Extragerea unui sentiment uman dintr-un text folosind o rețea neuronală recurentă și biblioteca Keras

Paul TEODORESCU

Institutul Național de Cercetare-Dezvoltare în Informatică – ICI București

paul.teodorescu@ici.ro

Rezumat: În acest articol se propune înțelegerea modului în care computerul poate să extragă un sentiment uman simplu (de exemplu: „mi-a plăcut”/„nu mi-a plăcut”), dintr-un text. Practic, computerul învață să includă corect o recenzie de film într-una din cele două categorii (pozitiv/negativ). Așadar, obiectivul propus este estimarea sentimentului uman (interpretat binar 0 sau 1). În acest exercițiu a fost utilizat API-ul Keras construit pe TensorFlow, o mulțime de recenzii de filme luate de pe IMDB și o rețea neuronală recurentă RNN (*Recurent Neuronal Network*) cu celule LSTM (*Long-Short Term Memory*) cu care se memorează cuvintele ce au fost deja întâlnite. Biblioteca Keras furnizează 50.000 de recenzii de filme care sunt deja pre-procesate. Prin hrănirea rețelei neuronale cu aceste texte (25.000 de texte pentru instruire, urmate de alte 25.000 de texte pentru testare), modelul construit de Keras (utilizând relațiile dintre cuvinte) reușește să determine cu exactitate sentimentul uman pozitiv sau negativ, altfel spus polaritatea textului. Utilitatea acestor cercetări de extragere a sentimentului uman este aproape fără sfârșit, de la monitorizarea social-media, măsurarea parametrului numit Vocea Clientului, analiza tweet-urilor și a postărilor Facebook, până la analize de *business* prin analize de texte.

Cuvinte cheie: bibliotecă, vector, tensor, matrice, celule LSTM, variabile, etichete, propagare (înainte și înapoi).

Extracting a human feeling from a text (a natural language processing task called Sentiment Analysis) using a recurrent neural network together with Keras library

Abstract: In this paper, it is proposed to understand how the computer is able to extract a simple human feeling of "liked" or "disliked" from a text. Basically the computer will learn to correctly place a movie review in one of the two categories of positive or negative. We'll see how, starting with input values and output values called labels, the computer begins to learn and correctly recognize the output value (in this case the 0 or 1 digit, zero representing a negative feeling and the one a positive feeling) through a model built on the technique called supervised learning. So the proposed objective is to guess the human feeling (translated by the number 0 or number 1) which is in fact the output value of the model, at a new value of the input, once this model has been known. In this exercise we will use Keras API built on TensorFlow, a set of movie reviews taken from IMDB and a recurring neural network RNN with LSTM (Long-Short Term Memory) cells to preserve the memory of the words that were previously encountered. Keras comes with a set of 50,000 movie reviews that were already pre-processed (this will be explained below). By feeding the neural network with these tens of thousands of texts (25,000 texts for training followed by another 25,000 texts for test), the model built by Keras (using relationships of the words), manages to guess with a good accuracy, the positive or negative human feeling, in other words the polarity of the text. The applications for sentiment analysis are endless starting from social media monitoring and VOC, tweets and facebook posts analyzes, to the business analysis by text analysis.

Keywords: library, vector, tensor, matrix, LSTM cells, labels, variable, back propagation, forward propagation.

1. Introducere

Exercițiul propus – numit „analiza sentimentului uman” - face parte din vasta paletă a cercetărilor în domeniul interacțiunii calculatoului cu limbajul uman. Printre provocările din domeniul NLP (*Natural Language Processing*) se pot aminti: recunoașterea vorbirii, înțelegerea limbajului natural-uman și generarea de limbaj uman. „Sentimentul” se referă la sensul unui cuvânt sau la o secvență de cuvinte și este asociat cu o opinie sau o emoție. „Analiza” este un proces de observare atentă a datelor și extragerea unor deducții. În acest caz a fost utilizată învățarea supervizată pentru a estima dacă recenzia unui film a fost pozitivă sau negativă. În ziua de azi se practică pe scară largă (de la social-media la marketing și medicină) analize de text sau analize lingvistice pentru extragerea și/sau cuantificarea unei stări afective și a informațiilor subiective. De exemplu, companiile folosesc analiza sentimentelor în monitorizarea social-media, monitorizarea

https://doi.org/10.33436/v30i3y202009

brand-ului sau cercetarea de piață. Una dintre problemele de bază în analiza sentimentului (problemă care va fi discutată în continuare) este determinarea polarității unei secvențe de text (în cazul de față este vorba despre o recenzie a unui film). În această problemă, practic de clasificare a unui text, sunt căutate cuvinte ce au încărcătură emoțională cum ar „nervos", „fericit", „trist". Este adevărat că analiza sentimentului uman este o problemă mult mai complexă decât simpla împărțire în două clase, întrucât trebuie ținut seama de contextul în care frazele au fost scrise, de subiectivitatea sau obiectivitatea lor. Acuratețea unui sistem care analizează sentimentul uman ar trebui să ne arate cât de bine se potrivește cu judecata umană și este măsurată în general pe cele două categorii, negative și pozitive. Studiile de statistică făcute de-a lungul timpului asupra acestui subiect arată că scorul sau coeficientul de consens între evaluările sau calificativele evaluatorilor este de 80%. De aceea, un program care reușește să atingă 70% acurațete în clasificarea sentimentului uman, o face aproape la fel de bine ca și oamenii, chiar dacă rezultatul obținut nu este așa de impresionant. Dacă un program de calculator ar avea rezultat 100% în orice situație, oamenii ar fi în dezacord în proporție de 20% asupra oricărui subiect. Pe de altă parte, calculatoarele fac greșeli foarte diferite de greșelile evaluatorilor umani și, de aceea, poate nu ar trebui comparate cele două valori. De exemplu, calculatorul nu poate procesa negațiile, exagerările, glumele sau sarcasmul, lucruri care sunt ușor de rezolvat pentru un cititor uman. În general, utilitatea cercetărilor din mediul academic în acest model uni-dimensional de pozitiv sau negativ este pusă astăzi sub semnul întrebării. Exercițiul merită făcut în scop pedagogic pentru a demonstra ușurința, flexibilitatea și modularitatea API-ului Keras. Fiind o bibliotecă de nivel scăzut, este mult mai flexibilă. Keras oferă în general cam toate funcționalitățile pentru a construi modele de *Deep Learning*, dar nu poate pune la dispoziție tot ce oferă TensorFlow: operații avansate, control, debugging etc. API-ul Keras este integrat în TensorFlow și, astfel, poate fi folosit oricând în cod *tf.keras*. Structura acestui studiu include: descrierea straturilor Keras ca piese fundamentale ale modelului Keras împreună cu descrierea fluxului operațional, felul cum a fost construit modelul, etapele de instruire și testare, vizualizarea performanței și evaluarea modelului. Pentru o analiză mai profundă s-au făcut încercări pe trei modele diferite, iar la final s-a realizat testarea și cu date alese ad-hoc. Rezultatele arată felul în care oamenii au învățat calculatorul să citească un text și să îl clasifice ca pozitiv sau negativ.

