UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA

FACULTATEA DE MATEMATICǍ ŞI INFORMATICǍ

SPECIALIZAREA INFORMATICǍ

LUCRARE DE LICENȚǍ

Învățarea supervizată în

predicția textelor

Conducător ştiinţific

Prof. univ. dr. Czibula Gabriela

Absolvent

**Naste Teodora Ioana**

**CUPRINS**

**INTRODUCERE**…………………………………………………..?

**1. [Primul capitol]** …………………………………………………?

1.1 [Primul subcapitol] ……………………………………………?

….

**4. [Aplicaţia practică]**……………………………………………...?

4.1 Dezvoltarea aplicației………………………………………….?

4.1.1 Definirea problemei și specificații funcționale …………...?

4.1.2 Analiză şi proiectare………………………………..……..?

4.1.3 Implementare…………………………………………… ?

4.1.4 Manual de utilizare……………………………… ………?

4.2 Rezultate experimentale………………………………………..?

4.3 Extinderi posibile……………………………………………....?

**CONCLUZII**………………………………………………………..?

**BIBLIOGRAFIE**……………………………………………………?

**INTRODUCERE**

Oamenii nu pot să scrie la fel de repede precum gândesc. O persoană tastează aproximativ 41,4 cuvinte pe minut; recordul mondial fiind de 216 cuvinte pe minut [18]. Însă pentru o persoană cu anumite deficiențe, această viteză este dificil de atins. O opțiune ar fii dispozitivele de recunoaștere a vocii, dar acestea nu pot fii folosite de persoanele cu deficiențe de vorbire. De aceea, în prezent, multe aplicații au un sistem de predicție automată a textului scris. Un astfel de sistem încearcă să prezică porțiuni de text, bazându-se pe textul deja introdus de utilizator. Astfel, tastarea caracter cu caracter este înlocuită de o simplă selecție a cuvântului dorit imediat ce acesta este oferit de către sistem, realizând astfel o comunicare mai ușoară și mai rapidă.

**CAPITOLUL 1**

**Rețele neuronale artificiale**

**Introducere**

Ideea de rețea neuronală a apărut încă din anul 1943, când McCulloch și Pitts au introdus pentru prima dată conceptul de neuroni. Aceștia erau elemente conceptuale ale unui circuit care puteau efectua simple operații. Însă interesul în rețelele neuronale a apărut abia în anii ’80, când dezvoltarea tehnologiei a crescut capacitatea de procesare a calculatoarelor. [8] De-a lungul timpului modelul de rețea neuronală s-a dezvoltat pentru a putea rezolva probleme cât mai diverese, de exemplu detecția obiectelor în imagini sau recunoașterea vorbirii; astfel au apărut modele mai complicate ale rețelelor neuronale clasice.

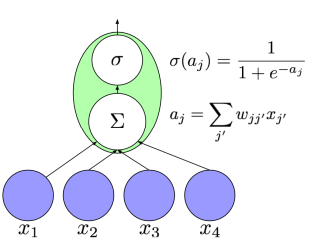
În acest capitol, voi prezenta pe scurt modul de funcționare al rețelelor neuronale tradiționale, urmând să detaliez rețelele neuronale recurente și un model aparte, anume LSTM (*Long Short-Term Memory*).

* 1. **Rețele neuronale artificiale**

**1.1.1 Modul de funcționare**

Rețelele neuronale artificiale sunt modele inspirate din structura creierului uman. Acestea sunt formate din un număr foarte mare de neuroni artificiali, denumiți noduri, fiind uniți de muchii unidirecționale. Fiecare nod *j* are asociată o funcție de activare , iar fiecare muchie are asociată o pondere , ce smenifică ponderea de la nodul *j* la nodul *j’* și nu este echivalentă cu , care reprezintă ponderea în sens invers. Astfel , valoarea a nodului  *j*  , calculată aplicând funcția de activare este:

În figura 1 [9], este prezentat felul în care un nod calculează valoarea unei funcții nonlineare , folosind suma ponderilor datelor sale de intrare.



**Figura 1. Modul de funcționare al unui neuron artificial**

Printre proprietățile particulare ale rețelelor neuronale se numără abilitatea de a învăța și a se adapta, de a generaliza o problemă sau de a organiza datele.

**1.1.2 Funcția de activare**

Funcția de activare poate să fie o funcție liniară simplă, astfel valoarea unui nod s-ar calcula după formula [8]:

Funcțiile de activare cel mai des folosite sunt funcțiile sigmoid și funcția tanh . Acestea sunt foarte practicate pentru rețelele *feedforward* și au fost aplicate și pentru rețelele neuronale recurente. O altă funcție de activare este funcția ReLU (*rectified linear unit*) , care este folosită, în principal, la rețelele neuronale cu structuri adânci (*deep neural networks*) pentru obiective precum detecția imaginilor, dar a fost folosită și în rețelele neuronale recurente de către Bengio et al. (2013) [9].

**1.1.3 Propagarea erorii**

Propagarea erorii este procesul prin care rețeaua învață. Valoarea obținută de noduri este trimisă la nodurile de ieșire care compară cu rezultatul dorit. Astfel o să avem o eroare pentru fiecare nod de ieșire. Scopul principal este minimizarea acestei erori. Metoda cea mai simplă pentru atingerea acestui scop este să se distribuie eroare unui nod de ieșire către toate nodurilor conectate cu acesta, proporţional cu valorile ponderilor asociate acestor conexiuni; după care se modifică ponderile de pe stratul de intrare și stratul ascuns.

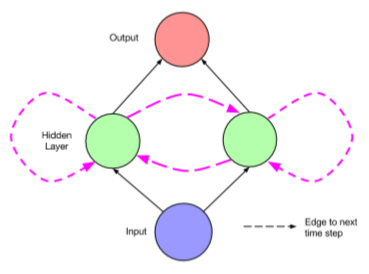
**1.1.4 Limitări**

Rețelele neuronale clasice sunt capabile să genereze aproximarea unei funcții nonlineare, fiind dat un set de exemple. Însă , este greu de stabilit cât de mare ar trebui sa fie rețeaua (câte noduri și câte straturi să conțină) pentru a produce un rezultat cât mai bun. Pe de altă parte, o altă problemă este timpul de antrenare al rețelei, care poate deveni foarte mare pentru rețelele cu multe straturi. În același timp, aceste rețele depind foarte mult de puterea de procesare a calculatorului [12].

1.2 **Rețele neuronale recurente**

**1.2.1 Modul de funcționare**

Spre deosebire de rețelele neuronale tradiționale, cele recurente au , în plus, muchii între noduri adiacente, formând cicluri și introducând noțiunea de timp în model. Aceste muchii se numesc muchii recurente. Pot exista și cicluri de dimensiune 1, adică un nod să aibă o muchie care indică spre el însuși. Astfel, la un moment , nodurile cu o muchie recurentă primesc informație din datele curente, dar și de la valorile anterioare a nodurilor ascunse . Un model de rețea neuronală recurentă este prezentată în figura 2 [9].



**Figura 2. Rețea neuronală recurentă**

Fiind dată o secvență de date de intrare (, o RNN calculează secvența de date de ieșire (, iterând următoarea ecuație [20] :

Aici reprezintă matricea ponderilor dintre datele de intrare și stratul ascuns, iar este matricea ponderilor recurente la momente diferite de timp.

