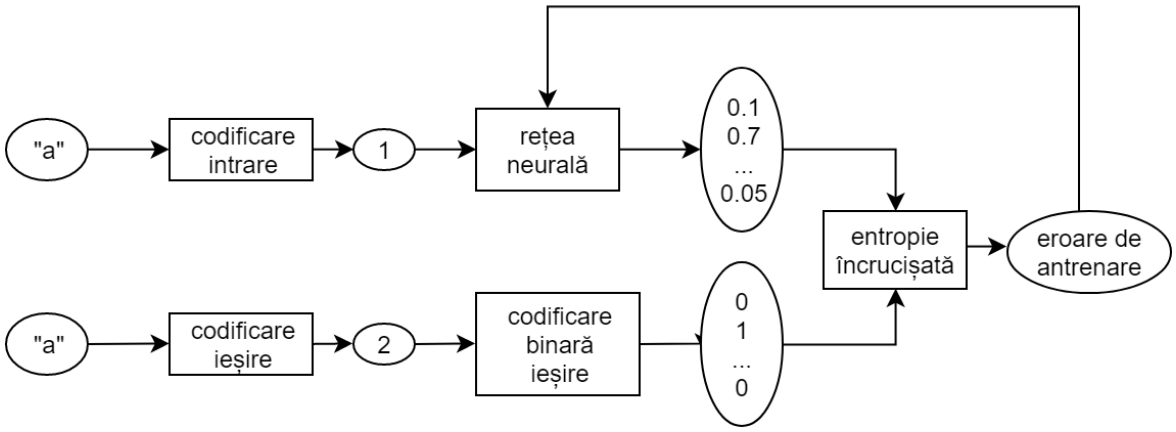
Față de codificarea caracterelor de la intrare, cele de la ieșire sunt ulterior convertite în formatul binar de vector-coloană cu 1 pe poziția corespunzătoare clasei, pentru a putea calcula entropia încrucișată, fiind vorba de o problemă de clasificare în mai multe clase.

O privire de ansamblu asupra acestei arhitecturi poate fi observată în Figură 5.2. Rețeaua RNR este descrisă la modul general, deoarece se pot face modificări precum modificarea tipului de celulă în LSTM sau GRU sau stivuirea mai multor straturi.

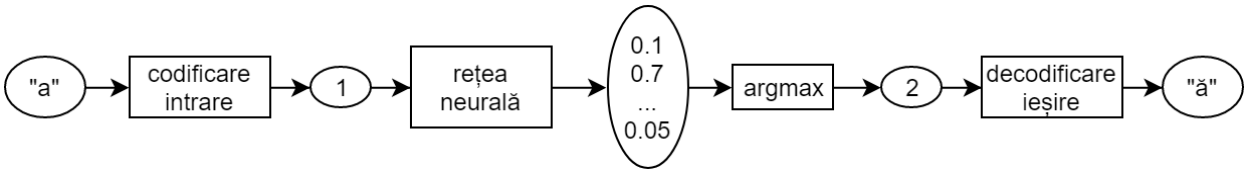


Figură 5.2 - Arhitectura la nivel de caracter

În Figură 5.3 este prezentată îndeaproape calea datelor și felul în care acestea se modifică. Se observă felul în care rețeaua învață să restaureze diacritica literei „ă” pentru litera „a”, iar în Figură 5.4 este descrisă modalitatea în care, în cadrul etapei de testare, rețeaua restaurează diacritica. Pentru o ușoară înțelegere a exemplului, nu se iau în considerare și restul caracterelor din exemplu și se face presupunerea că maparea literei „a” este 1 și literei „ă” este 2.



Figură 5.3 - Preprocesarea și metodologia de antrenare



Figură 5.4 - Preprocesarea și metodologia de testare

5.2 ARHITECTURA LA NIVEL DE CUVÂNT

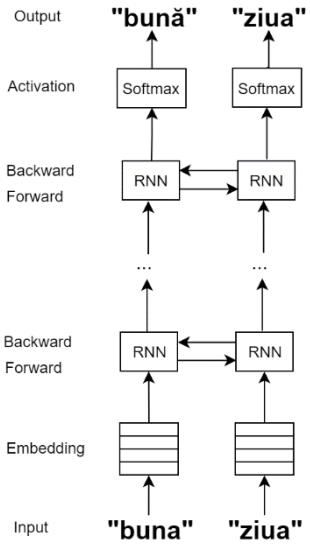
Fiecare exemplu de antrenare este împărțit pe cuvinte. În stratul de intrare al rețelei se vor găsi astfel cuvinte, fiecare în parte fiind codificat după o anumită regulă astfel încât să existe o relație de corespondență unu la unu.

Codificările cuvintelor, odată intrate în rețea, sunt transformate în reprezentarea lor vectorială, care este învățată pe măsură ce învață și rețeaua. Mai departe vectorii sunt trecuți prin RNR. Fiecare vector ce este asociat unui cuvânt va corespunde unei celule recurente. Ieșirile celulelor recurente vor intra într-un strat de tip dens cu activare softmax pentru clasificare. Etichetele de antrenare sunt cuvintele corespunzătoare cu diacritice, codificate după o anumită regulă și convertite binar cu 1 pe poziția corespunzătoare codificării fiecărui cuvânt. Astfel dimensiunea stratului de ieșire va fi egală cu numărul de clase posibile, adică dimensiunea vocabularului, care este foarte mare. Din acest motiv este necesară limitarea vocabularului la un număr suficient de mic cât să nu se depășească memoria disponibilă și nici antrenarea să nu dureze foarte mult, dar și suficient de mare încât acuratețea să fie acceptabilă.

Codificarea cuvintelor se face după numărul de apariții ale acestora în corpus. Cuvintele sunt ordonate după numărul de apariții descrescător, iar codificarea va fi poziția în clasament. Cu alte cuvinte, cuvintele cele mai frecvente vor avea valorile cele mai mici, iar cel mai frecvent cuvânt ar avea corespunzător numărul natural 1.

Motivarea acestei alegeri este nevoia existenței unei mapări facilă unu-la-unu a cuvintelor la numere, dar și posibilitatea limitării vocabularului, prin respingerea cuvintelor care au un număr asociat mai mare decât un prag. Astfel se păstrează informația cea mai relevantă.

După parcurgerea textului, numărarea aparițiilor, sortarea cuvintelor și asocierea unui număr fiecărui cuvânt rezultă un dicționar, ale cărui chei sunt cuvintele și valori sunt codificările. Este creat și un dicționar invers, ale cărui chei sunt codificările și valori sunt cuvintele.



Figură 5.5 – Arhitectura la nivel de cuvânt

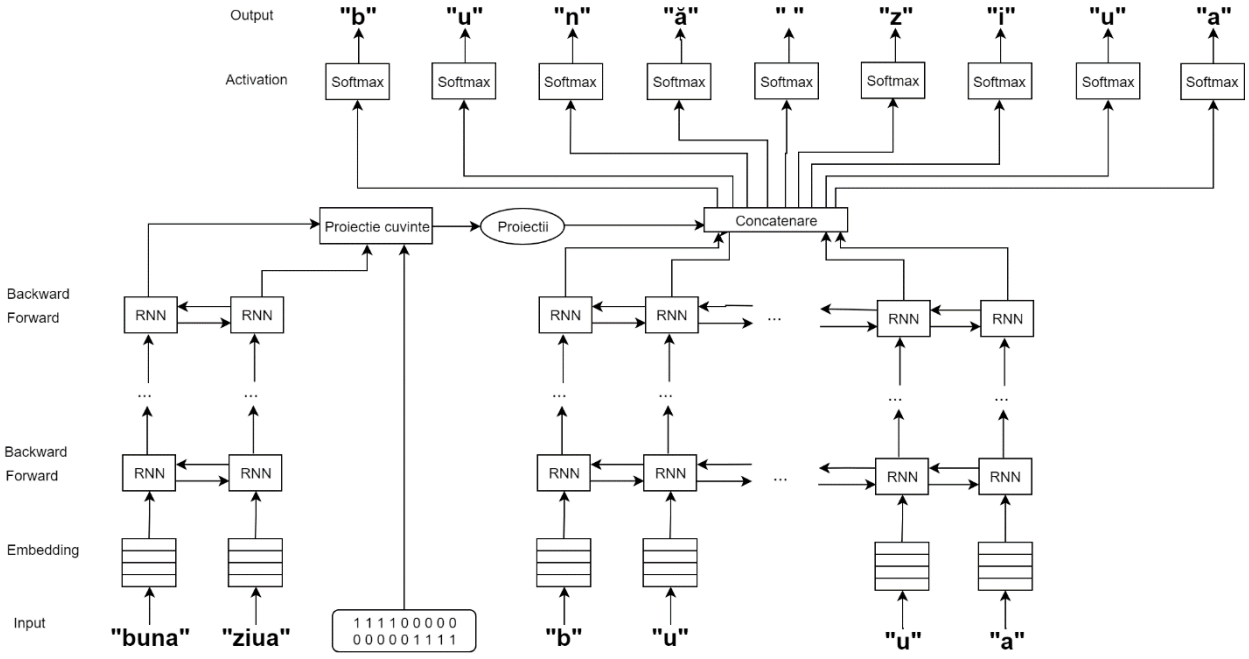
Acest procedeu este făcut întâi pentru cuvintele fără diacritice apoi pentru cuvintele cu diacritice. Dicționarul cuvintelor cu diacritice este utilizat la ieșirea sistemului, pentru a decodifica predicțiile.