2. Datele de lucru și rețeaua aleasă

În acest exercițiu s-a folosit o mulțime de recenzii de filme luate de pe IMDB, set de date oferit de Keras. Acest set are doar două etichete (*labels* sau *target*), și anume: Sentimentul pozitiv și Sentimentul negativ. Cele 25.000 de recenzii de filme sunt deja pre-procesate: fiecare cuvânt este reprezentat numeric, astfel încât fiecare recenzie este codată într-o secvență de indexuri (*tokens*), care sunt de fapt numere întregi. Fiecare cuvânt are un *token* care reprezintă frecvența de utilizare a acelui cuvânt în întregul set de date. Pre-procesarea înseamnă și eliminarea caracterelor non-alfabetice și eliminarea cuvintelor foarte obișnuite cum ar fi: „și", „sau", „de" etc., care fac legătura între cuvintele cheie. Și, bineînțeles, se elimină toate tag-urile HTML. Vocabularul a fost limitat la cuvintele intalnite cel mai des, 20.000 de cuvinte unice, pentru a restricționa dimensionalitatea datelor. Cifra zero nu reflectă nici un cuvânt și a fost utilizată pentru cuvinte necunoscute. Astfel, fiecare recenzie a fost transformată în vectori de numere întregi, urmând ca vectorul de ieșire să fie o valoare de 0 sau 1, care se traduce prin „da, mi-a plăcut filmul" sau „nu, nu mi-a plăcut filmul", ceea ce înseamnă o recenzie pozitivă sau negativă. Așadar, este vorba despre o clasificare a sentimentului uman folosind exemple reale din baza de date IMDB. Cum în înțelegerea limbajului scris este necesar să ținem cont de anumite cuvinte dintr-o frază, se va utiliza o rețea recurentă pentru a memora cuvintele care au fost întâlnite anterior. S-au utilizat celulele LSTM pentru ca datele/informațiile să nu se dilueze în timp, cum s-ar întampla în cazul rețelelor CNN (*Convolutional Neural Networks*). Celulele LSTM ne permit să nu uităm prea repede cuvintele din trecut pentru că ele pot afecta înțelesul frazei.

3. Schema fluxului operațional

În continuare, în Figura 1 se prezintă schema fluxului operațional, urmând ca în Figura 2 să se prezinte explicația acestuia:

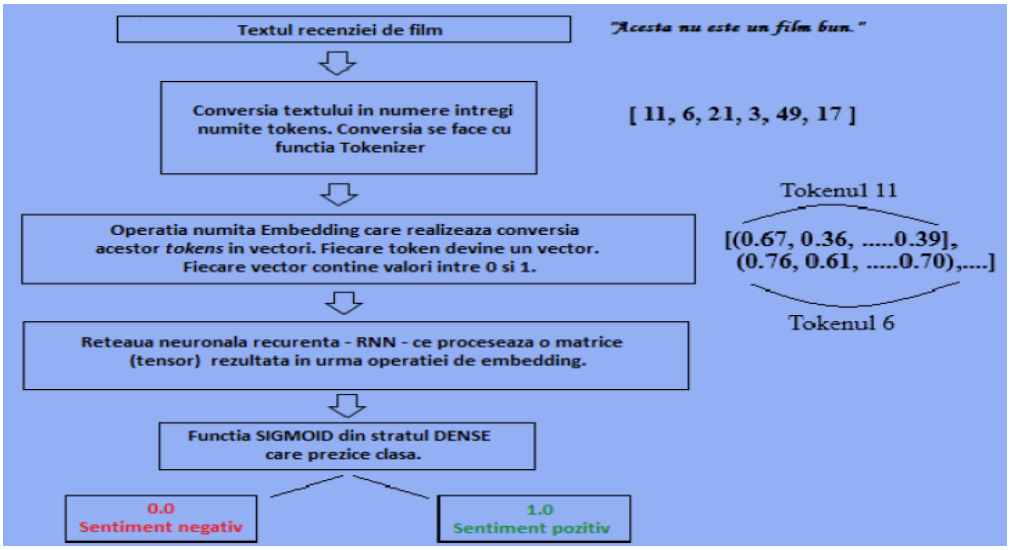

Figura 1. Schema fluxului operațional (desenat pentru o mai clară imagine a pașilor urmați)

Rețelele neuronale nu pot lucra cu text și de aceea se face conversia textului recenziei de film în așa-zișii indecși care sunt valori (numere) întregi. Practic se creează un dicționar în care fiecare cuvânt primește un index. În exemplul din grafic, cuvântului "acesta" îi corespunde tokenul 11, iar cuvântului „nu" cifra 6 etc. Se urmărește, ca acele cuvinte ce au înțeles semantic similar, să fie situate aproape unele de altele în acest spațiu numit embedding. Rețeaua începe să recunoască că există anumite tipare/patterns în care sunt folosite cuvintele împreună, că anumite cuvinte au un înțeles similar și face codificarea în vectori numiți embedding vectors. Rezultatul este un tensor de dimensiune 2 care constituie intrarea în rețeaua neuronală recurentă. Chiar dacă rețelele RNN pot lucra cu secvențe de mărimi diferite, Keras și TensorFlow necesită mărimi fixe (într-un mini set sau batch, datele au aceeași lungime). De aceea, se asigură ca secvențele să fie de aceeași lungime prin funcția pad_sequences. Ieșirea din rețea este apoi procesată de stratul DENSE (ultimul strat), în care funcția Sigmoid furnizează ca ieșire o valoare între 0 și 1. Dacă valoarea este sub 0.5 se consideră 0, adică Sentiment negativ, iar dacă este mai mare ca 0.5 se consideră 1, adică Sentiment pozitiv. Valoarea rezultată este optimizată folosind ADAM. Ultima funcție folosită este binary cross-entropy care folosește ca parametri valoarea calculată de rețea și valoarea reală a clasei din datele de antrenare (training).

Figura 2. Explicarea fluxului operațional

4. Straturile Keras

Se va începe prin importul din biblioteca KERAS a modulului *Sequence*, un modul ce aduce utilități pentru pre-procesarea datelor secvențiale. Apoi este necesar un alt import, descris intr-o instrucțiune ce va permite modelului (ce urmează a se construi) să conțină niște straturi (*layers*) speciale: aceasta se face prin importul modului Sequencial. De-abia acum se pot așeza straturile: un strat/*layer* numit *Embedding*, un alt strat numit *Dense* și un strat de celule LSTM. Binențeles, se face și importul bazei de date IMDB. La acest import se va lucra practic cu 2 tuple, tupla fiind o structură de date imutabilă, o lista specială care este imutabilă.

- x_train, x_test: liste de indexuri, sau secvențe de numere întregi;
- y_train, y_test: liste de numere întregi, 0 sau 1, numite etichete sau *labels*.

TensorFlow a fost instalat înainte de toate, și el va servi ca bază al intregului complex TensorFlow – Keras. Pentru multiprocesarea în siguranță este nevoie de modulul *sequence*:

```
from keras.preprocessing import sequence
```

Alegem modelul Secvențial care nu este altceva decât o stivă liniară de straturi. Apoi cu metoda *add*, se adaugă straturile dorite, cum se va vedea mai jos:

```
from keras.models import Sequential
```

Așadar, se va lucra cu trei straturi diferite: *Dense*, *Embedding* și un strat de celule LSTM (o rețea RNN). Stratul *Dense* este ales deoarece majoritatea bibliotecilor care au tipuri de date specifice calculelor de algebră liniară (adică vectori multidimensionali sau matrice de

n-dimensiuni) au așa-zisa reprezentare *Dense* în care majoritatea valorilor sunt foarte rar zero. Stratul *Dense* este un strat care conține o rețea neuronală conectată dens: acest lucru înseamnă un strat mai profund, mai exact și mai eficient în etapa de instruire (*training*), întrucât conține conexiuni mai scurte între straturile apropiate de intrare și cele de ieșire. Stratul *Embedding* convertește numerele întregi pozitive în vectori denși de lungime fixă:

```
from keras.layers import Dense, Embedding
from keras.layers import LSTM
from keras.datasets import imdb
```