**1.2.2 Antrenarea rețelei**

Antrenarea unei rețele RNN poate să fie foarte dificilă din cauza dependințelor foarte largi. Principalele probleme care apar în antrenarea rețelei sunt *vanishing* și *exploding gradients*. Acestea au loc în timpul propagării erorii. Presupunem că avem că avem o rețea simplă cu un singur nod recurent, iar la momentul rețeaua primește date de intrare și calculează o eroare la momentul *t* . Astfel , contribuția pe care o au datele de intrare de la momentul pentru datele de ieșire de la momentul *t* ori se va apropia de 0 sau va ”exploda” cu cât diferența t - devine mai mare. Probabilitatea ca oricare dintre aceste fenomene să aibă loc depinde de ponderea de pe muchiile recurente (, dar și de funcția de activare. De exemplu, dacă funcția de activare este sigmoid, atunci cel mai probabil valoarea va risca să se apropie foarte mult de 0 și să apară *vanishing gradient*, dar dacă se folosește funcția ReLU max(1,x) , atunci e ușor să ne imaginăm probabilitatea ca valoarea să crească foarte mult și să apară *exploding gradient.*

**1.2.3 Limitări**

Deși inițial rețelele neuronale recurente erau greu de antrenat, putând să ajungă la milioane de parametrii, arhitecturile dezvoltate în ultimii ani au avut performanțe uimitoare în sarcini foarte variate, precum: traduceri, generarea de etichete pentru imagini sau recunoașterea scrisului de mână [9]. Cu toate aceastea , nu este clar cum ar putea fii aplicate RNN pentru probleme ale căror date de intrare și ieșire au relații complicate non-monotone și diferă ca dimensiuni [20]. E important să nu uităm nici de problemele de antrenare ale rețelei.

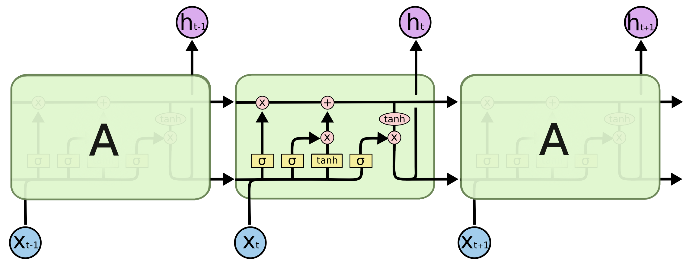
1.3 **Rețele neuronale LSTM**

O RNN simplă este greu de antrenat pentru dependințe foarte lungi, din cauza problemei *vanishing* și *exploding gradients* [9]. Rețelele Long Short-Term Memory (LSTM) sunt cunoscute pentru capacitatea lor de a învăța dependințe temporale largi.

**1.3.1 Componente**

LTSM rezolvă problema *vanishing gradient* introducând un spațiu intermediar de memorie prin intermediul celulei de memorie, care dispune de un sistem format din 3 ”porți”, *forget gate, output gate* și *input gate*. Aceasta înlocuiește nodul simplu din RNN. În Figura 3 [15] este prezent modul de lucru al unei rețele LSTM. După cum reiese și din nume modelul disune de 2 tipuri de memorie. Memoria pe durată lungă (*long term* memory), de care dispune ți o RNN simplă, vine în forma ponderilor calculate la fiecare pas. Memoria de scurtă durată (*short-term* memory) este reprezentată de niște valori temorare care sunt trecute de la un nod la nodurile următoare. În continuare o să detaliez componentele unei celule dintr-o LSTM și rolul fiecăreia.

* *Nodul de intrare (Input node)* este nodul care ia datele de intrare standard de la momentul curent și datele din stratul ascuns din momentele anterioare. Deobicei, suma ponderilor este trecută printr-o funcție *tanh* , dar în lucrări mai vechi se poate observa folosirea funcției sigmoid [9]. Valoarea nodului de intrare o vom nota cu .
* *Input gate,* precum nodul de intrare preia atât informația curentă , cât și cea anterioară, urmând ca acestora să le fie aplicată o funcție sigmoid. Denumirea de ”poartă” vine de la faptul că valoarea acesteia va fii înmulțită cu valoarea altui nod, astfel dacă valoarea este 0, fluxul celuilalt nod va fii întrerupt, iar dacă valoarea este 1, atunci toată informația este trecută la următorul nod. Valoarea pentru *input gate* o vom nota cu .
* *Starea internă* este un nod cu o funcție de activare liniară ce are o muchie recurentă ce indică spre el însuși. Valoarea notată cu a stării interne va fii : .
* *Forget gate* pune la dispoziție un sistem prin care rețeaua învață să sorteze informația pe care o consideră neimportantă din starea internă. Atfel , valoarea stării interne va deveni conform formulei , unde este valoarea obținută de *forget gate.*
* *Output gate* calculează valoarea finală a celulei, înmulțind valoarea ei cu cea a stării interne calculate anterior.



**Figura 3. Structura unei rețele LSTM**

**1.3.2 Mod de funcționare**

Mai întâi, informația anterioară este concatenată cu datele de intrare curente , după care sunt trecute prin *forget gate*. Aceasta decide prin intermediul unei funcții sigmoid ce informație poate sa fie „uitată”, valorile mai apropaite de 1 sunt considerate relevante și sunt trecute mai departe, iar valorile mai apropiate de 0 sunt considerate nerelevante și ”uitate”. În continuare valorile concatenate trec prin *input gate,* care decide , tot prin intermediul unei funcții sigmoid , ce valori ar trebui modificate. În același timp, o funcție tanh generează valosrile candidat din . Valorile candidat se înmulțesc cu rezultatul obținut de la *input gate*, selectând valorile importante din candidat. Următorul pas este modificarea stării celulei prin înmulțirea vechii stării cu rezultatele obținute de la cele 2 ”porți” anterioare. Pasul final este calcularea stării ascunse, care va fii trimisă mai departe prin rețea. Acest lucru este realizat de *output gate*. Starea celulei actuale este normalizată, după care se înmulțește cu valorile candidat din , care a fost trecută printr-o funcție sigmoid. Astfel, se alege informația relevantă care ar trebui sa fie transmisă următoarei celule [9].

**1.3.3 Aplicații**

LSTM au numeroase aplicații în diferite domenii. Printre acestea se numără: generarea de text, recunoașterea scrisului de mână, traducerea automată, captarea imaginilor sau predicții, dar utilizabilitatea lor nu se oprește aici. Rezultatele atinse folosind astfel de rețele sunt multe și foarte fascinante [14].

**CAPITOLUL 2**

**Modelul Markov cu stări ascunse**

**Introducere**

În acest capitol, voi defini componentele care alcătuiesc modelele ascunse markov , precum și modul în care aceastea funcționează.

**2.1 Modelul Markov**

Un model Markov definește un sistem probabilistic prin intermediul stării sistemului și a timpului de observare. Atât timpul, cât și starea sistemuui pot să fie continue sau discrete. În cazul în care sistemul are starea discrete și timpul de observare continuu, se obține un process Markov [16].

**2.1.1 Procese Markov**

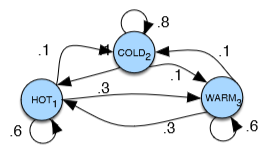
Considerăm un proces stocastic , unde fiecare este o variabilă aleatoare din spațiul de probabilități . este starea modelului la momentul *k.* Se spune despre procesul că are proprietatea Markov dacă:

Pentru A, *k*  [23]. Altfel spus, proprietatea Markov asigură faptul că într-un proces, starea următoare depinde doar de starea prezentă , fără să țină cont de stările anterioare. De exemplu, legile fizicii asigură faptul că mișcarea unei particule într-un pas mic de timp este determinată doar de poziția sa curentă și de velocitatea sa, nu are importanță cum a ajuns în starea respectivă.