5.3 ARHITECTURA HIBRID

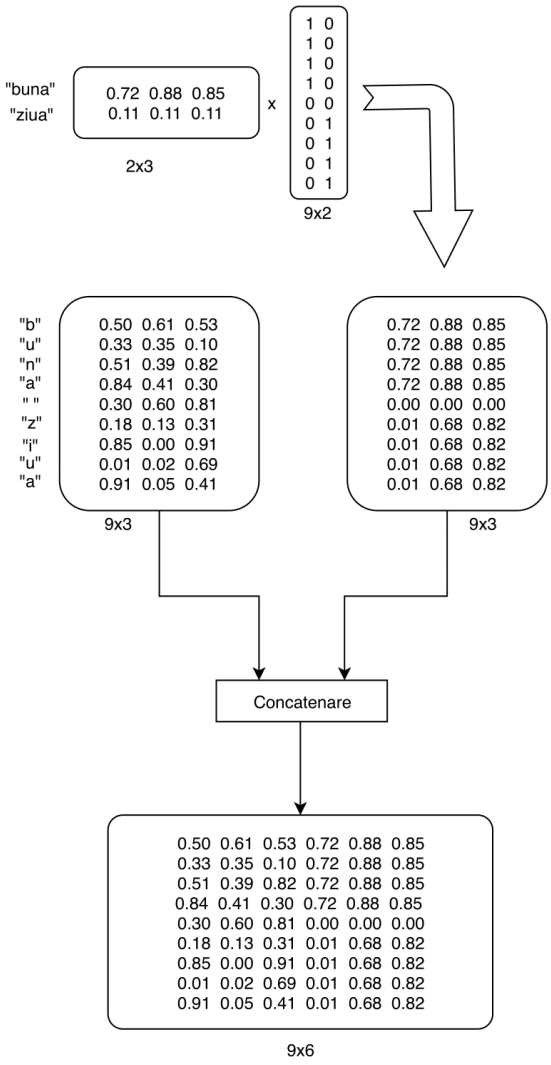
La intrarea rețelei vor fi, pentru fiecare exemplu, codificările cuvintelor din care este compus dar și a caracterelor și o hartă de proiecție a cuvintelor pe spațiul caracterelor. Atât codificările cuvintelor, cât și ale caracterelor, vor fi transformate în reprezentările lor vectoriale. Acestea sunt mai departe puse fiecare la intrarea unor RNR. Ieșirile RNR sunt aduse la aceleași dimensiuni utilizând harta de proiecție a cuvintelor, apoi concatenate. După concatenare se ajunge în stratul final, un strat dens cu activare softmax. La ieșirea rețelei se vor afla etichetele de antrenare, exemplele cu diacritice, la nivel de caracter.

În Figură 5.6 se poate observa caracterul hibrid al modelului. Pe de o parte există o RNR la nivel de cuvânt, iar pe de altă parte este RNR la nivel de caracter. Ieșirile rețelei la nivel de cuvânt sunt proiectate pe spațiul caracterelor pentru a putea fi concatenate cu ieșirile rețelei la nivel de caracter, iar predicția să se facă având în vedere atât informația despre caracter, cât și despre cuvânt.

Harta de proiecție este o metodă de legătură între cuvinte și caractere utilizată în [4]. Ea este utilizată deoarece în momentul în care cuvintele sunt codificate și apoi transformate în vectori se pierde informația legată de caracterele ce compun cuvântul. Metoda de păstrare a informației constă în formarea propoziției la nivel de caracter, iar în locul informației despre fiecare caracter este pusă informația despre cuvântul din care acesta face parte. Informația despre cuvânt poate fi fie reprezentarea vectorială a acestuia, fie rezultatele trecerii cuvintelor prin RNR. Pentru a se putea obține acest lucru, harta de proiecție are următorul format. Are numărul de linii egal cu numărul de cuvinte și numărul de coloane egal cu numărul de caractere. Valorile ei sunt 1, pe fiecare rând, pe coloane corespunzătoare pozițiilor caracterelor ce formează cuvântul din propoziție și 0 în rest. Un exemplu de astfel de hartă, corespunzător propoziției „bună ziua”, poate fi observată în Figură 5.7. Codul utilizat pentru generarea hărții de proiecție și cel care face proiecția și concatenarea sunt bazate pe [4].



Figură 5.6 - Privire de ansamblu asupra modelului hibrid



Figură 5.7 - Proiectarea informațiilor

În Figură 5.7 este prezentat, în mod concret, felul în care informația despre cuvinte este proiectată în spațiul caracterelor și apoi cum se face concatenarea datelor. Pentru simplitate s-a considerat o reprezentare vectorială de dimensiune 3, atât pentru cuvinte, cât și pentru caractere.

În prima fază vectorii corespunzători cuvintelor sunt stocați într-o matrice cu numărul de linii egal cu numărul de cuvinte și numărul de coloane egal cu dimensiunea vectorului de reprezentare. Această matrice este înmulțită cu transpusa hărții de proiecție. Se obține o matrice cu un număr de linii egal cu numărul de caractere și numărul de coloane egal cu dimensiunea reprezentării. Forma aceasta este identică formei matricei caracterelor, iar pe fiecare rând corespunzător unui caracter este pusă reprezentarea vectorială a cuvântului din care face parte. Matricea aceasta este concatenată cu matricea reprezentărilor vectoriale a caracterelor și se obține o matrice cu număr de linii egal cu numărul de caractere și număr de coloane egal cu dimensiunea reprezentării vectoriale a caracterelor adunată cu dimensiunea reprezentării vectoriale a cuvintelor.

Avantajul acestei matrice este faptul că ea conține, pentru fiecare caracter în parte, și informație despre cuvântul din care face parte. Un alt avantaj este faptul că, deoarece numărul de rânduri este egal cu numărul de caractere din propoziție, se poate face predicția la nivel de caracter, care este mult mai puțin costisitoare decât cea la nivel de cuvânt. Codificările de la intrare utilizate sunt cele din modelele la nivel de cuvânt și caracter. Codificarea de ieșire se va face la nivel de caracter folosind metodele prezentate în capitolul 5.1.

CAPITOLUL 6

DETALII DE IMPLEMENTARE

6.1 DEFINIRE MODEL

Definirea arhitecturii rețelei neurale a fost realizată utilizând librăria Keras. Procesul constă în construirea modelului prin adăugarea de straturi și crearea de legături între ele. Straturile sunt reprezentate de clase din Keras care primesc în constructor atribute precum dimensiunea lor și funcția de activare. Utilizând paradigma de programare funcțională, straturile vechi sunt date ca argumente straturilor noi pentru a se crea legătura dintre ele. La sfârșit un obiect de tip model este creat folosind straturile de intrare și de ieșire.

```
1.  def get_model(
2.      seq_len=16,
3.      emb_dim=64,
4.      lstm_dim=64,
5.      lstm_dropout=0.2,
6.      recurrent_dropout=0.2,
7.      output_dropout=0.2,
8.  ):
9.      inp = Input(shape=(None,))
10.     emb = Embedding(input_dim=len(CHARS_INP), output_dim=emb_dim)(inp)
```

```
    lstm = LSTM(
        lstm_dim,
        return_sequences=True,
        input_shape=(None, 32),
        dropout=lstm_dropout,
        recurrent_dropout=recurrent_dropout,
    )
    bilstm = Dropout(output_dropout)(Bidirectional(lstm)(emb))
    out = Dense(len(CHARS_OUT), activation="softmax")(bilstm)
    model = Model(inp, out)
    return model
```

După obținerea structurii modelului, acesta trebuie configurat cu funcția cost și algoritmul de optimizare utilizate, utilizând funcția *compile*.

```
model = get_model(seq_len, emb_dim, lstm_dim, lstm_dropout, recurrent_dropout, output_dropout)
model.compile(loss="categorical_crossentropy", optimizer=Adam(lr=initial_lr))
```

Funcția cost și algoritmul de optimizare sunt și ele definite în Keras și pot fi folosite cu valori implicite sau se pot specifica unele specifice. De exemplu, în cod este folosit algoritmul de optimizare Adam cu o rată de învățare inițială specificată în constructor. În mod implicit aceasta este 0.001.

6.2 ANTRENARE

Odată ce modelul este definit și configurat, poate începe antrenarea acestuia. În mod normal se folosește funcția *fit*, care primește ca argumente setul de date de antrenare, numărul de epoci, setul de date de validare și funcții ce se pot executa la diferite momente ale antrenării.

În situația de față este folosită funcția *fit_on_generator*. Aceasta va primi, în locul setului de date de antrenare, un obiect de tip generator care se ocupă de selectarea și aducerea la intrarea rețelei a anumitor exemple de antrenare. Generatorul aduce la intrare exemplele la nivel de loturi, iar funcția *fit_on_generator* va mai primi ca argument numărul de loturi utilizate într-o epocă. Trecerea unui lot se numește un pas de antrenare. În rest se păstrează aceleași argumente ca la funcția *fit*.

```
checkpoint = ModelCheckpoint(model_file_name, verbose=1, save_best_only=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=reduce_lr_patience, verbose=1)
csv_logger = CustomCsv("log_{}.csv".format(datetime.datetime.now().strftime('%Y_%m_%d_%H_%M')), append=True, separator=';')
early_stopping = EarlyStopping(patience=early_stopping_patience)
tensorboard = TensorBoard(log_dir="logs/{}".format(datetime.datetime.now().strftime('%Y_%m_%d_%H_%M')))

generator = Generator(X, Y, batch_size=128, steps=steps_per_epoch)
model.fit_generator(
    generator, steps_per_epoch=steps_per_epoch, epochs=128, validation_data=(X_val, Y_val),
    callbacks=[checkpoint, early_stopping, reduce_lr, tensorboard, csv_logger]
)
```

Funcțiile de tip *callback*, care se execută în anumite momente ale antrenării, sunt deja definite în Keras, însă pot fi și definite de utilizator. În cod se observă cum sunt definite acestea, folosind parametri doriți de programator. Funcțiile care reduc rata de învățare și care opresc antrenarea au valori ale răbdării diferite, care sunt citite de la tastatură în momentul pornirii antrenării.

Obiectul generator este instanțiat din clasa *Generator*, primind ca argumente setul de date de antrenare, dimensiunea lotului și numărul de loturi (numărul de pași) dintr-o epocă. Clasa

Generator este moștenită din clasa abstractă *Sequence* existentă în Keras. Sunt necesare implementările metodelor *__len__* și *__getitem__* pentru a putea selecta loturi la fiecare pas. Metoda *on_epoch_end* se execută la terminarea fiecărei epoci de antrenare și are rolul de a ordona aleatoriu indicii exemplelor din setul de date.

```
1.  class Generator(Sequence):
2.      def __init__(self, x_set, y_set, batch_size=256, steps=None):
3.          self.x, self.y = x_set, y_set
4.          self.batch_size = batch_size
5.          self.indices = np.arange(self.x.shape[0])
6.          self.steps = steps
7.
8.      def __len__(self):
9.          return self.steps or math.ceil(self.x.shape[0] / self.batch_size)
10.
11.     def __getitem__(self, idx):
12.         inds = self.indices[idx * self.batch_size:(idx + 1) * self.batch_size]
13.         batch_x = self.x[inds]
14.         batch_y = to_categorical(self.y[inds], num_classes=len(CHARS_OUT))
15.         return batch_x, batch_y
16.
17.     def on_epoch_end(self):
18.         np.random.shuffle(self.indices)
```

Pornirea unei antrenări se face din terminal folosind comanda:

```
python src/main.py --todo train --data train_file –model model_name --
steps_per_epoch 256 --reduce_lr_patience 3 --early_stopping_patience 6
```

Argumentele pot fi modificate în funcție de ceea ce se dorește. Un caz special este cel în care *model_name* corespunde unui fișier ce conține un model deja antrenat. Atunci acel model este încărcat și este continuată antrenarea lui.