Se încarcă datele, și cele de instruire și cele de test, dar se precizează că se lucrează doar cu 20.000 de cuvinte, cele mai populare. Setul de date pentru instruire va consta în 25.000 de recenzii de film, iar 25.000 de recenzii vor fi pentru test. Vocabularul de 20.000 de cuvinte unice se va folosi și în etapa de instruire (*training*) și în test. După încărcarea acestor cuvinte, se va arunca o privire la prima recenzie (folosind *x_train*[0]) (vezi Figura 3):

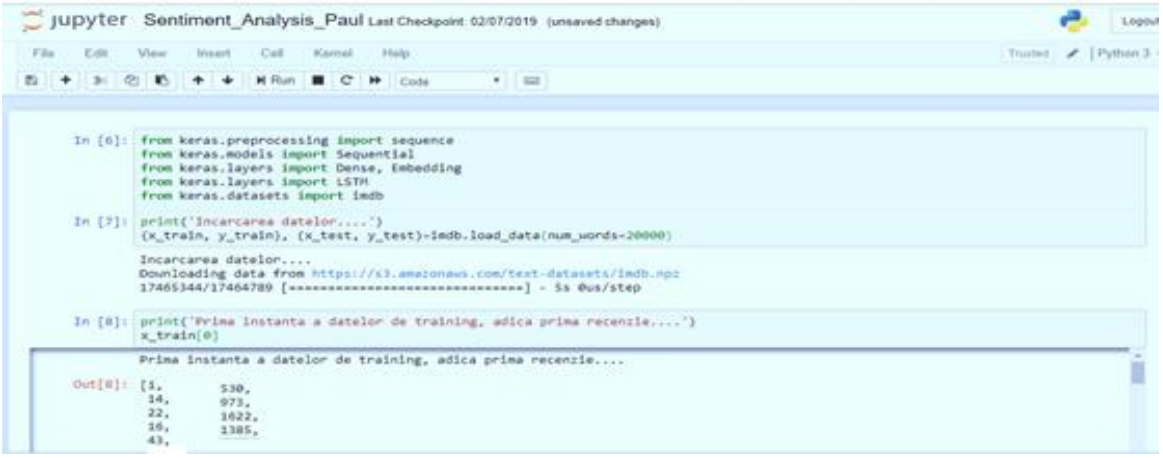

Figura 3. Rezultate lucrând cu Jupiter Notebook – biblioteci folosite și conversii. (*cercetare proprie*)

Keras a convertit cuvintele în numere întregi în operația de pre-procesare făcută de modulul special pe care l-am importat: pentru fiecare intrare/recenzie, textul respectiv a fost transformat într-un vector de numere – fiecare număr reprezentând frecvența de apariție a acelui cuvânt – folosind biblioteca *sklearn*. Având acum o reprezentare vectorială a fiecărui text, se poate continua, întrucât modelul lucrează doar cu numere și nu cu cuvinte. Se încearcă vizualizarea etichetei (sau a țintei) pentru prima recenzie (folosind y_train[0]) (vezi Figura 4).

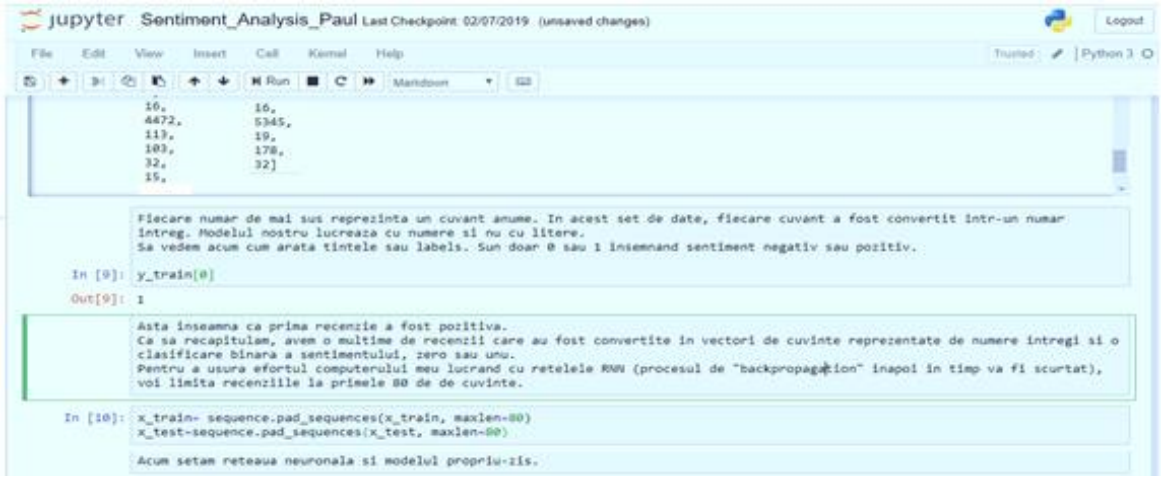

Figura 4. Rezultate lucrând cu Jupiter Notebook – citirea variabilelor. (*cercetare proprie*)

În continuare se va folosi funcția specială *pad_sequences*: este o funcție care trunchează secvențele la aceeași lungime. Practic, transformă lista de secvențe (sau liste de numere de numere întregi) în vectori de două dimensiuni. După aplicarea acestei funcții, se continuă lucrul cu vectori

NumPy de dimensiunea 2D. Prin utilizarea acestor vectori (în detrimentul listelor Python) se asigură un consum de memorie mult mai mic și un timp de rulare mai bun. Folosirea bibliotecilor NumPy asigură instrumente științifice (cum ar fi operații de algebră liniară) pentru calculul vectorilor/*arrays* de N-dimensiuni (sau obiecte de tip vectori multidimensionali). Acești vectori sunt matrice, toate de același tip și conținând colecții de date (cu valori ordonate și neschimbabile) numite tuple de numere întregi ne-negative. În cazul propus aici se lucrează cu vectori/*arrays* de două dimensiuni și forma/*shape* de (25000, 80) pentru datele de instruire (*training*) și (25000, 80) pentru datele de test. Cifra 80 este numărul de cuvinte: primele 80 de cuvinte din recenzie, cu care se lucrează:

```
x_train= sequence.pad_sequences(x_train, maxlen=80)
x_test=sequence.pad_sequences(x_test, maxlen=80)
```

Așadar, întregul proces va fi alimentat cu o sumedenie de recenzii convertite în vectori de numere întregi și cu clasificarea binară a sentimentului – etichetele – pentru fiecare din cele 25.000 de recenzii, din care să învețe rețeaua. Pentru a ușura munca calculatorului, se vor lua doar primele 80 de cuvinte ale acestor recenzii și pentru instruire și pentru test.

5. Modelul

Este timpul pentru construirea modelului propriu-zis. Se reușeste acest lucru cu doar câteva linii de cod unde nu trebuie uitată și adăugarea unei rețele recurente cu o celulă LSTM. Înainte de toate se precizează că se dorește un model secvențial, pentru a fixa topologia rețelei, fizică și logică. Apoi, se precizează o adițională pre-procesare prin stratul numit "*embedding*" care nu face altceva decât să convertească datele de intrare (doar primele 80 de cuvinte dintr-o anumită recenzie), în vectori denși de mărime fixă. Cum celula LSTM este complicată și este greu de explicat ce se întâmplă în interiorul rețelei neuronale, e mai ușor să ne închipuim o pâlnie prin care va trece textul recenziei împreună cu vocabularul explicativ. Vocabularul va spune care cuvinte sunt considerate pozitive și care cuvinte sunt considerate negative. Textul va trece de 128 de ori, valoare aleasă de cel ce face acest exercițiu și care reprezintă chiar numărul de straturi ascunse ale celulei LSTM. O singură linie de cod va construi rețeaua:

```
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
```

Această linie de cod spune că se adaugă o celulă (sau strat) LSTM cu anumite proprietăți:

- va avea 128 de straturi ascunse ce reprezintă de fapt numărul de reutilizări ale celulei LSTM, care astfel va suporta 128 de procese de intrare-ieșire (bucle sau *loop*-uri);
- va avea un *dropout* de 0.2 pentru intrări/ieșiri dar și un *dropout* la procesele sau conexiunile dintre unitățile recurente ale celulei. Practic, înseamnă o renunțare de intrări/ieșiri și o renunțare de conexiuni ale unităților celulei de 20%, pentru a reduce fenomenul de *over-fitting* dar și pentru a îmbunătăți performanța modelului.