**2.1.2 Lanțuri Markov**

Distribuția probabilității condiționale a stării următoare depinde de starea curentă și nu de alte stări anterioare. Adică *s(t)* depinde doare de *s(t-1)* , unde *s(t)* este starea la momentul *t.* Acest caz reprezintă un model Markov de ordinul întâi. În general, dacă starea curentă depinde de ultimele *n* stări anterioare, avem un model markov de ordinul n.

O secvență de evenimente aleatoare ce respectă un model Markov reprezintă un lanț Markov. Aceste secvențe pot să fie cuvinte, etichete sau simboluri reprezentând orice, de exemplu vremea. Figura 4 [5] reprezintă un lanț Markov care asignează probabilitatea unei secvențe de evenimente meteorologice, pentru care vocabularul este definit de cuvintele HOT, COLD și WARM. Stările sunt reprezentate ca noduri în graf, iar tranzițiile ca și muchii, având fiecare o probabilitate cu care se trce dintr-o stare în alta. Suma valorilor muchiilor care ies dintr-un nod trebuie să fie 1.



**Figura 4. Lanț Markov pentru vreme**

Un lanț Markov este specificat de următoarele componente [5]:

* un set de *N* stări,
* o matrice a probabilităților de tranziție, unde reprezintă probabilitatea de a trece din starea *i* în starea *j,* cu proprietatea că
* o distribuție inițială a probabilităților , unde reprezintă probabilitatea ca lanțul Markov să înceapă în starea *i,* pentru care Unele stări *j* pot să aibă , ceea ce înseamnă că aceastea nu pot să fie stări inițiale

**2.2 Model Markov cu stări ascunse (HMM)**

Un lanț Markov este folositor când vrem să calculăm probabilitatea pentru o secvență de evenimente observabile. Însă, în multe cazuri evenimentele sunt ”ascunse” ; nu le putem observa direct. Secvența de evenimente este un lanț Markov; starea următoare depinde doar de starea curentă. Funcția probabilistică pentru acest lanț este un proces stocastic generat de 2 mecanisme: un lanț Markov cu un număr finit de stări și un set de funcții aleatoare asociate cu fiecare stare. Pentru timp discret , se presupune că procesul este într-o anumită stare și o observație este generată de funcția corespunzătoare stării curente. Lanțul Markov iși schimbă după starea conform matricii de probabilități de tranziție. Observatorul vede doar rezultatul obținut de funcția asociată stării și nu poate observa direct stările lanțului Markov; de aici provine și numele de stări *ascunse*.

Ideea de bază este faptul că HMM este un model finit care descrie o distribuție de probabilități pentru o infinitate de posibile secvențe.

HMM poate fii definit de următoarele componente [5]:

* un set de *N* stări,
* o matrice a probabilităților de tranziție, unde
* o secvență de *T* observații, fiecare aparținând unui vocabular
* o secvență de posibile observații , fiecare exprimând probabilitatea ca observația să fie generată din starea *i*
* o distribuție inițială a probabilităților , unde reprezintă probabilitatea ca lanțul Markov să înceapă în starea *i,* pentru care Unele stări *j* pot să aibă , ceea ce înseamnă că aceastea nu pot să fie stări inițiale

Pentru un lanț Markov de ordinul I se pot face 2 presupuneri:

* **presupunerea Markov**: probabilitaea stării curente depinde doar de starea anterioară ,
* **independența rezultatului**: probabilitatea ca rezultatul unei observații depinde doar de starea care produce observația ,

Deobicei, folosind HMM se rezolvă 3 probleme fundamentale [4]. În primul rând, dându-se HMM și o secvență , vrem să aflăm probabilitatea ca HMM să genereze secvența sau pentru a determina secvența optimă de stări pentru a genera secvența. Alt exemplu ar fii, dacă se dă un număr mare de date și vrem să determinăm structura HMM care reprezintă cel mai bine datele. Acestea se mai pot numi și *problema de evaluare, problema de decodare* și , respectiv, *problema de antrenare.*

**2.2.1 Problema de evaluare**

Problema poate fii generalizată astfel: dându-se o HMM [1] și o secvență de observat *O*, să se determine probabilitatea . *A* este vectorul de tranziții , memorând probabilitatea stării *j* care urmează după starea *i* . Probabilitățile tranzițiilor d estare sunt independente de timp.

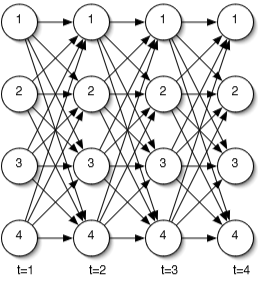
*B* reprezintă vectorul de observare care memorează probabilitatea ca observația *k* să fie produsă de starea *j* , independent de timp.

Altfel spus, vrem să evaluăm cât de bine prezice modelul o secvență. Pentru o secvență și o secvență de stări , probabilitatea pentru secvența de observare este:

Probabilitatea pentru secvența de stări este:

Astfel se poate calcula probabilitatea observațiilor modelului folosinf formula:

Însă, complexitatea acestei formule crește exponențial față de *T*, dar se poate observa că există multe calcule redundante . Pentru a remedia acest fapt se folosește algoritmul *forward*. Acesta presupune implementarea unei unei structuri care seamănă cu o ”împletitură de nuiele” (fiecare nod care o legătură spre fiecare nod de pe următoarea coloană) pentru fiecare stare, care calculează suma tuturor stărilor la momentul anterior de timp, astfel ca în final, ultima coloană să fie echivalentă cu probabilitatea secvenței de observare. O schemă pentru acest model se poate observa în Figura 5 [1]. Valoarea sumei este notată cu α și reprezintă probabilitatea ca secvența parțială să fie generată din starea la momentul . Astfel, α este definit în felul următor:

**

**Figura 5: Model tip ”împletitură de nuiele”**

Pașii algoritmului sunt următorii:

1. **Inițializarea**: .
2. **Recursivitatea**:

Pentru fiecare stare memorează probabilitatea de a ajunge în starea respectivă, observând secvența obținută până în momentul *t*.

1. **Finalizarea**:

**2.2.1 Problema de decodare**

Problema poate fii generalizată; pentru un HMM și o secvență de observații , vrem să aflăm cea mai probabilă secvență de stări [23] Altfel spus, se caută secvența de stări *Q* care are probabilitatea cea mai mare de a genera secvența de observare *O*. Cel mai comun algoritm folosit pentru rezolvarea problemei este *algoritmul Viterbi.* Acesta este similar cu algoritmul Forward de la subcapitolul anterior, diferența constă în faptul că la fiecare pas în loc de suma probabilităților, se calculează maximul, adică se ia starea cu probabilitatea cea mai mare din secvența parțială, care se definește astfel [1]:

Mai mult, algoritmul folosește de ”*backpoints*” , deoarece algoritmul calculează o probabilitate și o cale optimă.

Pașii algoritmului sunt următorii:

* **Inițializarea:** ,

Unde: reprezintă mulțimea de ”*backpoints*”, care va fii folosită în pasul 4 pentru a determina sevența optimă

* **Recusivitatea:**

Pentru fiecare moment de timp *t,* cunoscându-se probabilitatea de a fi în starea *t-1,* se calculează cea mai probabilă rută a stărilor care duce spre starea curentă. În același timp, se memorează câte un ”*backpoint”* pentru stările care fac parte din secvența dorită.

* **Finalizare:**
* **Backtracking pentru aflarea secvenței optime:**

**2.2.3 Problema de antrenare**

Problema poate fii generalizată astfel: fiind cunoscută o secvență de observare *O* și un set de stări HMM posibile, să se învețe parametrii *A* și *B*.

În funcție de felul în care sunt oferite datele de antrenare, există 2 abordări ale problemei: supervizată, când datele conțin și rezultatele așteptate, și nesupervizată, când nu cunoaștem rezultatul pe care ar trebui să îl obținem.