6.3 PREPROCESARE DATE

Propozițiile din setul de antrenare sunt de lungime diferită din punct de vedere al numărului de caractere. Deși RNR permite antrenarea secvențelor cu lungimi variabile, acest lucru duce la o durată mai mare de antrenare, deoarece nu se mai poate realiza paralelizarea operațiilor. Astfel se dorește obținerea unor secvențe de lungimi egale.

O primă abordare ar fi aducerea tuturor secvențelor la lungimea secvenței cele mai mari, prin umplerea lor cu caracterul spațiu. Această metodă însă creează multă redundanță și duce la un timp de antrenare foarte crescut.

O altă abordare ar fi lipirea tuturor exemplelor într-un singur șir de caractere foarte mare și apoi împărțirea acestuia în bucăți egale de caractere. În urma unei optimizări de hiperparametri s-a obținut valoarea optimă a lungimi ca fiind 256 de caractere. Chiar dacă această metodă funcționează iar modelul învață să restaureze diacritice, apar probleme precum faptul că unele cuvinte sunt tăiate, iar în alte situații sunt create exemple formate din două propoziții diferite și sunt alăturate cuvinte care nu au legătură.

Cea mai bună metodă este una care prelucrează într-un mod mai inteligent și adaptiv propozițiile. Se va urmări obținerea unor propoziții tot de lungime de 256 de caractere. Metoda sparge propozițiile lungi fără a tăia în interiorul cuvintelor, filtrează propozițiile cu o lungime mai mică decât un prag și apoi grupează propozițiile de lungime mică astfel încât propozițiile rezultate să aibă lungimea de 256 caractere sau mai puțin. Mai departe propozițiile care au mai puțin de 256 de caractere sunt umplute cu caracterul spațiu.

Exemplu, preprocesarea primelor 10 exemple din setul de antrenare având ca obiectiv propoziții de 40 de caractere, fără filtrare după lungime minimă. În prima bucată de cod se pot observa propozițiile inițiale, alături de lungimile lor.

```
{
    32: 'bună seara doamnelor şi domnilor',
    49: 'o ediţie specială o sută la sută în această seară',
    20: 'bună seara bun venit',
    25: 'ca să-l schimbe pe domnul',
    7: 'nu ştiu',
    64: 'discutăm în special despre congresul partidului naţional liberal',
    73: 'invitaţii mei din această seară sunt domnii ion cristoiu şi dan turturică',
    42: 'crin antonescu reales la al doilea congres',
    61: 'un congres fără istorie domnule cristoiu previzibil plicticos',
    72: 'până la urmă al domnului crin antonescu a fost să desfiinţeze acest post'
}
```

După spargerea propozițiilor, toate au lungimea mai mică sau egală cu 40. Nicio propoziție nu a fost împărțită prin tăierea în interiorul unui cuvânt. Acest lucru este realizat folosind funcția *wrap* a librăriei *textrwrap*. De observat este faptul că s-au obținut și propoziții de lungimi egale.

```
{
    32: ['bună seara doamnelor şi domnilor', 'a fost să desfiinţeze acest post'],
    34: ['crin antonescu reales la al doilea'],
    35: ['o ediţie specială o sută la sută în'],
    36: ['discutăm în special despre congresul', 'invitaţii mei din această seară sunt
', 'domnii ion cristoiu şi dan turturică'],
    7: ['congres', 'nu ştiu'],
    39: ['până la urmă al domnului crin antonescu'],
    40: ['un congres fără istorie domnule cristoiu'],
    25: ['ca să-l schimbe pe domnul'],
    27: ['partidului naţional liberal'],
    13: ['această seară'],
    20: ['bună seara bun venit','previzibil plicticos']
}
```

După gruparea propozițiilor s-a redus numărul de propoziții de lungime mică și a crescut numărul celor de lungime apropiată de 40. Concatenarea propozițiilor se face folosind caracterul de linie nouă cu scopul ca rețeaua să îl învețe drept marcaj pentru sfârșitul unei idei și să nu creeze corelații între caracterele dinaintea sa și caracterele de după acesta.

```
{
    32: ['a fost să desfiinţeze acest post', 'bună seara doamnelor şi domnilor'],
    34: ['crin antonescu reales la al doilea'],
    35: ['o ediţie specială o sută la sută în', 'ca să-
l schimbe pe domnul \n congres'],
    36: ['domnii ion cristoiu şi dan turturică', 'invitaţii mei din această seară sunt
', 'discutăm în special despre congresul', 'previzibil plicticos \n această seară'],
    37: ['partidului naţional liberal \n nu ştiu'],
    39: ['până la urmă al domnului crin antonescu'],
    40: ['un congres fără istorie domnule cristoiu'],
    20: ['bună seara bun venit']
}
```

În cadrul modelului de tip hibrid a fost necesar ca pentru fiecare exemplu de antrenare să se obțină harta de proiecție a cuvintelor în spațiul caracterelor. Funcția care face acest lucru a fost preluată și adaptată din [4].

6.4 Optimizarea hiperparametrilor

Optimizarea de hiperparametri s-a realizat într-un mod distribuit folosind librăria Hyperopt și mai multe stații de lucru, una master responsabilă de definirea experimentelor, una responsabilă de găzduirea bazei de date și mai multe lucrătoare, responsabile pentru antrenament. Această organizare este descrisă și în Figură 4.1.

Stația master se conectează la baza de date folosind datele de acces și specificând baza de date, precum și colecția în care dorește să scrie. Baza de date trebuie specificată deoarece fiecare utilizator are acces doar la anumite baze de date. De asemenea trebuie specificată și o cheie prin intermediul căreia se vor identifica experimentele optimizării curente. Aceasta poate fi utilă în situația în care se dorește reluarea optimizării după ce a fost oprită.

Apoi stația pornește optimizarea, definind funcția obiectiv, spațiul de căutare, locația de stocare a experimentelor care va fi în acest caz baza de date, algoritmul de sugestie a experimentelor și numărul maxim de experimente. Funcția *fmin* întoarce variabilele de control a celui mai bun experiment, după terminarea acestora.

Crearea unui experiment constă în inserarea unui document ce conține variabilele de control, identificatorul, cheia de identificare, marcă de timp și statusul *new*.

```
1.  trials = MongoTrials("mongo://name:pass@ip:port/hyperopt/jobs?authSource=hyperopt", ex
    p_key="optimization-1")
2.  best = hyperopt.fmin(
3.      objective.to_min,
4.      space=[
5.          hyperopt.hp.uniform("lr", 0.00001, 0.01),
6.          hyperopt.hp.uniform("lstm_dropout", 0.1, 0.5),
7.          hyperopt.hp.uniform("recurrent_dropout", 0.1, 0.5),
8.          hyperopt.hp.uniform("output_dropout", 0.1, 0.5),
9.      ],
10.     trials=trials,
11.     algo=hyperopt.tpe.suggest,
12.     max_evals=40,
13. )
14. print(best)
```

Stațiile lucrătoare pornesc un script din terminal care se conectează la baza de date și verifică dacă există experimente care nu sunt realizate, iar în cazul în care găsesc și le atribuie, prin modificarea statusului și adăugarea unei alte mărci de timp. Scriptul pornit se găsește în fișierul de instalare a librăriei Hyperopt și, pentru a funcționa corect, este necesară prezența scriptului ce conține funcția obiectiv sau a unei legături către acesta. Comanda de pornire a scriptului este următoarea.

```
./hyperopt-mongo-worker --mongo="mongo://name:pass@ip:port/hyperopt" --poll-interval=0.1
```

Funcția obiectiv este *to_min* iar aceasta va primi ca argument hiperparametrii cu care va face antrenarea, sub forma unui tuplu. După extragerea acestora din tuplu, funcția va defini modelul, va încărca setul de date și va realiza antrenarea. La sfârșit este necesar ca funcția să întoarcă o metrică de evaluare și statusul experimentului.

Scriptul va reveni în baza de date și va modifica documentul asociat experimentului prin actualizarea statusului în *ok* și va pune valoarea metricii obținute.

```
1.  def to_min(hyperparams=None, d=None):
2.
3.      print("Starting a new objective", hyperparams)
4.      (lr, lstm_dropout, recurrent_dropout, output_dropout) = hyperparams
5.      # ...
6.      return {
```

```
7.          "loss": train_history.history["val_loss"][-1],
8.          "status": hyperopt.STATUS_OK,
9.      }
```

După terminarea unui experiment, calculator master va crea altul cu statusul new, iar calculatoarele lucrătoare vor verifica periodic, la o perioadă specificată în momentul pornirii scriptului, baza de date pentru a identifica noi experimente.

6.5 EVALUAREA PERFORMANȚELOR

Pentru a evalua performanțele se folosește un set de date separat, care conține doar cuvinte fără diacritice, și a cărui acuratețe este măsurată prin intermediul unei interfețe web.

Restaurarea diacriticelor se face folosind un model cu ponderi antrenate, care este încărcat în funcția de testare. Setul de date de evaluare este împărțit în propoziții și fiecare propoziție va intra una câte una în rețea. Rețeaua va da probabilitățile de apariție a cuvintelor/literelor cu diacritice, iar funcția de testare va decodifica ieșirea rețelei și va obține propoziția cu diacritice. Aceasta este pusă într-un corpus de text împreună cu predicțiile rețelei obținute pentru celelalte propoziții. Corpusul astfel obținut este setul de date de evaluare cu diacritice restaurate.

Pentru rularea scriptului de testare se execută în terminal comanda:

```
python src/main.py --todo test --model model_name --out_file output_file
```

Se pot alege modelul ce urmează să fie evaluat și numele fișierului în care să fie pus textul cu diacritice restaurate. Calea către fișierul utilizat pentru evaluare nu poate fi modificată decât în cod.

Mai departe evaluarea acurateței se va face utilizând o interfață web dezvoltată de laboratorul SpeeD. Aceasta oferă posibilitatea încărcării corpusului de text cu diacritice restaurate și calculează acuratețea obținută la nivel de cuvânt, după următoarea metrică:

```
1.  def word_accuracy(true_words, pred_words):
2.      return sum(t == p for t, p in zip(true_words, pred_words)) / len(true_words)
```

Evaluarea se face în cadrul interfeței și nu folosind o funcție proprie deoarece doar pe serverul la care este legată interfața se găsește corpusul de text cu diacritice corecte, dar și deoarece rezultatele pot fi denumite și sunt salvate în mod automat pentru ca apoi să fie observate cu ușurință.

Pe lângă acuratețe, au fost dezvoltate scripturi în Python pentru calculul altor metrici de performanță, precizia, reamintirea și scorul F. Calculul acestora se face pentru fiecare literă cu diacritică în parte. Tot cu scopul evaluării performanțelor a fost dezvoltat un script care, pentru fiecare cuvânt care poate avea diacritice, notează toate formele luate de acel cuvânt dar cu diacritice diferite, împreună cu numărul de apariții ale fiecărei forme și numărul de forme greșite. Un alt script a fost dezvoltat în vederea comparării performanțelor mai multor modele între ele dar și cu diacriticele corecte.