Pentru neuronul de ieșire (va fi unul singur) se va folosi o funcție de activare de tip *sigmoid* având în vedere că se tratează o problemă de clasificare binară. Așadar, biblioteca *Keras* face ca lucrurile să pară simple chiar dacă ea face o analiză regresivă logistică și lucrează cu ecuații de regresie logistică (curbe sigmoidale, logaritmi naturali, constanta lui Euler, șanse (*odds*), raportul șanselor (*odds-ratio*) și probabilități s.a.m.d), cu optimizări ale ponderilor folosind *gradient descent* și cu o complicată rețea RNN având o celulă LSTM.

Se optimizează această rețea cu metoda ADAM (*Adaptive Moment Estimation*), iar ca ecuație sau formulă de calculare a erorii (sau a *loss*-ului) se va folosi ecuația *binary_crossentropy*. De altfel, această funcție este folosită implicit în probleme de clasificare binară, adică, acolo unde valorile țintă sunt în setul {0,1}. Funcția *cross_entropy* va calcula un scor care adună mediile diferenței între probabilitatea actuală și cea prezisă pentru clasa de predicție 1 (adică probabilitatea ca ieșirea modelului să fie sentiment pozitiv). Funcția necesită ca stratul de ieșire să fie configurat cu un singur nod și o funcție de activare *sigmoid* pentru a prezice probabilitatea unei ieșiri de

valoare 1. Algoritmul micșorează valoarea de *loss* spre o valoare perfectă a formulei de entropie încrucișată care este 0 (vezi Figura 5).

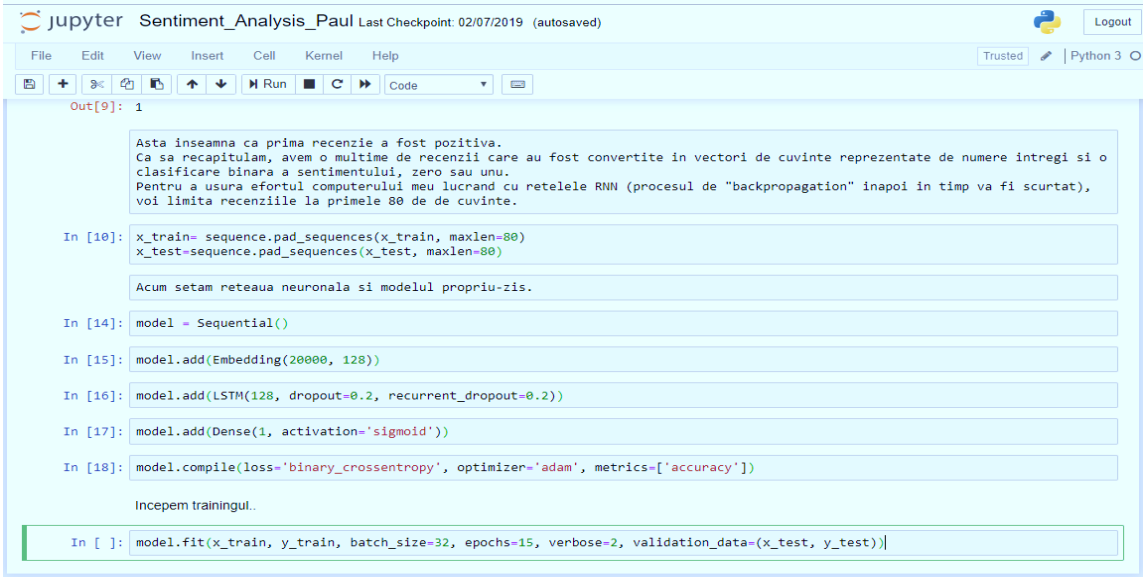

Figura 5. Construcția modelului. (*cercetare proprie*)

6. Etapa de training

Modelul fiind deja construit, se poate începe etapa de învățare a rețelei. Se iau mini seturi de date de câte 32 de recenzii (batch_size=32) pentru a se putea rula confortabil pe un PC fără GPU. Se rulează totul de 15 ori (adică se aleg 15 epoci) și se va face și o validare a datelor (vezi Figura 6):

```
model.fit(x_train, y_train, batch_size=32, epochs=15, verbose=2, validation_data=(x_test, y_test))
```

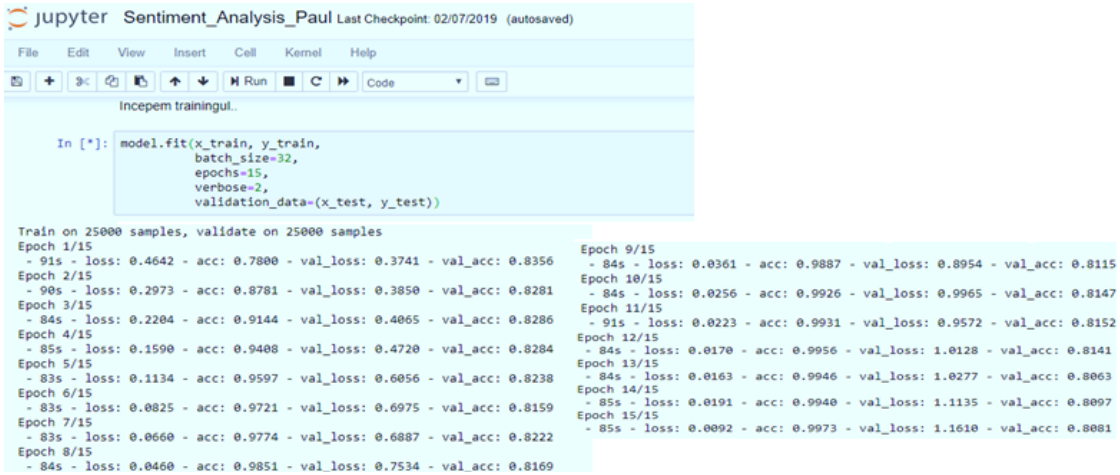

Figura 6. Rezultate lucrând cu Jupiter Notebook – învățarea modelului cu valoarea finală a pierderilor și a acurateții (*imagine de ecran*)

Din rezultatele care apar pe ecran, observăm cum acuratețea – *acc* – începe să crească apropiindu-se de 100%. (la epoca 15, acuratețea este 0.9973). La o acuratețe a predicției de 100% (1.000), modelul construit scoate un rezultat egal cu ținta (*target*). Rezultatele scoase de Keras ne arată pierderile (*loss*) și acuratețea (pe datele de validare, care în acest caz sunt aceleași cu datele de test, validation_data=(x_test, y_test)). Validarea se face la sfârșitul fiecărei epoci pentru a stopa procesul de instruire dacă eroarea, pe datele de validare, începe să crească prea mult. În practică este complicată această procedură pentru că eroarea pe datele de validare ar putea să fluctueze (semn că se produc minime locale) și nu este ușor de recunoscut semnul de începere, cu adevărat, al unui *overfit*. Ideal ar fi să existe un alt set independent de date (diferit de cel de instruire și cel de test) pentru validare, deci pentru o evaluare obiectivă a modelului (*unbiased evaluation*).