Cea mai simplă abordare supervizată pentru a determina modelul este de a porni de la un set foarte mare de date de antrenare etichetate. De exemplu, se pot defini 2 seturi de etichete [1]:

* este setul de etichete pe care le alipim stărilor din HMM
* este setul de cuvinte pe care le alipim setului de observații

Folosind acest sistem de etichetare, putem afla parametrii modelului folosind următoarele formule:

* **Matricea de tranziții**

Unde reprezintă de câte ori a urmat după în datele de antrenamet

* **Matricea de observare**

Unde reprezintă de câte ori a fost etichetat cu în datele de antrenament

* **Probabilitățile inițiale**

Pentru abordarea nesupervizată, algoritmul standard folosit este algoritmul ”*forward-backword*” sau algoritmul **Baum-Welch**. Algoritmul estimează iterativ probabilitățile inițiale pentru tranziții și observații , după care le folosește pentru a genera probabilități derivate din ce în ce mai bune. Acest lucru se realizează calculând, prima dată, probabilitatea ”*forward*” , împărțind-o după pe toate drumurile care au contribuit la calcularea acestei probabilități. Astfel, se definește probabilitatea ”*backward”* , care reprezintă probabilitatea de a găsi observații de la momentul *t+1* la final. Presupunând că ne aflăm în starea *i* în momentul *t ,* formula probabilității este următoarea [23] :

Pașii pentru a calcula probabilitatea ”*backward”* :

1. **Inițializarea:**
2. **Recursivitatea:**
3. **Finalizarea:**

Estimarea probabilității de tranzție se realizează ușor împărțind numărul estimate de tranziții din *i* în *j* la numărul estimate de tranziții din *i*. Pentru a calcula numărul de tranziții din *i* în *j* se definește o probabilitate care e probabilitatea de a fi în starea *i* la momentul *t* și în starea *j* la momentul *t+1.* Formula pentru această probabilitate este următoarea:

Numărul estimat de tranziții din starea *i* în starea *j* se calcuează făcând suma probabilităților , cu . Numărul de tranziții din starea *i* se calculează realizând suma probabilităților din toate tranzițiile stării. În final formula pentru *a* va arăta în felul următor:

Pasul următor este să se calculeze probabilitatea de observare a unui simbol din vocabular în starea *j.* Aceasta este câtul împărțirii numărului estimate de câte ori eram în starea *j* și observam simbolul la numărul estimate de câte ori eram în starea *j*. Din nou trebuie calculată probabilitatea de a fi în starea *j* la momentul *t*, care se notează cu . Astfel , numărătorul se calculează ca sumă din pentru fiecare moment în care observația este simbolul . Numitorul va fii pur și simplu sumă din în toate momentele . În final, formula pentru *b* va arată în felul următor:

Cunoscând acești termini, se pot reestima probabilitățile de tranziție *A* și de observare *B* dintr-o secvență *O.* Aceste reestimări stau la baza algoritmului ”*forward – backward”*. Pe scurt. modul de lucru al algoritmului Baum-Welch începe cu o aproximare initial a parametrilor HMM , după care se calculează probabilitățile și , urmând ca în pasul următor să se folosească aceste două probabilități pentru a recalcula *A* și *B*.

**2.3 HMM vs LSTM**

**CAPITOLUL 3**

**Abordări folosite în analiza textelor**

**Introducere**

Oamenii au început să scrie informații pe hârtie acum mii de ani. În acest timp, creierul a câștigat foarte multă experiență în înțelegerea limbajului. Putem să citim un articol și să înțelegem ce înseamnă de fapt, putem să compunem un mesaj de la 0 sau să selectăm informația relevantă dintr-un mesaj. Toate aceste lucrui se încearcă să fie automatizate cu ajutorul inteligenței artificiale, iar motivele pentru care am dori asta sunt diverse. Prin procesarea unui text , calculatorul poate să proceseze limbajul nostrum natural. Astfel, un calculator poate să învețe informația relevantă dintr-un text și să ofere un rezumat rapid, să facă o traduceri dintr-o limbă în alta sau să ofere sugestii de completare a unui mesaj. Toate aceste abilități reușesc să facă viața de zi cu zi a fiecărui om mai ușoară și comunicarea cu societatea mai simplă și mai rapidă. În acest articol, o să mă axez , în principal , pe modelarea limbajului și predicția de texte. O să prezint câteva modele din literatură care și-au propus să acopere aceste domenii, folosind diferite rețele neuronale și obținând rezultate relevante.

1.1 **Rețele neuronale recurente**

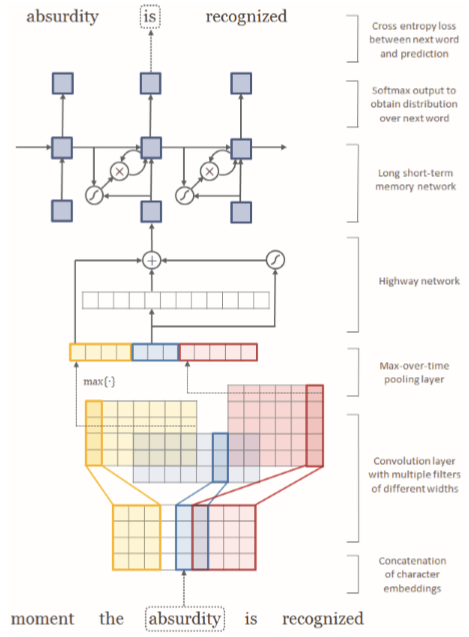
Rețelele neuronale recurente (RNN) au arătat rezultate promițătoare în diferite sarcini ce necesită învățare automată, mai ales atunci când numărul datelor de intrare sau ieșire variază, după cum este cazul în modelarea limbajelor. Ce este interesant de observat este faptul că majoritatea succeselor recente nu s-au realizat cu o rețea neuronală recurentă tradițională, ci , mai degrabă, cu o rețea sofisticată cu noduri ascunse, precum *long short-term memory* (LSTM) [2]. Acestea au aplicații mai ales în recunoașterea vorbirii, generarea de text și traduceri automate.

În continuare , o să prezint abordarea folosită de Kim Yoon et al, pentru predicția textului caracter cu caracter. Modelul propus de ei folosește 2 rețele neuronale: o rețea neuronală de convoluție (CNN) pentru procesarea intrărilor la nivel de caracter și o rețea neuronală recurentă LSTM, care primește rezultatele obținute de CNN [6].

**1.1.1 Arhitectura modelului**

Deși majoritatea experimentelor realizate pentru modelarea limbajului natural combinau *word embeddings* ca intrarecu caracteristici ale analizei caracter cu caracter, modelul propus de Kim Yoon et al folosește ca intrare, rezultatul obținut la nivel de caracter de o CNN cu un singur strat și cu *max-over-time pooling,* acest fapt reducând foarte mult numărul de parametrii.

Arhitectura modelului este prezentată în Figura 1. Aici modelul ia cuvântul *absurdity* ca și date de intrare la momentul current și îl combină cu informația învățată anterior pentru a prezice următorul cuvânt *is*. Primul strat realizează o căutare printer seturile de caractere și le aranjează într-o matrice . După, CNN aplică operații între și mai multe matrici de filtrare . În exemplul current se pot observa 12 astfel de filtre : 3 de lățime 2 (albastru), 4 de lățime 2 (galben) și 5 de lățime 4 (roșu). În următorul pas, operația de *max-over-time pooling* este aplicată pentru a obține o reprezenare de dimensiune fixă a cuvântului, care este trimisă unei rețele intermediare *highway* , care are rolul de optimizare. Rezultatele sunt folosite ca și date de intrare pentru LSTM. În final, se mai aplică o transformare afină și o funcție softmax pentru normalizarea datelor pentru a obține distribuția următorului cuvânt.