6.6 COLECTAREA CORPUSULUI DE TEXT

Scopul este colectarea unui corpus de text cu diacritice corecte folosind o aplicație de tip web-crawler care să ruleze periodic și să extragă articolele noi apărute pe site-uri de știri în limba română. A fost dezvoltată în acest sens o aplicație în Java care să e ocupe de această sarcină și, cu ajutorul planificatorului *cron* din Linux, ea rulează periodic, odată la 4 ore.

Aplicația funcționează prin conectarea la feed-ul RSS al site-urilor, parcurgerea acestora, obținerea titlurilor și a link-urilor știrilor noi și accesarea paginilor. Apoi se utilizează o librărie

pentru a prelucra codul HTML al paginii și a extrage textul știrii, care va fi mai apoi scris într-un fișier denumit după titlul ei. Librăria utilizată se numește Snacktory [27] și se bazează pe algoritmul utilizat de o extensie pentru web browsere, Readability, menită să afișeze doar esențialul unui articol online în momentul accesării acestuia.

În total, sunt utilizate 24 de surse de text, reprezentate de site-uri românești de știri și de publicații online. Din acest motiv aplicația a fost proiectată să ruleze în paralel, extragerea știrilor din fiecare sursă făcându-se pe câte un fir de execuție separat.

Odată strânse textele articolelor, ele sunt mai departe procesate în 2 etape.

Întâi este utilizat un script în Python care se ocupă de filtrarea și organizarea fișierelor extrase. Filtrarea este necesară deoarece algoritmul de extragere a textului articolului nu funcționează întotdeauna corect sau este extras corect dar textul nu conține diacritice.

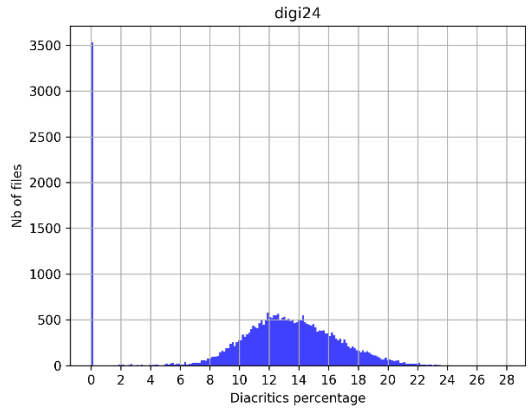
Astfel sunt realizate întâi filtrări după fișiere eronate, care sunt considerate cele cu conținut identic cu a altor fișiere și cele de dimensiune redusă a octeților. Apoi este realizată filtrarea după procentul de litere ce acceptă diacritice care au diacritice. Sunt păstrate doar fișierele ale căror procent depășește un prag, iar celelalte sunt mutate într-o locație separată. De asemenea fișierele fără diacritice pot fi și ele considerate eronate. Pragul procentului de diacritice și pragul dimensiunii fișierului în octeți sunt parametri ce pot fi dați ca argument funcției de curățare. După ce se realizează filtrarea, scriptul unește textele articolelor într-un singur fișier, câte unul pentru fiecare sursă.

În continuare va fi reprezentată grafic histograma numărului de fișiere în funcție de procentul de diacritice pentru sursa de știri digi24.

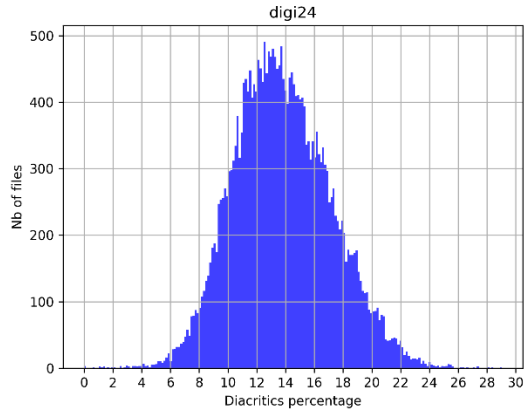
Figură 6.1 prezintă situația inițială, în care se poate observa diferența majoră dintre numărul de fișiere care au procentul de diacritice 0 și numărul de fișiere care au procentul de diacritice diferit de 0.

În Figură 6.2 este prezentată histograma după primul pas, cel de curățare a fișierelor care se repetă și care au dimensiunea în octeți mai mică decât un prag. Aceasta a realizat eliminarea în totalitate a fișierelor cu procent 0 diacritice.

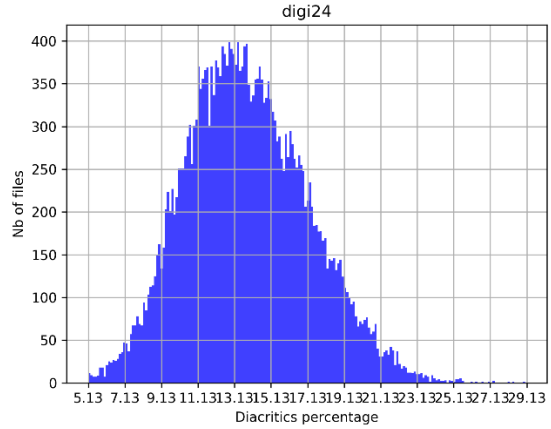
În Figură 6.3 se observă influența filtrării fișierelor care au procentul de diacritice mai mic de 5%.



Figură 6.1 – Histograma inițială



Figură 6.2 – Histograma după curățare



Figură 6.3 – Histograma după filtrare

Un lucru interesant de observat este faptul că histograma tinde la forma unei distribuții normale.

După terminarea curățării și organizării se trece la a doua etapă, în care se face curățarea la nivelul textului, utilizând aplicația TextCorpusCleaner dezvoltată de laboratorul SpeeD. Curățarea de data aceasta este făcută prin eliminarea caracterelor speciale, înlocuirea numerelor și a adreselor email sau web cu forma lor vorbită, eliminarea propozițiilor prea scurte, eliminarea propozițiilor care conțin puține cuvinte în limba română, folosind un vocabular, iar, în final, convertirea tuturor literelor în lowercase.

După terminarea acestui pas sunt obținute corpusurile separate pentru fiecare sursă în parte care pot apoi să fie folosite, în mod separat sau unite într-un fișier, în antrenarea unui model de limbă sau a unei rețele neurale.

Până la momentul curent s-a strâns un corpus total de 5 milioane de propoziții, compus din 108 milioane de cuvinte, dintre care 32 milioane cu diacritice. Acestea au fost numărate după realizarea tuturor pașilor de curățare și filtrare.

CAPITOLUL 7

EXPERIMENTE

7.1 CONFIGURAȚIA EXPERIMENTALĂ

7.1.1 Setul de date

Pentru a realiza antrenarea a fost nevoie de un corpus de text suficient de mare, cu diacritice corect puse. A fost folosit corpusul de text oferit de laboratorul SpeeD, cel mai mare corpus de text scris în limba română cu diacritice. Acesta este compus din mai multe corpusuri, obținute din diferite surse, cum ar fi transcrieri din emisiuni de televiziune, transcrieri din Parlamentul European sau articole extrase de pe site-uri de știri. În total corpusul conține 220 milioane de cuvinte.

Textul a fost curățat în prealabil cu ajutorul aplicației TextCorpusCleaner.

Chiar dacă în procesul de obținere a setului de date au existat și cazuri în care diacriticele nu erau puse, acestea au fost puse deoparte și, folosind sistemul de diacritice cu n-grame antrenat cu setul de date cu diacritice corecte, le-au fost puse diacriticele în vederea augmentării setului de date.

În ideea realizării unei antrenări rapide dar și cu rezultate semnificative s-a folosit doar un singur corpus dintre toate cele disponibile. Mai exact a fost ales cel compus din transcrieri ale emisiunilor de televiziune, denumit *talkshows*.

Pe lângă corpus cu diacritice pentru antrenare, acest set de date oferă și un corpus fără diacritice ce poate fi folosit pentru evaluarea performanțelor.

În Tabel 7.1 este descris corpusul de antrenare din punct vedere al numărului de cuvinte, litere și propoziții, având în vedere și prezența diacriticelor. Un cuvânt are diacritice dacă cel puțin una din litere este *ă, â, î, ș* sau *ț*. Un cuvânt acceptă diacritice dacă cel puțin una din litere este *a, i, s* sau t.

Metrica urmărită	**Număr apariții**
Cuvinte cu diacritice	13M
Cuvinte care acceptă diacritice	29M
Număr total cuvinte	39M
Litere cu diacritice	14M
Litere care acceptă diacritice	56M
Număr total litere	224M
Număr propoziții	2.6M
Număr cuvinte unice	0.16M

Tabel 7.1 - Descrierea corpusului de antrenare *talkshows.train*

Setul de date de evaluare este descris în Tabel 7.2.

Metrica urmărită	**Număr apariții**
Cuvinte care acceptă diacritice	1.6M
Număr total cuvinte	2M
Litere care acceptă diacritice	3.6M
Număr total litere	11M
Număr propoziții	0.13M
Număr cuvinte unice	0.04M

Tabel 7.2 - Descrierea corpusului de evaluare *talkshows.dev*

7.1.2 Resurse hardware și software

Pentru realizarea experimentelor au fost puse la dispoziție 5 stații de lucru cu sistem de operare Ubuntu. Una principală, cu acces la interfață grafică, de pe care s-a făcut scrierea codului, interpretarea rezultatelor dar și rularea de antrenări. La celelalte stații accesul a fost făcut de la distanță, prin SSH, și au fost folosite doar pentru rularea antrenărilor. Toate stațiile au în componența lor o placă video NVIDIA Quadro M4000, compatibilă CUDA.

Pe stația principală s-a folosit CUDA 9.2, iar pe celelalte CUDA 8. Toate stațiile au rulat antrenarea folosind un mediu virtual bazat pe Python 3.5, ce folosește Tensorflow 1.12, Keras 2.2.4 și Hyperopt 0.1.1.

7.2 REZULTATE EXPERIMENTALE

Având în vedere multitudinea de aspecte ce pot fi variate precum și faptul că acestea au un impact mare în ceea ce privește indicii de performanță, experimentele au fost realizate într-un mod sistematic. Experimentele sunt prezentate din punct de vedere al modificării preprocesării datelor, codificării lor, hiperparametrilor și a arhitecturii.

Deoarece cel mai reprezentativ indice de performanță este eroarea pe setul de evaluare, se vor stabili anumite referințe de bază, care vor fi comparate cu rezultatele obținute de sistem.

Metodă	**WER [%]**
Fără diacritice	31.89
1-gram	2.43

Tabel 7.3 – Referințe de bază

WER fără diacritice este calculat pe setul de evaluare în stare inițială, când nu sunt puse deloc diacritice.