7. Etapa de test

După procesul de instruire (*training*), se testează adevărata putere de predicție pe care o are modelul construit, printr-o execuție cu datele de test (vezi Figura 7). Este important de amintit că, această etapă – etapa de test – în învățarea supervizată, este instanța absolută și finală. De asemenea este bine de precizat că, în lumea reală, nu se începe testarea decât după ajustarea modelului, prin „joaca cu hyperparametri“. Se consideră că, în procesul de validare, modelul a fost ajustat (*finetuned*). Ajustarea se poate face de exemplu și prin alegerea unei alte funcții de activare la ieșire în locul funcției *sigmoid*, cum ar fi *softmax*, folosit în probleme de clasificare și care „strivește” ieșirea în intervalul (0,1). Sau se poate folosi un număr mai mare de straturi ascunse ale celulei LSTM, de exemplu 512 în loc de 128. Bineînțeles că pot fi folosite mai multe celule LSTM, așa cum se va vedea în exercițiul al doilea unde au fost folosite două celule LSTM în loc de una.

După execuția cu datele de test (în această etapă nu mai există ținte/*labels* așa cum au fost în etapa de *training*) se poate vedea scorul modelului sau scorul final al acestui model (numit și „pierdere” într-o problemă de clasificare - *classification loss*) care este de fapt prețul plătit pentru o predicție incorectă. Acuratețea rezultată este de 81% pentru modelul acestui exercițiu. Nu pare un rezultat prea impresionant dar nu trebuie uitat că au fost luate în considerare doar primele 80 de cuvinte și, doar cu acestea s-a încercat ghicirea sentimentului autorului. Așadar nu este un rezultat prea prost (vezi Figura 7).

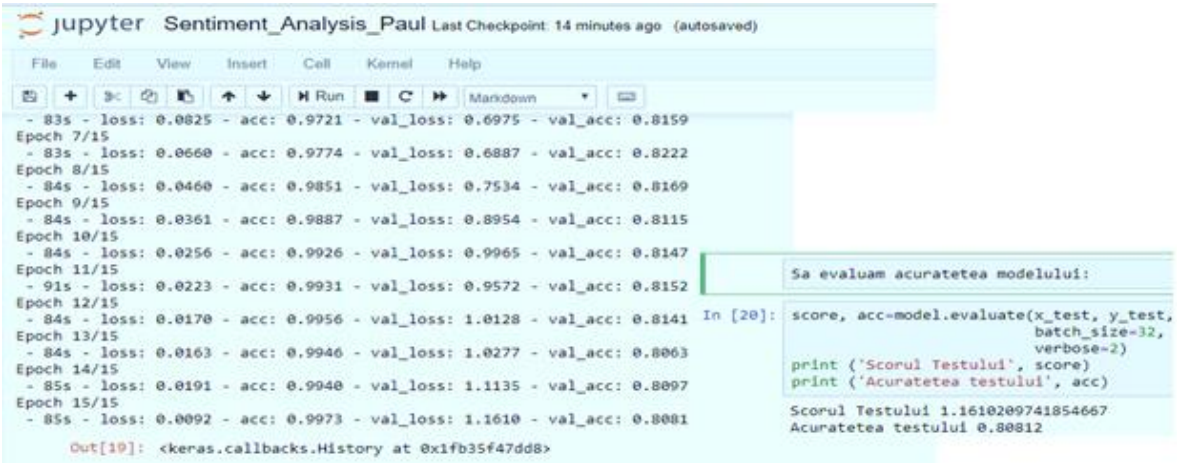

Figura 7. Rezultate lucrând cu Jupiter Notebook – testarea. (*cercetare proprie*)

Toate liniile de cod folosite se pot vedea în Anexa 1.

8. Model nou cu un optimizator diferit

Dacă nu se mai alege ADAM ca optimizator ci SGD (Stochastic Gradient Descent) cu o rată de învățare de 0.01 și momentum de 0.9, codul devine (vezi Figura 8):

```
opt = SGD (lr=0.01, momentum=0.9)
model.compile(loss='binary_crossentropy', optimizer=opt, metrics=[racy'] 'accu)
```

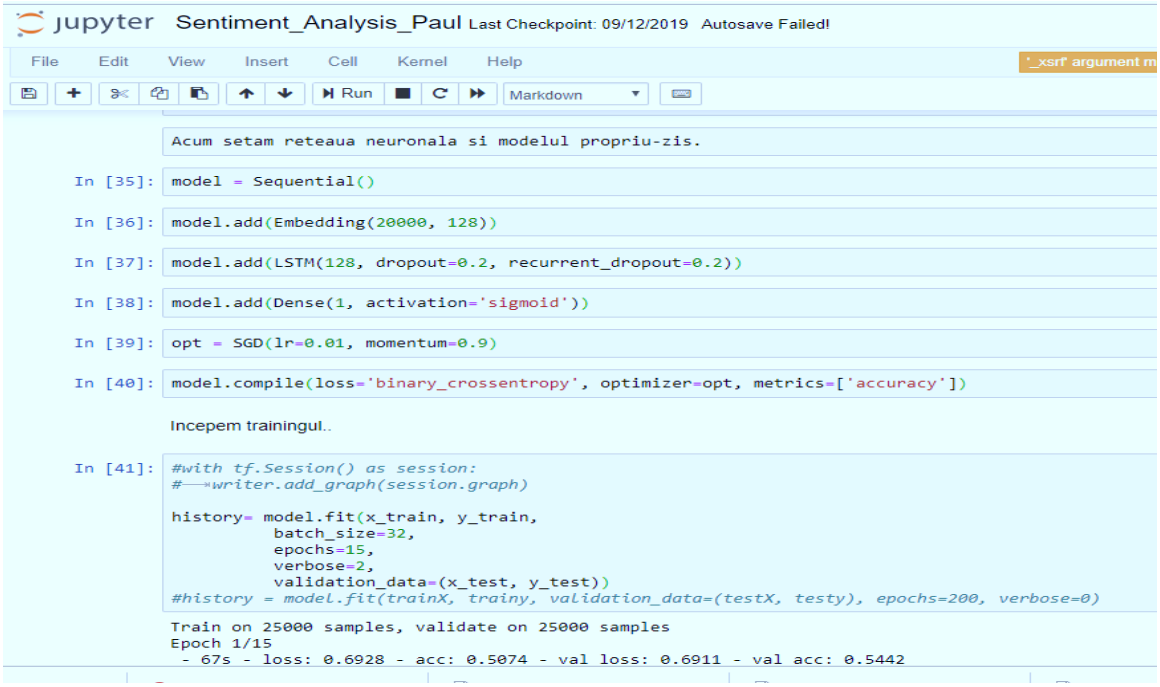

Figura 8. Rezultate lucrând cu Jupiter Notebook – setare model nou cu un alt optimizator. (*cercetare proprie*)

Noile rezultate ale *loss*-ului și ale acurateții sunt (vezi Figura 9):