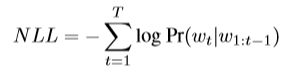
**Figura 1. Arhitectura modelului aplicată pe un exemplu concret**

**1.1.2 Modelarea limbajului folosind RNN**

Rețelele recurente neuronale sunt un tip de rețele potrivite pentru procesarea secvențelor. La fiecare pas *t* , o RNN primește , care este starea neuronului la momentul t și starea ascunsă , care reprezintă informația strânsă până în momentul curent.

Pentru modelul prezentat în [6], se presupune un vocabular de cuvinte de dimensiune fixă, modelarea specifică o distribuție peste , fiind cunoscută secvențele anterioare = []. Un model recurent realizează aceasta aplicând o transformare afină stratului ascuns, urmată de o normalizare softmax.

Având în vedere că = [] este o secvență de cuvinte, antrenamentul rețelei presupune minimizarea funcției *likelihood* (Figura 3 [6]) a secvenței. Acest lucru este deobicei realizat cu propagare in timp (*backpropagation through time).*



**Figura 3. Formula funcției likelihood**

**1.1.3 CNN la nivel de caracter**

În [6], intrările în momentul *t* sunt rezultatele obținute de CNN la nivel de caracter. Modelul folosește *character embeddings*, față de alegerea populară de a folosi *word embeddings.* Acest lucru este avantajos, deoarece orice cuvânt se poate forma, chiar dacă nu face parte din vocabular, cât timp caracterele există în *embedding.* Un alt benefiu este faptul că poate să potrivească cuvinte scrise greșit , emoticoane și se descurcă mai bine cu cuvinte cu frecvență rară [11].

Reprezentarea la nivel de caracter a unui cuvânt *k* format din secvența de caractere [] este data de matricea , unde coloana *j* conține un *character embedding* asociat lui .

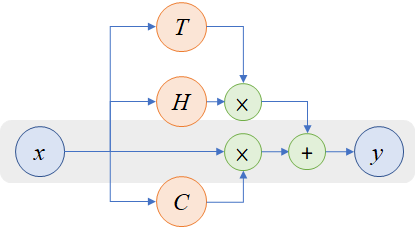
În continuare se aplică *narrow convolution* între și un filtru *H* și *max-over-time,* pentru a capta caracteristica cea mai importantă (cea cu valoarea cea mai mare) pentur fiecare filtru. În esență, un filtru alege o n-gramă în care dimensiunea corespunde cu lățimea filtrului. Acesta este procesul descris pentru un singur filtru, dar modelul abordat folosește mai multe filtre, de lățimi diferite pentru a obține vectorul pentur cuvântul *k*. Dacă avem un total de *h* filtre, atunci este reprezentarea cuvântului *k* , ce va avea rolul de dată de intrare pentru LSTM. În mod normal, pentru procesarea limbajului natural, *h* se alege din intervalul [100, 1000].

**1.1.3 Rețele neuronale highway**

Datele de intrare pentru modelul propus sunt repzentările formate anterior de CNN, însă Kim Yoon et al au observant îmbunătățiri adâugând o rețea intermediară care procesează înainte de intrarea în LSTM. Acea rețea intermediară este o rețea neuronală *highway*, care a fost recent introdusă de Srivastava et al.

Ideea de a implementa acest nou tip de rețea a pornit de la faptul că este dificil de antrenat o rețea cu foarte multe straturi (de ordinal sutelor). Numele de rețele *highway* (autostradă) vine de la faptul că arhitectura lor permite informației să treacă prin mai multe straturi pe ceea ce cercetătorii numesc *information highways* ( autostrăzi de informație ) suferind doar modificări minore. Asemănător cu LSTM și aceste rețele dispun de 2 ”porți”, *transform gate* și *carry gate.* Acestea adaugă 2 transformări neliniare, față de o rețea simplă *feedforward*, având rolul de a exprima ce procent din rezultat este produs din transformarea datelor de intrare și, respectiv, din transportarea acesteia , de unde provin și numele *transform gate* (transformare)și *carry gate* (transportare). Așa cum, într-o rețea simplă, straturile sunt alcătuite din unități de calcul, așa au creat autorii conceptul de bloc pentru rețelele *highway*. Pentru blocul *i* se descriu starea blocului *Hi(x)* și rezultatul obținut prin *transform gate* *Ti(x).* Astfel, rezultatul produs de un bloc are formula următoare: [19].

În Figura 4 [21], este prezentat modul de funcționare al rețelei. *H* reprezintă o funcție de transformare urmată de o funcție de activare, *T* este funcția de transformare din *transform gate* , iar *C* este funcția de transportare din *carry gate*. Un caz particular este ca *C = 1 – T*.



**Figura 4. Circuit de funcționare al rețelei neuronale highway**

**1.1.4 Rezultate**

La testarea modelului s-au folosit 2 seturi de date, unul mai mic (DATA-S) și unul mai mare (DATA-L), pentru fiecare limbă aleasă. În Tabelul 1 [6], sunt trecute limbile alese și dimensiunile datelor folosite. |*V|* este dimensiunea vocabularului de cuvinte, |*C|* este dimensiunea vocabularului de caractere, iar *T* este numarul de simboluri din setul de antrenament. Datele pentru limba engleză au fost luate din *Penn Treebank*, pentru limba arabă de la *News-Commentary corpus*, iar restul sunt luate de la *2013 ACL Workshop on Machine Translation*.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | DATA – S | | | DATA - L | | |
| |*V|* | |*C|* | *T* | |*V|* | |*C|* | *T* |
| Engleză (EN) | 10.000 | 51 | 1 mil. | 60.000 | 197 | 20 mil. |
| Cehă (CZ) | 46.000 | 101 | 1 mil. | 206.000 | 195 | 17 mil. |
| Germană (DE) | 37.000 | 74 | 1 mil. | 339.000 | 260 | 51 mil. |
| Spaniolă (ES) | 27.000 | 72 | 1 mil | 152.000 | 222 | 56 mil. |
| Franceză (FR) | 25.000 | 76 | 1 mil. | 137.000 | 225 | 57 mil. |
| Rusă (RU) | 62.000 | 62 | 1 mil. | 497.000 | 111 | 25 mil. |
| Arabă (AR) | 86.000 | 132 | 4 mil. | - | - | - |

**Tabel 1. Dimensiunea datelor de intrare pentru testare**

În Tabelul 2 [6], sunt prezentate arhitecturile pentru modelele folosind cele 2 seturi de date. Aici, *d* este dimensiunea pentru *character embeddings*, *w* este lățiea filtrelor, *h* este numărul de matrici filtru,dată ca și o funcție pe baza lățimii, *f,g* sunt funcții nonliniare, *l* este numărul de straturi și *m* este numărul de unități ascunse.

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Dimensiune mică | Dimensiune mare |
| CNN | *d* | 15 | 15 |
| *w* | [1,2,3,4,5,6] | [1,2,3,4,5,6,7] |
| *h* | [25 \* *w*] | [min{200,50 \* *w*}] |
| *f* | Tanh | Tanh |
| Highway | *l* | 1 | 2 |
| *g* | ReLU | ReLU |
| LSTM | *l* | 2 | 2 |
| *m* | 300 | 650 |

**Tabel 2. Arhitectura modelelor pentru testare**

În modelarea limbajelor, perplexitatea (PPL) este o măsurătoare care determină cât de bine un model prezice datele de testare. Un model bun este acela care e ce mai puțin ”confuz” când îi sunt prezentate noi date, adică cel cu perplexitatea mai mică.