Pentru a evalua performanțele modelului pe măsura antrenării se urmăresc erorile de antrenare și de validare. Eroarea de antrenare se obține pe exemplele trecute prin rețea pentru antrenare. Eroarea de validare se obține pe o mică parte din setul de antrenare, mai exact primele 5000 de exemple. Acestea nu sunt folosite la antrenare, ci doar sunt trecute prin rețea la fiecare sfârșit de epocă pentru a verifica performanțele rețelei pe date pe care nu le-a întâlnit.

În cazul referinței de tip 1-gram este înlocuit fiecare cuvânt întâlnit care ar putea avea diacritice cu forma lui cu diacritice cea mai frecvent întâlnită în setul de antrenare.

7.2.1 Preprocesarea datelor

În varianta inițială, modelul este compus dintr-o RNR cu celule de tip LSTM la nivel de caracter. Datele sunt preprocesate prin lipirea tuturor exemplelor urmată de împărțirea pe bucăți, cu riscul împărțirii tăierii în interiorul cuvintelor. Primul experiment urmărește modificarea performanțelor în urma grupării exemplelor de antrenare într-un mod adaptiv, prezentat în capitolul 6.3, față de metoda de preprocesare inițială.

Ambele antrenări au fost derulate pentru 63 de epoci, fiecare antrenare în parte a durat între 90 și 95 de ore.

Preprocesare	**WER pe setul de evaluare [%]**
Cu tăiere în interiorul cuvintelor	1.33
Grupare fără tăiere în interiorul cuvintelor	0.97

Tabel 7.4 - Rezultate experimentale ale modificării metodei de preprocesare a datelor

Se observă o îmbunătățire considerabilă adusă de schimbarea metodei de preprocesare a datelor. Deci metoda de preprocesare propusă poate fi folosită în continuare în experimentele viitoare.

Problemele ridicate în acest moment sunt timpul enorm de antrenare, datorat de faptul că se parcurge întreg setul de date, și faptul că rețeaua nu are nicio posibilitate de a se adapta în situații în care eroarea oprește să mai scadă și rețeaua nu mai învață nimic.

7.2.2 Metoda de antrenare

Utilizarea unui generator pentru antrenare aduce avantajul unei antrenări mult mai rapide și, în același timp, diversificate, prin selectarea în mod aleatoriu a exemplelor aduse în rețea la fiecare epocă. Generatorul alege, în fiecare epocă, un număr de loturi din setul de date.

Problema adaptării antrenării se poate rezolva folosind funcții de tip *callback*, disponibile în Keras, care să verifice, la sfârșitul fiecărei epoci, dacă eroarea a scăzut. Dacă aceasta nu a mai scăzut după un anumit număr de epoci, se pot lua anumite măsuri. Una este reducerea ratei de învățare, iar alta este oprirea antrenării. Fiecare dintre acest măsuri are un număr propriu de epoci fără îmbunătățiri după care pot fi luate, denumit răbdare.

În desfășurarea următoarelor experimente se fixează dimensiunea lotului de antrenare la 128 de exemple și reducerea ratei de învățare cu un factor de 0.1. Sunt variate numărul de loturi utilizate per epocă, răbdarea funcției de oprire din timp și răbdarea funcției de reducere a ratei de învățare.

Loturi per epocă	Răbdare oprire din timp (ESP)	Răbdare reducere rată de învățare (RLRP)	Număr epoci	Timp antrenare [ore]	Eroare antrenare	Eroare validare	WER set evaluare [%]	Loturi trecute prin rețea [număr loturi * număr epoci]
128	3	2	63	3.15	11.56E-03	8.10E-03	1.61	8064
128	6	3	79	2.63	9.93E-03	7.17E-03	1.42	10112
128	8	4	87	2.90	9.84E-03	6.92E-03	1.38	11136
256	6	3	62	4.13	8.66E-03	6.19E-03	1.21	11520
256	8	4	87	5.80	8.57E-03	6.08E-03	1.19	22272
512	3	2	34	3.97	8.41E-03	5.99E-03	1.19	17408
512	6	3	48	5.60	8.03E-03	5.77E-03	1.13	24576
512	8	4	49	5.72	8.08E-03	5.83E-03	1.14	25088
1024	3	2	25	5.83	7.99E-03	5.68E-03	1.12	25600
2048	3	2	20	9.33	7.44E-03	5.37E-03	1.04	40960

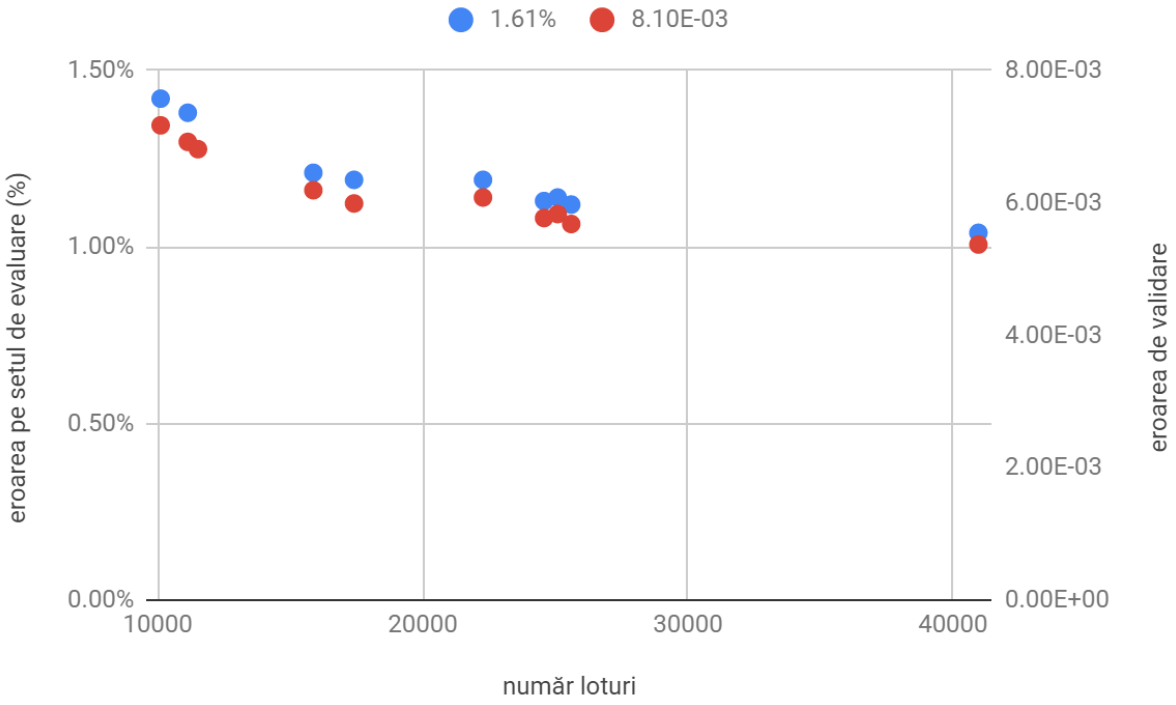
Tabel 7.5 – Rezultate experimentale ale modificării metodei de antrenare

Se observă, odată cu creșterea numărului de loturi pe epocă, o îmbunătățire dată de scăderea drastică a timpului de antrenare, dar care este acompaniată de o scădere a performanței, dată de creșterea erorilor de antrenare și validare și a WER. Totuși modelul se descurcă foarte bine comparativ cu cel anterior, deși există o diferență considerabilă între numărul de exemple trecute prin aceste rețele.

Pentru următoarele experimente se va păstra o configurație echilibrată din punct de vedere al timpului de antrenare cât și al WER pe setul de validare.

În Figură 7.1 este reprezentată grafic dependența dintre numărul de loturi văzute de rețea de-a lungul antrenării și eroarea de validare, respectiv eroarea pe setul de evaluare.

Se observă, în Figură 7.1, că din punctul în care BPE = 256, ESP = 6 și RLRP = 3, pentru care numărul de loturi este aproximativ 15000, eroarea sistemului nu mai scade la fel de mult ca pentru valorile anterioare. Se alege acest set de hiperparametri pentru realizarea următoarelor experimente, urmând să se realizeze o antrenare folosind setul pentru care performanța obținută a fost cea mai bună, împreună cu alte configurații ce vor fi ulterior descoperite.



Figură 7.1 - Variația performanțelor în funcție de numărul de loturi trecute prin rețea

7.2.3 Modificare hiperparametri prin optimizare

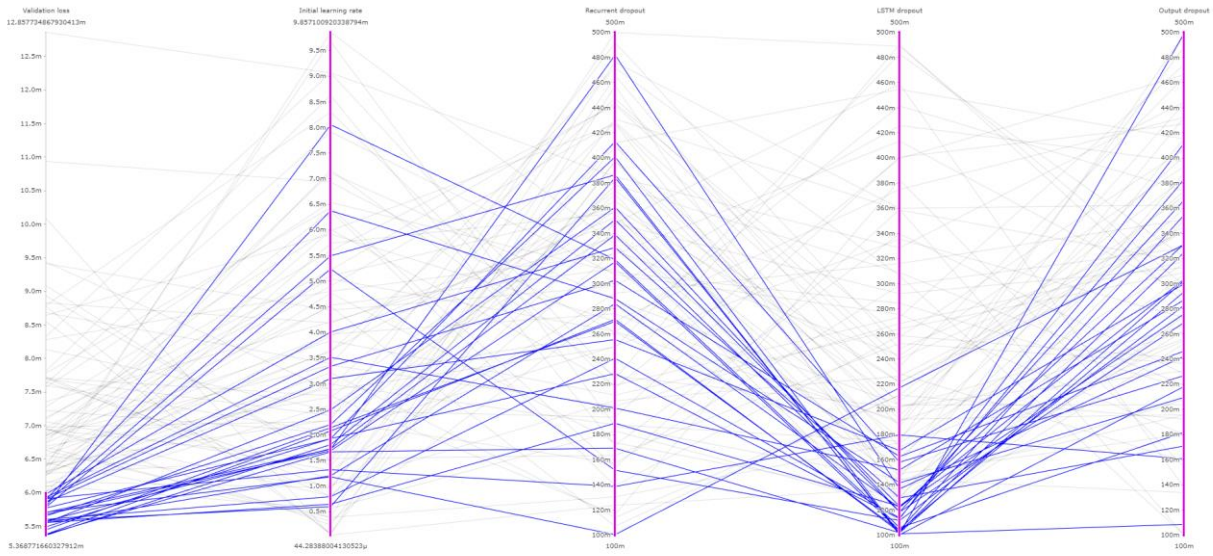
Se dorește obținerea unor valori optime pentru rata de învățare inițială și pentru ratele de renunțare. Experimentele de până acum au folosit rata de învățare 0.001 și toate ratele de renunțare 0.2.