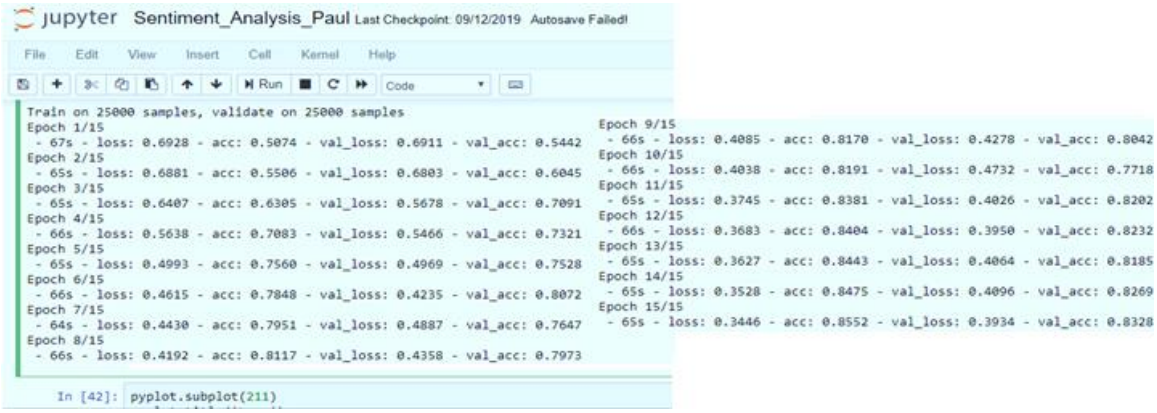

Figura 9. Rezultate lucrând cu Jupiter Notebook – noile rezultate în *training*. (*cercetare proprie*)

Comparând cu rezultatele etapei de instruire pentru modelul anterior din Figura 6, se observă cum eroarea la epoca 15 a crescut iar acuratețea a scăzut. Așadar s-a demonstrat eficiența optimizatorului ADAM care de altfel este o extensie recentă a metodei de optimizare SGD.

9. Vizualizarea performanței algoritmului ales

Pentru a putea vizualiza gradul în care algoritmul ales suferă de un *bias* vis-à-vis de datele alese, se poate opta pentru un grafic numit „curba de învățare”. Această curbă ne poate arăta performanța algoritmului în procesul de învățare de-a lungul timpului odată cu creșterea numărului de repetiții în procesul de învățare (numărul de iterații sau epoci) sau odată cu creșterea numărului de exemple (setul de date). Practic ne arată cum, de-a lungul timpului, rețelele neuronale își optimizează parametrii interni sau altfel spus „începe să învețe”. Tehnica numită „plotare” ne desenează această curbă folosind valori/*metrics* care sunt fie eroarea de predicție (*loss*) fie acuratețea. Se știe că, cu cât *loss*-ul este mai mic, cu atât modelul a învățat mai bine. Un *loss* de 0.0 ne indică că modelul a învățat perfect din datele de instruire (*training*). Curbele acestor valori se desenează pentru ambele seturi de date – instruire și test – ca apoi să se analizeze prin comparație. Așadar, plotarea ne va arăta două grafice, unul pentru eroare/*loss* și altul pentru acuratețe, pentru

ambele seturi de date. Linia albastră este pentru datele de instruire (*training*), iar linia portocalie pentru datele de test (vezi Figura 10).

Figura 10. Curbele de învățare cu datele de *training* și cu datele de test, cu optimizator SGD. (*cercetare proprie*)

În analiza dinamicii acestor curbe, se caută identificarea celor trei tipuri de curbe de învățare: *underfit*, *overfit* și *good fit*. Ne interesează să observăm și descreșterea la o valoare cât mai mică (minimizarea) a valorii de *loss* folosită ca indicator/*metrics*. O potrivire bună (*good fit*) în curba de învățare este de fapt obiectivul tuturor exercițiilor de ML(*Machine Learning*). Curba *good-fit* va arăta cum *loss*-ul descrește (și pe datele de *training* și pe datele de test) până la un punct de stabilitate iar cele două curbe vor tinde să conveargă spre acelasi punct. La un moment dat, distanța dintre ele va fi foarte mică. Acea distanță se numeste „distanța de generalizare".

A continua însă procesul de învățare dincolo de acel punct, va duce la *overfit*. La analiza curbelor, problemele apar atunci când:

- curbele tind să conveargă, dar nu se poate vedea pe graficul plotării momentul în care curbele se apropie una de cealaltă pentru că nu sunt suficiente date sau nu sunt suficiente repetiții de învățare (epoci). În acest caz este clar că trebuie mărit volumul de date de instruire sau mărit numărul de epoci;
- punctul de convergență final al celor două curbe are o eroare mare. În acest caz este clar că nu ajută creșterea volumului de date și nici a numărului de epoci. Este necesar un alt algoritm (mai complex poate) sau o creștere a numărului de caracteristici/*features* ale exemplelor folosite;
- cele două curbe nu au tendința de a converge din cauza curbei de test care se comportă neregulat, impredictibil. În această situație ori se va mări numărul de exemple sau de epoci, ori se vor elimina caracteristicile (*features*) exemplelor folosite, ori se vor schimba niște parametrii cheie ai algoritmului.

De exemplu, analizând Figura 10, se poate vedea cum graficul pentru pierderile/*loss* cu optimizator SGD ne arată un *loss* rezonabil în ambele seturi de date. Iar acuratețea din graficul din dreapta sa, ne arată semne de convergență de-a lungul epocilor. Ideal ar fi obținerea de curbe care încep cu o eroare diferită: cea de test cu o eroare mai mare, iar cea de instruire cu o eroare mai mică. Cu cât crește setul de date de *training* (sau cel al repetițiilor/epocilor), cu atât diferența în spațiu ale celor două curbe trebuie să se reducă până când, la un moment dat, să se apropie de o valoare comună a erorii.

10. Evaluarea modelului cu o singură celulă LSTM

De-a lungul procesului de *training* al modelului, starea curentă a modelului, la fiecare pas, poate fi evaluată. Evaluarea se face după valoarea acurateții, a cărei curbă oglindește performanța modelului. De obicei, acuratețea se calculează după ce procesul de învățare a luat sfârșit, adică, atunci când toți parametrii modelului au fost „învățați" și „fixați". Evaluarea pe setul de date de instruire oferă o idee despre cât de bine **învață** modelul. Se poate face și evaluarea pe datele de test care oferă o idee despre cât de bine **generalizează** modelul (vezi Figura 11):

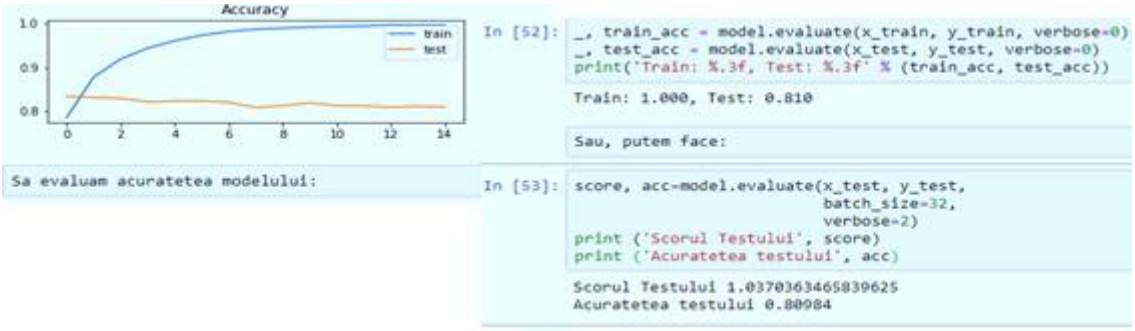

Figura 11. Rezultate lucrând cu Jupiter Notebook – evaluarea acurateții. (*cercetare proprie*)

Așadar, pe datele de instruire s-a obținut o acuratețe de 100%, iar pe datele de test o acuratețe de 81%. Însă nu trebuie uitat că performanța modelului de ML este relativă, iar ideile noastre despre scorul pe care un model bun îl poate atinge nu au sens și nu pot fi interpretate decât în contextul existenței mai multor scoruri obținute pe mai multe modele, instruite cu aceleași date. De aceea, se va construi mai departe un nou model.