Pentru a se observa mai clar eficiența modelului propus, se face o comparație cu alte modele existente în literatură. Mai mult, Kim Yoon et al, compară 2 modele LSTM implementate, unul cu 200 de unități ascunse (LSTM-Small) și unul cu 650 de unități ascunse (LSTM-Large). Pentru fiecare dintre acestea s-au folosit, pe rând, ca date de intrare cuvinte (LSTM-Word) sau caractere (LSTM-Char). În Tabelul 3 sunt prezentate rezultatele obținute pe setul de date pentru limba engleză, comparate cu alte modele care au folosit același set de date din *Penn Treebank*. Coloana *Dimensiune* se referă la numărul aproximativ de parametrii al modelului. Aici am trecut doar câteva dintre modelele din literatură comparate deoarece aveau rezultate asemănătoare.

|  |  |  |
| --- | --- | --- |
|  | PPL | Dimensiune |
| LSTM-Word-Small | 97.6 | 5 mil |
| LSTM-Char-Small | 92.3 | 5 mil |
| LSTM-Word-Large | 85.4 | 20 mil |
| LSTM-Char-Large | 78.9 | 19 mil |
|  | | |
| KN-5 (Mikolov et al, 2012) | 141.2 | 2 mil |
| RNN (Mikolov et al, 2012) | 124.7 | 6 mil |
| Deep RNN (Rascanu et al, 2013) | 107.5 | 6 mil |
| genCNN (Wang et al, 2015) | 116.4 | 8 mil |
| LSTM-1 (Zaremba et al, 2014) | 82.7 | 20 mil |
| LSTM-2 (Zaremba et al, 2014) | 78.4 | 52 mil |

**Tabel 3. Rezultate pentru setul de date din Penn Treebank**

Se poate observa că modelul cu mai puține unități ascunse propus de Kim Yoon et al depășește ca și performanță modelele cu aceleași dimensiuni existente, iar cel cu mai multe unități obține aproximativ aceeași perplexitate ca și literatura existentă (Zaremba et al, 2014), cu toate că ca cu aproximativ 60% mai puțini parametrii.

Pentru celelalte limbi, s-a făcut o comparație cu modelul dezvoltat de Botha (2014), care folosește un model KN-4 și un model MLBL., mai puțin pentur limba arabă, unde autorii au antrenat un model KN-4. KN-4 este un model Kneser-Ney, care nu folosește o rețea neuronală , iar MLBL este un model morfologic biliniar. În Tabelul 4 și Tabelul 5 se pot observa rezultatele obținute pentru setul de date de dimensiune mic, respectiv cel de dimensiune mare. Word , Morph și Char sunt modele care primesc ca și input cuvinte,morfeme, și respectiv caractere .

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | DATA-S | | | | | |
| CZ | DE | ES | FR | RU | AR |
| Botha | KN-4 | 545 | 366 | 241 | 274 | 396 | 323 |
| MLBL | 465 | 296 | 200 | 225 | 304 | - |
| 200 unități ascunse | Word | 503 | 305 | 212 | 229 | 352 | 216 |
| Morph | 414 | 278 | 197 | 216 | 290 | 230 |
| Char | 401 | 260 | 182 | 189 | 278 | 196 |
| 650 unități ascunse | Word | 493 | 286 | 200 | 222 | 357 | 172 |
| Morph | 398 | 263 | 177 | 196 | 271 | **148** |
| Char | **371** | **239** | **165** | **184** | **261** | **148** |

**Tabel 4. Rezultatele perplexității pentru setul mic de date**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | DATA-L | | | | | |
| CZ | DE | ES | FR | RU | AR |
| Botha | KN-4 | 862 | 463 | 219 | 243 | 390 | 291 |
| MLBL | 643 | 404 | 203 | 227 | **300** | 273 |
| 200 unități ascunse | Word | 701 | 347 | 186 | 202 | 353 | 236 |
| Morph | 615 | 331 | 189 | 209 | 331 | 233 |
| Char | **578** | **331** | **169** | **190** | 313 | **216** |

**Tabel 5. Rezultatele perplexității pentru setul mare de date**

Este evident din Tabelul 4 că modelul bazat pe caractere are cea mai mare performanță dintre toate celelalte modele prezentate.

Din cauza constrângerilor de memorie, a fost antrenat doar modelul LSTM mai mic (Tabelul 5). Și aici se poate observa superioriatea modelui bazat pe caractere, deposit doar cu puțin la limba rusă de modelul MLBL al lui Botha.

1.2 **Completarea automată a codului**

Modelele pentru procesarea limbajului se folosesc și pentru completarea de cod. În acest domeniu modelele tradiționale statistice nu sunt considerate cele mai potrivite, deoarece nu reușesc să acopere la fel de bine mulțimea mare de simboluri necunoscute din variabile și nume de funcții după cum au realizat Subhasis și Chinmayee [3]. Ei au încercat diferite modele care s-au dovedit mai eficiente, folosind vector de cuvinte pentru modelarea simbolurilor din cod , după cum este prezentat în [13] ,și tehnici de învățare bazate pe rețele neuronale.

În [3] , datele de antrenament și de testare sunt împărțite 50 – 50 și dându-se o secvență de cod, se prezice următorul simbol. Secvențele se aleg aleator dintre liniile complete din setul de date pentru antrenament , simbolul imediat următor fiind cel considerat corect că ar urma în secvența. În continuare , voi prezenta metoda de modelare a simbolurilor și câteva dintre metodele de învățare abordate de Subhasis Das și Chinmayee Shah.

**1.2.1 Modelarea simbolurilor**

Primul pas în modelarea simbolurilor este să se construiască un dicționar cu toate simbolurile sau cuvintele care apar în cod, care va ajuta la prezicerea următorului simbol. Pentru aceasta , similar cu metoda prezentată pentru modelele Markov, se citesc toate liniile din fișier și sunt trecute în dicționar seturile de simboluti alfanumerice și spațiu, seturi de caractere speicale, spațiu și simbolurile pentru a marca o linie nouă. De exemplu, pentru linia de cod: for (i = 0; i < n; i++) { vor fii trecute în dicționar următoarele: for, i, =, 0, ; , i, <, n, ; , i , ++) {. E de remarcat faptul că separarea nu respectă neapărat limbajul, astfel ++) { este tratat ca un singur simbol, deși , în mod normal , conține 3.

După separarea tuturor datelor, se poate observa că dicționarul nu este complet, deoarece pot apărea simboluri noi în cod care încă nu au fost introduse. De asemenea, în codul folosit pentru antrenament pot fii anumite denumire specifice doar acelor fișiere , care nu vor mai apărea deloc. Pentru a rezolva această problemă, autorii articolului au ales să împartă simbolurile în 2 categorii:

1. *Simboluri cheie.* Simbolurile care apar frecvent vor fii catalogate ca și simboluri cheie. În principal, aceste simboluri vor fii simboluri specifice limabjului de programare în care este scris codul.
2. *Simboluri de poziție.* Simbolurile mai rar întâlnite, de exemplu numele de variable sau de funcții, vor fii categorizate ca și simboluri de poziție. Numele lor vine de la faptul că în cod orice simbol care nu este simbol cheie este înlocuit cu un șir de caractere de forma POS\_TOK\_ii , unde ii este poziția simbolului în secvență. Astfel, pentru exemplul de cod dat mai sus s-ar realiza următoarea înlocuire: for, POS\_TOK\_2, =, 0, ; , POS\_TOK\_2, <, n, ; , POS\_TOK\_2 , ++) {. În acest fel, dacă apare un simbol de poziție într-o secvență ca cea de mai sus, se va înlocui șirul de caractere cu simbolul de poziție corespunzător. Dacă simbolul nu a mai apărut până în momentul respectiv în secvență , o să fie etichetat cu UNKNOWN (NECUNOSCUT).