Optimizarea se va realiza cu ajutorul librăriei Hyperopt. Se definește spațiul de căutare ca fiind intervalul [0.00001, 0.01] pentru rata de învățare și [0.1, 0.5] pentru ratele de renunțare.

În urma realizării a 100 de experimente a fost selectat setul de hiperparametri cu care eroarea de validare a fost minimă.

Arhitectură	Rata de învățare inițială	LSTM dropout	Recurrent dropout	Output dropout	Eroare antrenare	Eroare validare	WER set evaluare [%]
Înainte de optimizare hiperparametri	1.00E-03	0.20	0.20	0.20	8.66E-03	6.19E-03	1.21
După optimizare hiperparametri	1.81E-03	0.12	0.35	0.28	7.49E-03	5.53E-03	1.11

Figură 7.2 – Rezultate experimentale obținute în urma optimizării hiperparametrilor



Figură 7.3 - Influența variației hiperparametrilor asupra erorii de validare

În Figură 7.3 este un grafic cu toate experimentele realizate în procesul de optimizare. Prima coloană din stânga este metrica urmărită, eroarea de validare. Următoarele coloane sunt valorile variabilelor de control. Evidențiate cu albastru sunt experimentele pentru care s-au obținut cele mai mici valori ale erorii de validare, cu scopul determinării variabilelor de control care au rolul cel mai important în determinarea acesteia. Se observă faptul că pentru coloanele 2, 3 și 5 nu există o corelație cu valorile erorii. În schimb pe coloana a 3 a, lstm_dropout, valorile sunt concentrate în jurul pragului inferior, ceea ce denotă o corespondență între valori mici ale acestui hiperparametru și valori mici ale erorii de validare.

7.2.4 Arhitectura rețelei

Experimentele de până acum utilizează o rețea neurală bidirecțională cu celule LSTM. Se va modifica arhitectura rețelei având în vedere numărul de straturi, tipul de celulă și direcția (unidirecțională sau bidirecțională). Sunt utilizați hiperparametrii impliciți, nu cei obținuți prin optimizare.

Denumire arhitectură	Tipul celulei	Direcție	Nr straturi	Număr parametri [milioane]	Număr epoci	Timp antrenare [ore]	Eroare antrenare	Eroare validare	WER set evaluare [%]
BiLSTM-1	LSTM	Bidirecțional	1	0.8	62	4.13	8.66E-03	6.19E-03	1.21
BiLSTM-2	LSTM	Bidirecțional	2	2.4	55	9.2	3.92E-03	3.07E-03	0.58
LSTM-1	LSTM	Unidirecțional	1	0.4	71	2.9	5.71E-02	5.14E-02	11.53
LSTM-2	LSTM	Unidirecțional	2	0.9	88	7.0	5.42E-02	4.93E-02	11.12
BiGRU-1	GRU	Bidirecțional	1	0.6	65	4.0	1.04E-02	6.99E-03	1.41
BiGRU-2	GRU	Bidirecțional	2	1.8	47	6.1	5.21E-03	3.64E-03	0.71

Tabel 7.6 – Rezultate experimentale modificare arhitectură rețea

Se observă mai multe lucruri importante.

- Performanțele cresc odată cu stivuirea a două straturi recurente.
- Utilizarea unei rețele unidirecționale scade drastic performanța sistemului.
- Celula de tip GRU obține rezultate mai slabe, dar apropiate de cea LSTM.
- Arhitectura de rețea bidirecțională cu 2 straturi stivuite și celule LSTM obține cele mai bune rezultate, dar este în același timp și cea mai costisitoare, atât din punct de vedere al memoriei, vezi numărul de parametri, cât și al timpului de antrenare.

Utilizarea hiperaparametrilor optimizați pe arhitectura inițială aduce o ușoară creștere a performanțelor celei mai bune arhitecturi.

Denumire arhitectură	Rata de învățare inițială	LSTM dropout	Recurrent dropout	Output dropout	Eroare antrenare	Eroare validare	WER set evaluare [%]
BiLSTM-2	1.00E-03	0.20	0.20	0.20	3.92E-03	3.07E-03	0.58
BiLSTM-2	1.81E-03	0.12	0.35	0.28	3.61E-03	2.85E-03	0.54

Tabel 7.7 – Arhitectura cu performanțe cele mai bune și hiperparametri optimizați

7.2.5 Modele la nivel de cuvânt și hibride

Dacă până acum experimentele au fost realizate la nivel de caracter, următoarele vor urmări realizarea lor la nivel de cuvânt și într-un mod hibrid, combinând informațiile la nivel de cuvânt și caracter.

Rețeaua la nivel de cuvânt

Este utilizată arhitectura BiLSTM-2 cu hiperparametri optimi.

Datorită faptului că o rețea la nivel de cuvânt trebuie să realizeze clasificarea cuvintelor de la intrare într-un număr de clase egal cu vocabularul cuvintelor de la ieșire, se creează un ultim strat de clasificare de dimensiuni mari, cu foarte multe ponderi, care se antrenează greu și care ocupă mai multă memorie decât este disponibilă. Singura soluție viabilă este limitarea vocabularului. Prin următoarele experimente se dorește analizarea arhitecturii la nivel de cuvânt atunci când vocabularul de la ieșire este limitat la diferite valori.

Denumire arhitectură	Dimensiune vocabular ieșire	Număr epoci	Timp antrenare [ore]	Eroare antrenare	Eroare validare	WER set evaluare [%]
Word-level	5k	62	10	1.34E-02	1.13E-02	8.9971
	10k	55	21	1.60E-02	1.30E-02	8.9152
	15k	71	36	1.68E-02	1.35E-02	8.9076

Tabel 7.8 – Rezultate experimentale arhitectură la nivel de cuvânt

În timpul evaluării rețelei aceasta a fost predispusă la prezicerea de codificări greșite, care duceau la punerea unor alte cuvinte în locul celor inițiale care nu aveau nicio legătură. Acest aspect a fost reparat în pasul de post-procesare prin înlocuirea cuvântului de la ieșire cu cel de la intrare dacă are o altă formă.

Timpul de antrenare per epocă nu a fost raportat deoarece au existat fluctuații mari a acestei valori.

Rețeaua oferă performanțe slabe, iar utilizarea pasului de post-procesare este ceea ce ajută îmbunătățirea lor.

Rețeaua hibridă

Este cea prezentată în capitolul 5.3. În primă instanță a fost folosită arhitectura BiLSTM-1 cu hiperparametri optimi atât pentru calea la nivel de caracter cât și pentru calea la nivel de cuvânt. Această arhitectură a fost denumită *word-char-combined*.

Ulterior au mai fost dezvoltate două arhitecturi.

i. Adăugarea unui strat de RNR bidirecțional cu celulă LSTM după pasul de concatenare a informațiilor la nivel de caracter și la nivel de cuvânt. Arhitectura a fost denumită *word-char-combined-2*.

ii. Utilizarea arhitecturii BiLSTM-2 cu hiperparametri optimi pentru calea la nivel de caracter și pentru calea la nivel de cuvânt. Arhitectura a fost denumită *word-char-combined-3.*

Rezultatele obținute pot fi observate în

Denumire arhitectură	Număr epoci	Timp antrenare [ore]	Eroare antrenare	Eroare validare	WER set evaluare [%]
Word-char-combined (BiLSTM-1)	65	5.4	5.91E-03	6.08E-03	3.23
Word-char-combined-2 (BiLSTM-1 după concat)	52	7.8	3.10E-03	3.20E-03	-
Word-char-combined-3 (BiLSTM-2)	46	7.6	4.09E-03	4.17E-03	-

Tabel 7.9 – Rezultate experimentale model hibrid

7.2.6 Modificare codificării caracterelor de la ieşire

Experimentele de până acum au folosit codificarea la ieșire bazată pe clasificarea fiecărui caracter în alt caracter, dintr-o listă.. Numărul de clase este egal cu numărul de elemente din listă, mai exact 44.

Următoarele experimente vor folosi codificarea caracterelor în 3 clase, metodă descrisă în capitolul 5.1.

Denumire arhitectură	Număr epoci	Timp antrenare [ore]	Eroare antrenare	Eroare validare	WER set evaluare [%]
BiLSTM-2	60	10	3.33E-03	2.74E-03	0.55
Word-char-combined	50	5	2.70E-03	3.30E-03	2.69
Word-char-combined-2	56	8.4	2.20E-03	2.60E-03	4.15
Word-char-combined-3	50	7.5	2.40E-03	2.90E-03	3.58

Tabel 7.10 – Rezultate experimentale obținute în urma utilizării a 3 clase

Pentru evaluarea modelului a fost nevoie de un pas suplimentar de postprocesare pentru a trece din clasa în caracter.

7.2.7 Modelul cel mai bun

Cele mai bune rezultate aparțin arhitecturii RNR la nivel de caracter, cu 2 straturi stivuite, celule LSTM și bidirecțională, obținând un WER pe setul de validare de 0.54 %. Performanțele acesteia se pot îmbunătăți prin adăugarea unui număr mai mare de exemple de antrenare. Folosind antrenarea cu un generator ce aduce 2048 de loturi de antrenare în fiecare epocă, WER se îmbunătățește la 0.48%.

În Tabel 7.11 sunt prezentate precizia, reamintirea și scorul F obținute la nivelul caracterelor cu diacritice și al caracterelor care acceptă diacritice. Sistemul obține rezultate bune pe majoritatea caracterelor, cu excepție formele cu diacritice ale literei “a”.

Caracter	Precizie	Reamintire	Scor F
“a”	99.64	99.49	99.56
“ă”	98.64	99.03	98.84
“â”	99.79	99.87	99.83
“i”	99.98	99.99	99.99
“î”	99.94	99.90	99.92
“s”	99.95	99.97	99.96
“ș”	99.91	99.85	99.88
“t”	99.93	99.95	99.94
“ț”	99.73	99.64	99.68

Tabel 7.11 - Metrici de performanță la nivelul caracterelor

CONCLUZII

Restaurarea diacriticelor este o sarcină interesantă, cu un domeniu de aplicabilitate larg. Ea poate fi abordată din mai multe perspective în urma cărora se pot obține rezultate în conformitate cu rigorile gramaticii limbii române. Rezolvarea ei utilizând tehnici bazate pe rețele neurale profunde este probabil cea mai complexă abordare, dar și cu cel mai mare potențial.

Ideea este una simplă în aparență, dar proiectarea unui astfel de sistem trebuie făcută având în vedere multe aspecte. După cum am văzut, aceste aspecte pot fi preprocesarea datelor, arhitectura rețelei, codificarea datelor, dimensiunile sau metoda de antrenare.