11. Utilizarea unui nou model ce folosește o rețea RNN cu două celule LSTM

În continuare, se va construi un nou model cu o rețea recurentă, având două celule LSTM (în loc de unul cum s-a folosit mai sus), pentru a vedea cum se schimbă rezultatele. Codul va fi ușor schimbat, așa cum se vede în tabelul următor:

Modelul cu o singură celulă LSTM	Modelul cu două celule LSTM
model = Sequential() model.add(Embedding(20000, 128)) model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2)) model.add(Dense(1, activation='sigmoid')) model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])	model=Sequential() model.add(Embedding(20000,128)) model.add(dropout(0.3)) model.add(LSTM(128, return_sequences=true, dropout=0.2, recurrent_dropout=0.2)) model.add(LSTM(128, dropout0.3, recurrent_dropuout=0.2)) model.add(Dense(1,activation='sigmoid')) model.compile(loss='binary_crosentropy', optimizer='adam', metrics=['accuracy'])

Aplicând *model.Summary* se poate vedea descrierea modelului nou creat cu cele patru straturi: stratul *Embedding*, urmat de două straturi cu celule LSTM și la final stratul *Dense* (vezi Figura.12).

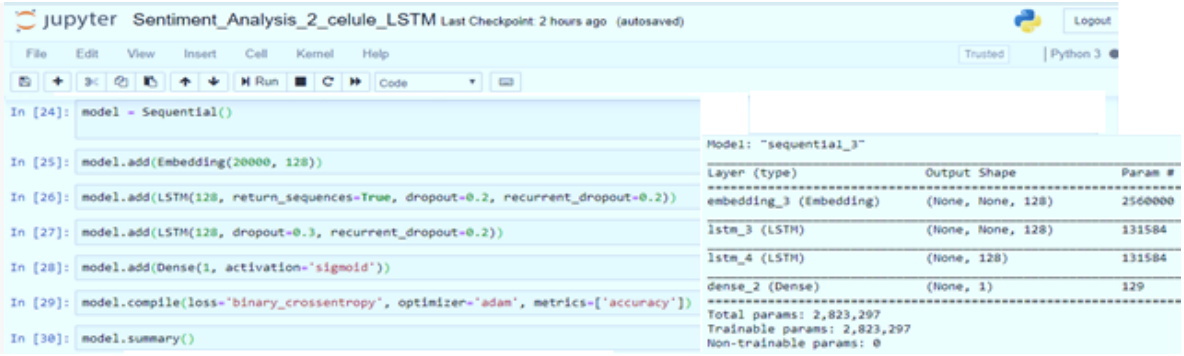

Figura 12. Rezultate lucrând cu Jupiter Notebook – descrierea noului model. (*cercetare proprie*)

Rezultatele lucrând cu două celule LSTM sunt prezentate în Figura 13.

Așa cum a fost precizat mai devreme, scorurile (valorile) celor două *metrics* folosite sunt relative și au sens doar dacă se compară cu valorile mai multor modele. Iată un rezumat comparativ (tabelul de mai jos) al celor două modele ce au folosit aceleași date (recenzii de filme în acest caz) cu mențiunea că pierderea/*loss* nu este un procent (așa cum este acuratețea), ci un număr care reprezintă o evaluare a cât de bine algoritmul a modelat datele sau cât de mică a fost eroarea predicției. Un aspect important al acestei erori (care, în cazul acesta este un *cross-entropy loss*) este acela că poate fi privită și ca o penalizare pentru o predicție greșită.

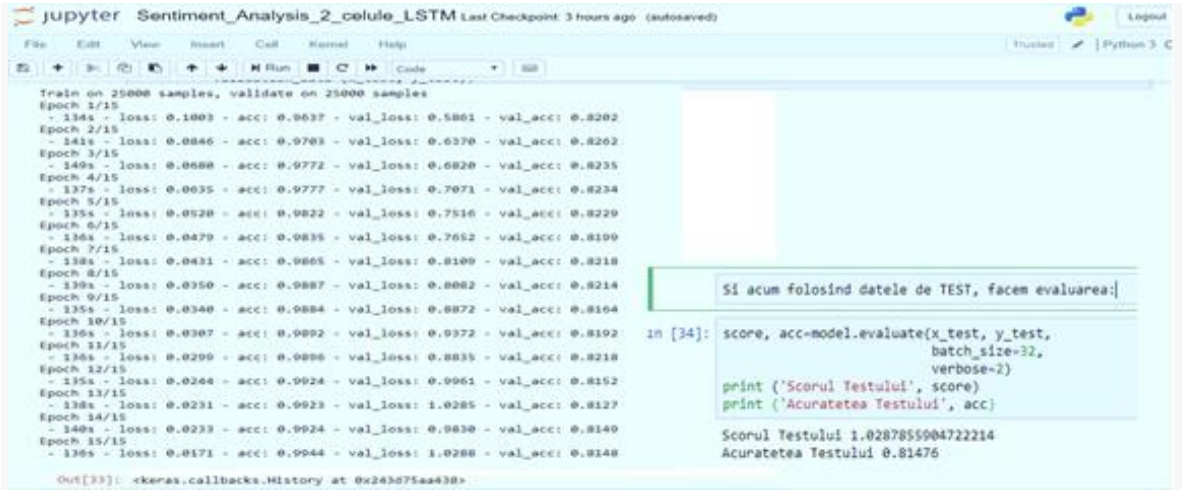

Figura 13. Rezultate lucrând cu Jupiter Notebook – *training* și test. (*cercetare proprie*)

Tip de date	Loss si Acuratețe la modelul cu o singură celulă LSTM	Loss si Acuratețe la modelul cu două celule LSTM
Date de TRAINING	Loss: 0.0171 Acuratețe: 0.994	Loss: 0.009 Acuratețe: 0.997
Date de TEST	Scor/loss: 1.161 Acuratețe: 0.808	Scor/loss: 1.028 Acuratețe: 0.814

Pentru a vizualiza Graful (vezi Figura 14) ne folosim de TensorBoard inserând aceste linii de cod:

```
logs_path = '/tmp/tensorflow/Sentiment_Analysis_2LSTM'
writer = tf.summary.FileWriter(logs_path)
with tf.Session() as session:
    writer.add_graph(session.graph)
```

și se pornește TensorBoard:

```
(py36) C:\Users\paul.teodorescu>tensorboard --logdir=temp/tensorflow/Sentiment_Analysis_2LSTM --port 8892
TensorBoard 1.12.2 at http://Paul-Teodorescu:8892 (Press CTRL+C to quit)
```