**1.2.2 Metode de învățare**

Subhasis Das și Chinmayee Shah au exemplificat , în [3] , 5 dintre modelele pe care le-au încercat. Dându-se K + W simboluri cheie și de poziție, și cele K + W + 1 simboluri de ieșire s-au construiit modele care au la baza un vector de cuvinte, notați cu , unde [13]. Pasul următor este , ca pentru un set de simboluri , să se calculeze un scor pentru fiecare simbol de ieșire. Acest scor este reprezentat printr-o funcție: Eroarea pentru această metodă este definită prin funcția: . În continuare, o să prezint 2 dintre modelele abordate și rezultatele obținute

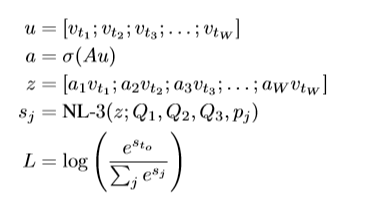
1. Modelul ***feed-forward with soft attention*** presupune folosirea unei rețele neuronale feed-forward,care are asociată fiecărei poziții din vectorul de cuvinte rezultat din modelarea simbolurilor o pondere cu valoare între 0 și 1. Aceasta este rezultatul obținut aplicând funcția sigmoid asupra unei combinații liniare a vectorilor concatenați.

Rețelele feed-forward sunt un element esențial în deep learning. Numele acestora provine de la faptul că informația este transmisă prin intermediul rețelei, de la datele de intrare până la cele de ieșire. Dacă acest model s-ar extinde să conțină conexiuni de feedback în sensul opus parcurgerii normale, atunci ar deveni rețele neuronale recurente [22]. Fiecare neuron calculează suma ponderilor neuronilor de intrare, urmând să aplice o funcție de activare pentru a normaliza această sumă. Pentru acest caz particular, funcția folosită este funcția sigmoid.

Modul de funcționare al acestor rețele este după cum urmează: datele de antrenament sunt transmise prin rețea, obținându-se un rezultat. Acesta este comparat cu rezultatul așteptat, calculându-se o eroare. În continuare se aplică algoritmul backpropagation; eroarea calculată este propagată înapoi în rețea și este folosită pentru a actualiza ponderile fiecărui neuron, astfel încât eroarea să devină cât mai mică. Măsura cu care se modifică fiecare pondere se numește rată de învățare și aceasta este o constată pozitivă [24].

Subhasis și Chinmayee extind acest model, adăugând soft attention. Ideea de a include „atenție” în antrenarea rețelelor neuronale a devenit foarte populară recent , permițând modelelor să realizeze o legătură între 2 entități, de exemplu o imagine și descrierea sa. Luong Minh-Thang et al au realizat un model pe bază de atenție pentru o mașină neuronală care să realizeze traducerea unui text din engleză în română. Experimentând cu 2 tipuri de atenție: globală, care ia în considerare toată stările codificatorului, și locală, care se concentrează pe o porțiune mai mică, au obținut rezultate cu 5.0 puncte BLEU mai multe decât alte modele fără atenție [10]. Rafel și Daniel au implementat un model pentru a rezolva problemele de „adunare” și „multiplicare” din LSTM, care s-a dovedit că poate să ofere rezultate foarte bune, atât pentru secvențe de mărime fixă, cât și pentru secvențe de mărime variabilă, într-un timp foarte scurt ( 254 secunde pentru modelul bazat pe atenție și 917 secunde pentru rețeaua neuronală recurentă) [17].

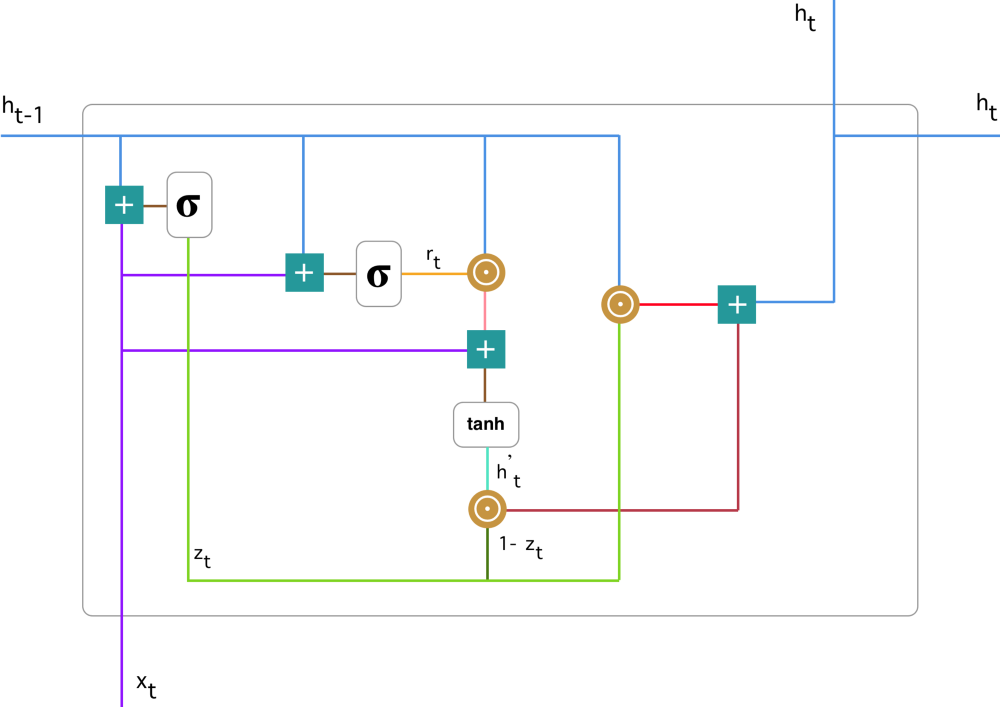
Pentru acest exemplu în particular, modelul va lua forma urmăroare [3]:



Unde NL-k(z;Q1,Q2,Q3,pj) este o funcție care merge de la *u* la pentru un model *feed-forward neural network* prezentat în [3],în care *k* reprezintă numărul de straturi neliniare conținute în model; este atenția calculată; reprezintă concatenarea vectorilor de cuvinte, la care s-au aplicat atențiile calculate.

1. Un alt model încercat a fost unul bazat pe ***rețele neuronale de tip GRU (Gated Recurrent Unit)***. Față de rețelele neuronale clasice, acestea pot decide ce informație ar trebui să fie transmisă spre rezultat. Totodată, pot să fie antrenate să păstreze informații pentru o perioadă foarte lungă de timp sau să uite informație care nu este relevantă pentru predicție. Rețelele GRU sunt foarte asemănătoare cu cele LSTM, deoarece sunt similare ca design și produc rezultate la fel de bune [7].

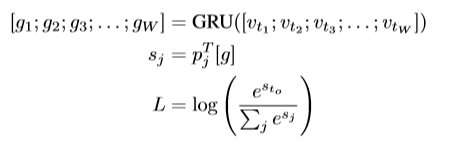
Rețelele GRU dispun de 2 porți de intrare, *update gate* și *reset gate*. În update gate se decide cât din informația curentă se va transmite la timpul t+1. În reset gate se decide cât din informația din trecut poate fii „uitată”. Modul de funcționare al acestor rețele, în general, este precum în Figura 1 [7].