Chiar dacă dezvoltarea unei rețele neurale implică folosirea multor cunoștințe matematică, acest lucru este rezolvat prin apariția de librării precum Keras [22] prin intermediul cărora se poate defini într-un mod ușor rețeaua. Căutarea configurației optime este adevărata provocare în ceea ce privește dezvoltarea unui asemenea sistem, deoarece există o gamă foarte largă de parametri și hiperparametri ce pot fi modificați. Alegerea lor nu este întotdeauna intuitivă și de cele mai multe ori e nevoie de determinarea lor experimentală. În ajutorul acestui aspect vin librării precum Hyperopt [23], cu ajutorul cărora căutarea setului optim se poate face într-un mod automat și inteligent.

Pe lângă implementarea operațiilor matematice și alegerea setului de hiperparametri apare o altă problemă, complexitatea temporală necesară pentru antrenarea rețelelor neurale profunde. Pentru îmbunătățirea acestui aspect se folosesc opțiuni precum rularea operațiunilor pe plăci grafice sau utilizarea doar a anumitor exemple din setul de antrenare. De asemenea, există și o complexitate spațială considerabilă, dat fiind numărul mare de ponderi la care poate ajunge o rețea neurală.

Chiar și așa, dezvoltarea acestui tip de sistem se face într-o manieră sistematică, prin realizarea mai multor experimente cu diferiți parametri. În cadrul experimentelor trebuie totuși ca metricile de performanță să rămână constante pentru a putea realiza comparația rețelelor.

O abordare sistematică, pas cu pas, tratând diferite aspecte ale rețelei, este probabil cea mai bună metodă de a explora variantele existente, împreună cu implicațiile lor.

În cadrul lucrării s-a pornit de la un model simplu care, pe măsură ce s-au adus modificări, s-a îmbunătățit. Totuși, în momentul în care s-a ajuns la un model mai complex de tipul celui hibrid, așteptarea ar fi fost ca performanțele să fie cele mai bune. Acest lucru, însă, nu s-a întâmplat.

Un alt aspect al rețelelor neurale este necesitatea existenței unui set de date de antrenare îndeajuns de mare. În cadrul lucrării a fost disponibil un corpus de text cu diacritice corecte, cu care s-a putut antrena rețeaua. De asemenea, în paralel cu dezvoltarea rețelei, a fost colectat și un alt corpus de text folosind o aplicație de tip web-crawler. Corpusul colectat poate fi folosit pentru antrenarea rețelei dezvoltate, dar poate fi utilizat și în alte aplicații.

Lucrarea ilustrează faptul că utilizând o arhitectură de RNR relativ simplă, cu parametrii potriviți, sarcina restaurării diacriticelor poate fi rezolvată cu succes.

CONTRIBUȚII PERSONALE

Codul inițial al arhitecturii RNR a fost pus la dispoziție de către laboratorul SpeeD. Mai departe, am avut datoria de a-i aduce îmbunătățiri și de a implementa, pe baza lui, diferite experimente.

Principala contribuție este realizarea de experimente și interpretarea rezultatelor lor.

În scopul realizării experimentelor, am avut în vedere implementarea mai multor aspecte:

- Scrierea codului necesar arhitecturilor și metodelor prevăzute de experiment. Realizarea acestora presupune definirea modelului, preprocesarea datelor, codificarea datelor și implementarea procesului de antrenare și cel de testare;
- Dezvoltarea algoritmul de grupare a exemplelor în exemple de lungime fixă, într-un mod adaptiv;
- Configurarea librăriei Hyperopt [23] pentru a funcționa, în cadrul proiectului, întâi pe un singur calculator, apoi în mod distribuit.

Pentru interpretarea rezultatelor au fost dezvoltate scripturi cu rolul analizei rezultatelor experimentale și a fișierelor ce conțin text cu diacritice restaurat

BIBLIOGRAFIE

[1] D. Tufiș și A. Ceaușu, „DIAC+: A professional diacritics recovering system," *Proceedings of LREC,* 2008.

[2] L. Petrică, H. Cucu, A. Buzo și C. Burileanu, „A Robust Diacritics Restoration System Using Unreliable Raw Text Data," *Spoken Language Technologies for Under-Resourced Languages,* 2014.

[3] Ș. Rușeți, T.-M. Cotet și M. Dascălu, „Romanian Diacritics Restoration Using Recurrent Neural Networks," 2018.

[4] H. Cristescu, „Romanian-diacritics-restoration," 2018. [Interactiv]. Available: https://github.com/horiacristescu/romanian-diacritic-restoration. [Accesat 26 Iunie 2019].

[5] H. Cucu, A. Buzo, L. Besacier și C. Burileanu, „SMT-based ASR Domain Adaptation Methods For Under-Resourced Languages: Application To Romanian," *Speech Communication,* vol. 56, pp. 195-212, 2013.

[6] C. E. Shannon, „A Mathematical Theory of Communication," *The Bell System Technical Journal,* 1948.

[7] D. Jurafsky și J. H. Martin, Speech and Language Processing (3rd ed. draft), 2018.

[8] R. Livni, S. Shalev-Shwartz și O. Shamir, „On the Computational Efficiency of Training Neural Networks," *Advances in neural information processing systems,* 2014.

[9] I. Goodfellow, Y. Bengio și A. Courville, Deep Learning, MIT Press, 2016.

[10] Y. Upadhyay, „Towards Data Science: Introduction to FeedForward Neural Networks," 7 martie 2019. [Interactiv]. Available: https://towardsdatascience.com/feed-forward-neural-networks-c503faa46620. [Accesat 17 iunie 2019].

[11] Skymind, „A Beginner's Guide to LSTMs and Recurrent Neural Networks," [Interactiv]. Available: https://skymind.ai/wiki/lstm#recurrent. [Accesat 18 iunie 2019].

[12] S. Ronaghan, „Deep Learning: Overview of Neurons and Activation Functions," 26 iulie 2018. [Interactiv]. Available: https://medium.com/@srnghn/deep-learning-overview-of-neurons-and-activation-functions-1d98286cf1e4. [Accesat 19 iunie 2019].

[13] O'Reilly, „Softmax in TensorFlow," [Interactiv]. Available: https://www.oreilly.com/library/view/practical-convolutional-neural/9781788392303/2e8989a5-acd0-460a-bd66-44602f26a926.xhtml. [Accesat 19 iunie 2019].

[14] J. Brownlee, „Loss and Loss Functions for Training Deep Learning Neural Networks," 28 ianuarie 2019. [Interactiv]. Available: https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/. [Accesat 20 iunie 2019].

[15] R. Reed și R. J. MarksII, în *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks,* 1999, p. 155.

[16] A. S. Walia, „Types of Optimization Algorithms used in Neural Networks and Ways to Optimize Gradient Descent," 10 iunie 2017. [Interactiv]. Available: https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f. [Accesat 20 iunie 2019].

[17] O. Grigore, *Rețele neuronale și Sisteme fuzzy,* Universitatea Politehnica din București, 2019.

[18] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever și R. Salakhutdinov, „Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research,* 2014.

[19] J. Brownlee, „How to Use Word Embedding Layers for Deep Learning with Keras," 4 octombrie 2017. [Interactiv]. Available: https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/. [Accesat 21 iunie 2019].

[20] T. Mikolov, I. Sutskever, K. Chen, G. Corrado și J. Dean, „Distributed representations of words and phrases and their compositionality," în *26th International Conference on Neural Information Processing Systems,* 2013.

[21] Google Brain, „TensorFlow: A system for large-scale machine learning," în *12th USENIX Symposium on Operating Systems Design and Implementation,* 2016.

[22] F. Chollet, „keras," 2015. [Interactiv]. Available: https://github.com/keras-team/keras. [Accesat 22 iunie 2019].

[23] J. Bergstra, D. Yamins și D. D. Cox, „Hyperopt: A Python library for optimizing the hyperparameters of machine learning algorithms," 2013.

[24] D. Saha, „What is MongoDB and Why to use it?," 27 mai 2015. [Interactiv]. Available: https://www.dotnettricks.com/learn/mongodb/what-is-mongodb-and-why-to-use-it. [Accesat 24 iunie 2019].

[25] DataFlair, „Why MongoDB – 10 Reasons to Learn MongoDB for 2019," 2019 ianuarie 2019. [Interactiv]. Available: https://data-flair.training/blogs/why-mongodb/. [Accesat 24 iunie 2019].

[26] nVIDIA, „CUDA," [Interactiv]. Available: https://developer.nvidia.com/cuda-zone. [Accesat 24 iunie 2019].

[27] „snacktory," [Interactiv]. Available: https://github.com/karussell/snacktory. [Accesat 26 iunie 2019].

[28] A. Karpathy, *CS231n Convolutional Neural Networks for Visual Recognition,* Stanford University, 2018.

[29] A. Karpathy, „The Unreasonable Effectiveness of Recurrent Neural Networks," 21 mai 2015. [Interactiv]. Available: http://karpathy.github.io/2015/05/21/rnn-effectiveness/. [Accesat 19 iunie 2019].

[30] C. Olah, „Understanding LSTM Networks," 27 august 2015. [Interactiv]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accesat 19 iunie 2019].

ANEXA 1

```
1.  import argparse
2.  import functools
3.  import os
4.  import sys
5.  import pdb
6.  import datetime
7.  import time
8.  import pdb
9.  import math
10. import numpy as np
11. import hyperopt
12. import objective
13. from hyperopt.mongoexp import MongoTrials
14. from textwrap import wrap
15. from itertools import groupby
16. from collections import Counter
17. from keras.callbacks import EarlyStopping, ModelCheckpoint, Callback,TensorBoard, Redu
    ceLROnPlateau, CSVLogger
18.
19. from keras.layers import Bidirectional, Dense, Dropout, Embedding, Input, LSTM
20.
21. from keras.optimizers import Adam
22.
23. from keras.models import Model, load_model
24.
25. from keras.utils import Sequence, to_categorical
26.
27. from keras.preprocessing.sequence import pad_sequences
28.
29. CHARS_INP = " \n-0123456789abcdefghijklmnopqrstuvwxyz"
30. CHARS_OUT = " \n-0123456789abcdefghijklmnopqrstuvwxyzăâîșț"
31.
32. NO_DIACRITICS_MAP = {"ă": "a", "â": "a", "î": "i", "ș": "s", "ț": "t"}
33.
34.
35. def read_data(name):
36.     with open(name, "r", encoding="utf-8") as f:
37.         return f.read()
38.
39.
40. def write_data(name, text):
41.     with open(name, "w", encoding="utf-8") as f:
42.         return f.write(text)
43.
44.
45. def remove_diacritics(text):
46.     if isinstance(text, list):
```

```
        return [
            sentence.translate(str.maketrans(NO_DIACRITICS_MAP)) for sentence in text

        ]
    else:
        return text.translate(str.maketrans(NO_DIACRITICS_MAP))


def encode(text, chars):
    char_to_id = {c: i for i, c in enumerate(chars)}
    if isinstance(text, list):
        return [[char_to_id[c] for c in sentence] for sentence in text]
    else:
        return [char_to_id[c] for c in text]


def decode(idxs, chars):
    id_to_char = dict(enumerate(chars))
    return "".join(id_to_char[i] for i in idxs)


def get_model(
    seq_len=16,
    emb_dim=64,
    lstm_dim=64,
    lstm_dropout=0.2,
    recurrent_dropout=0.2,
    output_dropout=0.2,
):
    inp = Input(shape=(None,))
    emb = Embedding(input_dim=len(CHARS_INP), output_dim=emb_dim)(inp)
    lstm_1 = LSTM(
        lstm_dim,
        return_sequences=True,
        input_shape=(None, 32),
        dropout=lstm_dropout,
        recurrent_dropout=recurrent_dropout,
    )
    bilstm_1 = Dropout(output_dropout)(Bidirectional(lstm_1)(emb))

    lstm_2 = LSTM(
        lstm_dim,
        return_sequences=True,
        input_shape=(None, 32),
        dropout=lstm_dropout,
        recurrent_dropout=recurrent_dropout,
    )
    bilstm_2 = Dropout(output_dropout)(Bidirectional(lstm_2)(bilstm_1))

    out = Dense(len(CHARS_OUT), activation="softmax")(bilstm_2)
    model = Model(inp, out)
    return model


       def filter_by_length(sentences, min_length):
           return [word for word in sentences if len(word) >= min_length]


       def wrap_data(text, seq_len):
           to_ret = []
           for sentence in text:
               if len(sentence) <= seq_len:
                   to_ret.append(sentence)
               else:
                   to_ret.extend(wrap(sentence, seq_len))
           return to_ret


```

```
114.        def index_sentences(text):
115.            sorted_list = sorted(text, key=len, reverse=True)
116.            indexed = {k: list(g) for k, g in groupby(sorted_list, key=len)}
117.            return indexed
118.
119.
120.        def group_indexed(grouped, sentences, seq_len, min_length):
121.            to_ret = []
122.            for key in sorted(grouped.keys(), reverse=True):
123.                print(key)
124.                try:
125.                    grouped[key]
126.                except KeyError:
127.                    continue
128.                while len(grouped[key]) > 0:
129.                    current_sentence = grouped[key].pop()
130.                    if key + min_length <= seq_len:
131.                        to_fill = seq_len - key - 1
132.                        while to_fill >= min_length:
133.                            try:
134.                                current_sentence += "\n" + grouped[to_fill].pop()
135.                                to_fill = seq_len - len(current_sentence) - 1
136.                            except KeyError:
137.                                to_fill -= 1
138.                            except IndexError:
139.                                to_fill -= 1
140.                    to_ret.append(current_sentence)
141.            return to_ret
142.
143.
144.        class CustomCsv(CSVLogger):
145.            def __init__(self, file_name, append, separator):
146.                self.start = datetime.datetime.now()
147.                super().__init__(file_name, append=append, separator=separator)
148.            def on_epoch_end(self, batch, logs={}):
149.                logs['time'] = (datetime.datetime.now() - self.start).total_seconds()/6
    0
150.                super().on_epoch_end(batch, logs)
151.
152.        class Generator(Sequence):
153.            def __init__(self, x_set, y_set, batch_size=256, steps=None):
154.                self.x, self.y = x_set, y_set
155.                self.batch_size = batch_size
156.                self.indices = np.arange(self.x.shape[0])
157.                self.steps = steps
158.
159.            def __len__(self):
160.                return self.steps or math.ceil(self.x.shape[0] / self.batch_size)
161.
162.            def __getitem__(self, idx):
163.                inds = self.indices[idx * self.batch_size:(idx + 1) * self.batch_size]

164.                batch_x = self.x[inds]
165.                batch_y = to_categorical(self.y[inds], num_classes=len(CHARS_OUT))
166.                return batch_x, batch_y
167.
168.            def on_epoch_end(self):
169.                np.random.shuffle(self.indices)
170.
171.
172.        def train(data_file_name, model_file_name, steps_per_epoch, reduce_lr_patience,
     early_stopping_patience):
173.            seq_len = 256
174.            emb_dim = 128
175.            lstm_dim = 256
176.            lstm_dropout = 0.0744918661125156
177.            recurrent_dropout = 0.350952475317764
178.            output_dropout = 0.276006937077111
```

```
179.            initial_lr = 0.00181100241785388
180.            min_length = 8
181.            validation_size = 5000
182.
183.            sentences = read_data(data_file_name).split("\n")
184.
185.            print("Started wrapping")
186.            print(datetime.datetime.now().time())
187.            sentences = wrap_data(sentences, seq_len)
188.            print("Started filtering")
189.            print(datetime.datetime.now().time())
190.            sentences = filter_by_length(sentences, min_length)
191.            print("Started indexing")
192.            print(datetime.datetime.now().time())
193.
194.            indexed_sentences = index_sentences(sentences)
195.            grouped_indexed = group_indexed(indexed_sentences, sentences, seq_len, min_
    length)
196.
197.            validation_data = grouped_indexed[:validation_size]
198.            train_data = grouped_indexed[validation_size:]
199.
200.            text_inp = remove_diacritics(train_data)
201.            text_inp_val = remove_diacritics(validation_data)
202.
203.            if model_file_name is None:
204.                model_file_name = (
205.                    "models/model-
    {}".format(datetime.datetime.now().strftime('%Y_%m_%d_%H_%M'))
206.                )
207.
208.            if os.path.isfile(model_file_name) is not True:
209.                model = get_model(seq_len=seq_len,
210.                                  emb_dim=emb_dim,
211.                                  lstm_dim=lstm_dim,
212.                                  lstm_dropout=lstm_dropout,
213.                                  recurrent_dropout=recurrent_dropout,
214.                                  output_dropout=output_dropout)
215.
216.                model.compile(loss="categorical_crossentropy", optimizer=Adam(lr=initia
    l_lr))
217.                print(model.summary())
218.            else:
219.                model = load_model(model_file_name)
220.            # print(f"Training on {steps_per_epoch} steps with reducing learning rate p
    atience {reduce_lr_patience} and early stopping patience {early_stopping_patience}")
221.            checkpoint = ModelCheckpoint(model_file_name, verbose=1, save_best_only=Tru
    e)
222.            reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=redu
    ce_lr_patience, verbose=1)
223.            csv_logger = CustomCsv("log_{}.csv".format(datetime.datetime.now().strftime
    ('%Y_%m_%d_%H_%M')), append=True, separator=';')
224.            early_stopping = EarlyStopping(patience=early_stopping_patience)
225.            tensorboard = TensorBoard(log_dir="logs/{}".format(datetime.datetime.now().
    strftime('%Y_%m_%d_%H_%M')))
226.            X = pad_sequences(encode(text_inp, CHARS_INP))
227.            Y = pad_sequences(encode(train_data, CHARS_OUT))
228.
229.            X_val = pad_sequences(encode(text_inp_val, CHARS_INP))
230.            Y_val = pad_sequences(encode(validation_data, CHARS_OUT))
231.            Y_val = to_categorical(Y_val, num_classes=len(CHARS_OUT))
232.
233.            steps = steps_per_epoch
234.            generator = Generator(X, Y, batch_size=128, steps=steps)
235.            model.fit_generator(
236.                generator, steps_per_epoch=steps, epochs=128, validation_data=(X_val, Y
    _val),
```

```
237.                callbacks=[checkpoint, early_stopping, reduce_lr, tensorboard, csv_logg
    er]
238.            )
239.
240.
241.        def optimize():
242.            print("Starting optimizer")
243.
244.            trials = MongoTrials("mongo://name:pass@ip:port/hyperopt/jobs?authSource=hy
    peropt", exp_key="new3")
245.            best = hyperopt.fmin(
246.                objective.to_min,
247.                space=[
248.                    hyperopt.hp.uniform("lr", 0.00001, 0.01),
249.                    hyperopt.hp.uniform("lstm_dropout", 0.05, 0.2),
250.                    hyperopt.hp.uniform("recurrent_dropout", 0.1, 0.5),
251.                    hyperopt.hp.uniform("output_dropout", 0.1, 0.5),
252.                ],
253.                trials=trials,
254.                algo=hyperopt.tpe.suggest,
255.                max_evals=40,
256.            )
257.            print(best)
258.
259.
260.        def test(model_file_name, out_file_name="out"):
261.            outputs = []
262.            os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"
263.            os.environ["CUDA_VISIBLE_DEVICES"] = ""
264.            model = load_model(model_file_name)
265.            sentences = read_data("data/validation_set_wodia").split("\n")[:-1]
266.            for i, sentence in enumerate(sentences):
267.                encoded_sentence = encode(sentence, CHARS_INP)
268.                converted_sentence = np.array(encoded_sentence).reshape(-
    1,len(encoded_sentence))
269.                predicted = model.predict(converted_sentence).argmax(axis=2).flatten()

270.                decoded_prediction = decode(predicted, CHARS_OUT)
271.                outputs.append(decoded_prediction)
272.                curr = i*100/len(sentences)
273.                sys.stdout.write("\r%0.2f%%" % curr)
274.                sys.stdout.flush()
275.            write_data(out_file_name, "\n".join(outputs))
276.
277.
278.        def main():
279.            parser = argparse.ArgumentParser(
280.                description="Deep learning for diacritics restoration."
281.            )
282.
283.            parser.add_argument(
284.                "--
    todo", choices=["train", "test", "optimize"], nargs="+", help="what to do"
285.            )
286.
287.            parser.add_argument(
288.                "--
    model", type=str, help="where to store new model/where to load model from"
289.            )
290.
291.            parser.add_argument(
292.                "--
    steps_per_epoch", type=int, help="where to store new model/where to load model from"
293.            )
294.
295.            parser.add_argument(
```

```
296.                 "--
    reduce_lr_patience", type=int, help="where to store new model/where to load model from
    "
297.             )
298.
299.             parser.add_argument(
300.                 "--
    early_stopping_patience", type=int, help="where to store new model/where to load model
     from"
301.             )
302.
303.             parser.add_argument(
304.                 "--out_file", type=str, help="where to store test output file"
305.             )
306.
307.             parser.add_argument("--data", type=str, help="training data location")
308.
309.             args = parser.parse_args()
310.
311.             model_file_name = args.model
312.             data_file_name = args.data
313.
314.             if "train" in args.todo:
315.                 train(data_file_name, model_file_name, args.steps_per_epoch, args.reduc
    e_lr_patience, args.early_stopping_patience)
316.
317.             if "test" in args.todo:
318.                 test(model_file_name, args.out_file)
319.
320.             if "optimize" in args.todo:
321.                 optimize()
322.
323.
324.         if __name__ == "__main__":
325.             main()
```