**Figura 1. Rețea neuronală recurentă GRU**

În primul pas, intrarea este concatenată cu informația din cele celule anterioare, notată , fiecare fiind înmulțită cu ponderea sa specifică. Trecând prin *update gate,* li se aplică o funcție de activare sigmoid cu ajutorul căreia se decide câtă din informația inițială ar trebui pasată mai departe. În pasul 2, aceleași valori trec prin *reset gate*, aplicându-li-se funcția de activare sigmoid pentru a decide cât din informație se uită. Valorile mai apropiate de 1 se păstrează, iar cele apropiate de 0 se uită. Pasul următor constă în construirea unei noi celule de memorie , ce se folosește de informația primită de la reset gate pentru a stoca informația relevantă de până acum, folosind funcția *tanh.* Pasul final este de construi memoria finală , care o să fie trecută în continuare prin rețea. Aceasta selectează din informațiile trecute și din memoria curentă datele relevante folosind rezultatul obținut de update gate [7].

Mai specific, în [3] , modelul a avut forma următoare:



Unde, reprezintă datele de ieșire pentru a i-a celulă GRU.

**1.2.3 Rezultate**

În această secțiune , voi prezenta rezultatele obținute folosind cele 2 metode descrise anterior. Acuratețea fiecărui model poate fii văzută în tabelele de mai jos.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Metodă,DimFereas-tră,#Cheie** | **Acuratețe cunoscută** | **Acuratețe absolută** | **Acuratețe chei** | **Acuratețe**  **poziție** | **Top 3** |
| Atenție, 40, 1000 | 67.5 | 56.0 | 79.9 | 48.4 | 83.6 |
| GRU,-,- | - | - | - | - | - |

*Test de acuratețe (%) proiect Linux*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Metodă,DimFereas-tră,#Cheie** | **Acuratețe cunoscută** | **Acuratețe absolută** | **Acuratețe chei** | **Acuratețe**  **poziție** | **Top 3** |
| Atenție, 40, 500 | 47.9 | 40.6 | 64.6 | 17.6 | 66.5 |
| GRU,40, 500 | 41.6 | 35.2 | 64.4 | 4.9 | 59.1 |

*Test de acuratețe (%) folosind biblioteca Twisted din Python*

În prima coloană a tabelului sunt trecute metoda, dimensiunea ferestrei și numărul de cuvinte cheie. Testul proiectului Linux nu are rezultatele raportate pentru GRU , deoarece s-au obținut performanțe foarte slabe.

Acuratețea cunoscută este acuratețea pentru pedicțiile în care următorul simbol este un simbol cheie sau un simbol de poziție din fereastra considerată. Acuratețea absolută este acuratețea pentru toate cazurile , incluzând cazurile în care următorul simbol nu este nici simbol cheie, nici un simbol de poziție deja cunoscut. Acuratețe chei este acuratețea cu care se prezic corect simboluri cheie, iar acuratețe poziție este acuratețea cu care se prezic corect simboluri de poziție. Top 3 reprezintă procentajul cazurilor când următorul simbol se află între primele 3 predicții, ignorând cazurile când apar variabile sau nume de funcții noi.

Se poate observa că că modelul ***feed-forward with soft attention*** a dat rezultate mai bune decât cel folosind GRU. Chiar mai mult, în [3], s-a ajuns la concluzia că acest model a performat mai bine decât toate celelalte modele încercate. Acest lucru se datorează, în special , genericității modelului și a factorului de atenție. Se mai poate observa și că acuratețea pentru simbolurile de poziție este mai mare pentru Linux, un proiect C , decât pentru Python. Însă , predicția pentru top 3 este ridicată pentru ambele, ceea ce arată că programul învață anumite șabloane de limbaj pe care le folosește pentru a obține predicții cât mai bune. De exemplu, atunci când o variabilă apare pentru prima dată (este inițializată) înaintea unei instrucțiuni if , programul prezice că această variabilă va fii folosită în interiourul condiției.

**CAPITOLUL 4**

**[titlu]**

Ar fi de preferat ca aplicația practică să o personalizați (de ex. să îi dați un acronim). Nu denumiți acest capitol doar “Aplicația practică”.

**BIBLIOGRAFIE**

[1] Blunsom, Phil. *Hidden markov models.* Lecture notes*, August* 15.18-19 : 48, 2004

[2] Chung, Junyoung, et al. *Empirical evaluation of gated recurrent neural networks on sequence modeling.* arXiv preprint arXiv:1412.3555 , 2014.

[3] Das Subhasis, Chinmayee Shah. *Contextual Code Completion Using Machine Learning*, 2015.

[4] Eddy, S. R., *Hidden Markov models. Current Opinion in Structural Biology*, 6(3), 361–365. doi:10.1016/s0959-440x(96)80056-x , 1996

[5] Jurafsky, Daniel & Martin H. James , *Speech and Language Processing,* capitolul 8, *Hidden Markov Models,* 2018

[6] Kim, Yoon, et al. *Character-aware neural language models*. In: Thirtieth AAAI Conference on Artificial Intelligence, 2016.

[7] Kostadinov, Simeon, *Understanding GRU Networks.* Available on [towardsdatascience.com/](https://towardsdatascience.com/) , 2017

[8] Kröse, Ben, et al. , *An introduction to neural networks*, 1993

Levinson, S. E. et al ,  *An Introduction to the Application of the Theory of Probabilistic Functions of a Markov Process to Automatic Speech Recognition. Bell System Technical Journal, 62(4), 1035–1074,* 1983

[9] Lipton, Zachary C., John Berkowitz, and Charles Elkan. *A critical review of recurrent neural networks for sequence learning*. arXiv preprint arXiv:1506.00019 , 2015.

[10] Luong, Minh-Thang et al. *Effective approaches to attention-based neural machine translation*. arXiv preprint arXiv:1508.04025 , 2015

# [11] Ma, Edward, *Besides Word Embedding, why you need to know Character Embedding?.* Available on [towardsdatascience.com/](https://towardsdatascience.com/), 2018

[12] Maad M. Mijwel, *Artificial Neural Networks Advantages and Disadvantages.* Available on [https://www.linkedin.com](https://www.linkedin.com/) , 2018

[13] Mikolov, Tomas, et al. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* , 2013.

[14] Moawad, Assaad, *The magic of LSTM neural networks.* Available on [https://medium.com](https://medium.com/) , 2018

[15] Olah, Christopher, *Understanding LSTM Networks.* Available on <https://colah.github.io/>, 2015

[16] Porumbel Valentin, *Recunoaștere vocală minimală pe sisteme embedded folosind inteligența artificială*, Universitatea ”Politehnică” din București, 2016

[17] Raffel, Colin, and Daniel PW Ellis. *Feed-forward networks with attention can solve some long-term memory problems*,  arXiv preprint arXiv:1512.08756 , 2015.

[18] Ratatype , <https://www.ratatype.com/learn/average-typing-speed/> , 2019.

[19] Srivastava, Rupesh Kuma et al. *Highway networks*. arXiv preprint arXiv:1505.00387 ,2015.

[20] Sutskever, Ily et al. *Sequence to sequence learning with neural networks*. In *Advances in neural information processing systems*, 2014.

[21] Tsang, Sik-Ho, *Review: Highway Networks — Gating Function To Highway (Image Classification)*. Available on [towardsdatascience.com/](https://towardsdatascience.com/), 2019

[22]Tushar, Gupta, *Deep Learning: Feedforward Neural Network.* Available on [towardsdatascience.com/](https://towardsdatascience.com/), 2017

[23] van Handel, Ramon. *Hidden markov models* *.* Unpublished lecture notes ,2008.

[24] Vikas, Gupta, *Understanding Feedforward Neural Network,* Available on [learnopencv.com/](https://www.learnopencv.com/) , 2017