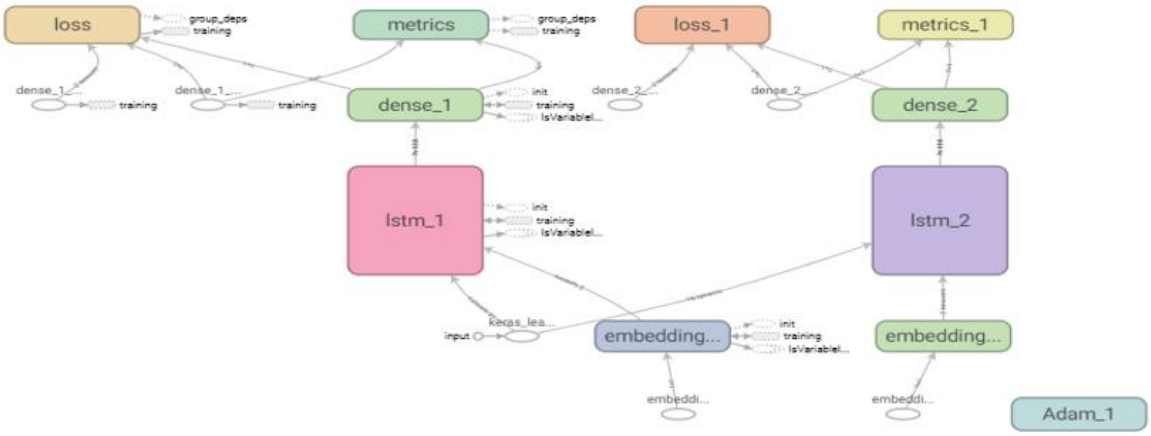

Figura 14. Graful computațional desenat de TensorBoard

12. Testarea modelului cu noi date de test alese ad-hoc

Pentru a se înțelege transformarea recenziilor într-un tensor de două dimensiuni și pentru a se putea demonstra mai ușor clasificarea corectă sau incorectă a rețelei deja construită (se va alege modelul cu o singura celulă LSTM) se vor lua în continuare noi date de test, compuse ad-hoc. Pentru ușurința cititorului și de dragul înțelegerii mecanismului Keras, ele apar aici în românește, dar în exercițiu este folosită limba engleză. Cu cele 8 recenzii scrise ad-hoc, se construiește Vocabularul și apoi se obține un tensor de două dimensiuni/*shape* – o matrice cu 8 linii și 20 de coloane care înseamnă 8 secvențe a câte 20 *tokens* fiecare. Totul este pregătit pentru a folosi modelul construit anterior, având o rețea RNN cu o singură celula LSTM (care a învățat pe datele de instruire oferite de Keras), pentru a prezice sentimentul uman. Se folosește linia de cod *Model.predict.* Așa cum am precizat mai devreme, funcția Sigmoid va scoate la ieșire o valoare între 0 și 1. Dacă este sub 0.5 se va considera Sentiment negativ, iar dacă este peste 0.5 înseamnă Sentiment pozitiv.

Din 8 recenzii, doar una a fost clasificată greșit (vezi Figura 15):

Figura 15. Rezultatele finale ale predicției modelului cu o celulă LSTM

13. Concluzie

Cu acest exercițiu s-a reușit construcția unei rețele neuronale capabilă să citească o recenzie de film și să extragă un fel de înțeles din cuvintele folosite. Altfel spus, o mașină a fost învățată să culeagă o secvență de cuvinte dintr-o recenzie scrisă de un om și să o clasifice ca pozitivă sau negativă. Calculatorul a fost învățat să citească ceea ce este un lucru impresionant. În revistele, articolele și tutorialele online există o multitudine de exerciții similare, dar cum fiecare model construit implică zeci de variabile și hiperparametri, fiecare exercițiu arată diferit. Este regretabil că nu se explică bine calea urmată și motivele pentru care se fac toți acești pași. Articolul prezentat încearcă să umple acest gol și chiar mai mult, să compare rezultatele obținute cu două modele de rețele neuronale diferite. Nu s-a căutat obținerea unor rezultate de excepție care să concureze cu cele ale cercetătorilor străini din lumea ML, dar măcar să fie un ghid ajutător pentru tinerii informaticieni români. S-a lucrat cu platforma Keras (având TensorFlow în spatele ei), ceea ce a permis scrierea unui număr mic de linii de cod. Având date bune de instruire (*training*), rețeaua neuronală a făcut singură totul. Acest exercițiu a fost un bun exemplu de utilizare a rețelelor RNN în ceea ce s-ar numi puțin exagerat „extragerea sentimentului uman dintr-un text”. Totodată a fost elocvent pentru a demonstra forța platformei Keras.

Este interesant de menționat faptul că recenziile (care sunt de fapt scurte texte ce exprimă o părere despre un film sau despre un produs) joacă un rol vital în succesul unui film sau în vânzarea unui produs. Oamenii analizează blog-uri, examinează recenziile unui site cum este IMDB pentru a ști rating-ul filmului, echipa de filmare și alte evaluări. Deci nu numai informația transmisă din gură în gură aduce audiența la cinema: recenziile joacă un rol important în această privință. Social-media a devenit o parte importantă a vieții umane. Oamenii doresc să-și împărtășească toate evenimentele din viața lor pe social-media, care a devenit astfel un tablou al mândriei și stimei de sine al celui ce postează fotografii, texte, clipuri video etc. Textele joacă un rol esențial în diseminarea informației, acolo găsindu-se opiniile oamenilor în cele mai la modă subiecte, în

politică, filme etc. Aceste texte scurte, postate pe rețelele de socializare au căpătat o mare importanță prin simplitatea și eficiența lor în influențarea mulțimii. Ele chiar sunt folosite de către motoarele de căutare. De aceea devine importantă obținerea de idei ce sumarizează acel text și extragerea unui sentiment din el. Analiza sentimentelor este extrem de utilă în monitorizarea rețelelor de socializare, deoarece ele permit obținerea unei imagini de ansamblu asupra opiniei publice din spatele anumitor subiecte. Instrumentele de monitorizare a rețelelor de socializare (ce conțin algoritmi și modele ca cele studiate în acest articol) fac acest proces, în timp real, mai rapid și mai ușor ca niciodată. A fost demonstrat că schimbările de sentimente în mass-media sunt corelate cu schimbările pe piața bursieră. Organizațiile au devenit mai conștiente de aplicațiile ce fac analiza sentimentelor pe piața în care activează și, prin alimentarea cu datele proprii, au ajutat la creșterea serviciilor care folosesc o tehnologie IT ca cea descrisă mai sus, ce furnizează extragerea sentimentului uman. În acest fel, deciziile comercianților și ale analiștilor financiari au fost mult îmbunătățite.

Anexa 1

```
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense, Embedding
from keras.layers import LSTM
from keras.datasets import imdb
print('Incarcarea datelor....')
(x_train, y_train), (x_test, y_test)=imdb.load_data(num_words=20000)
print('Prima instanta a datelor de training, adica prima recenzie....')
x_train[0]
y_train[0]
x_train= sequence.pad_sequences(x_train, maxlen=80)
x_test=sequence.pad_sequences(x_test, maxlen=80)
model = Sequential()
model.add(Embedding(20000, 128))
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(x_train, y_train,
          batch_size=32,
          epochs=15,
          verbose=2,
          validation_data=(x_test, y_test))
```

BIBLIOGRAFIE

1. Agarwal, B., Nayak R., Mittal, N., Patnaik, S. (2020). *Deep Learning – best approaches for sentiment analysis*.
2. Bird, S., Klein, E., Loper, E. (2009). *Natural Language Processing with Python*.
3. Bo, Pang, Lillian, Lee, Shivakumar, Vaithyanathan (2002). *Thumbs up? Sentiment Classification using Machine Learning Techniques*.

4. Brandwatch.com – *Understanding sentiment analysis*.
5. Géron, A. (2019). *Hands-on Machine Learning with Keras and TensorFlow*.
6. Github.com – *Public repositories regarding sentiment analysis*.
7. Github.com – *Sentiment-Analysis* (code samples).
8. Gulli, A., Pal, S. (2017*). Chapter 6: Recurrent Neural Network* — RNN.
9. https://medium.com/datadriveninvestor/deep-learning-lstm-for-sentiment-analysis-in-tensorflow-with-keras-api-92e62cde7626.
10. https://www.kaggle.com/ngyptr/lstm-sentiment-analysis-keras.
11. Liu, Bing (2015*). Sentiment Analysis: Mining Opinions, Sentiments, and Emotions*.
12. MonkeyLearn Inc. (2019). *Sentiment Analysis*.
13. Srinivasa-Desikan, B. (2018). *Natural Language Processing and Computational Linguistics – A practical*.
14. Troussas, C., Virvou, M. (2020). *Advances in Social Networking – based Learning*.
15. *Udemy Courses*. Sentiment Analysis with LSTM and Keras in Python.
16. Vrejoiu, M. H. (2019). *Reţele neuronale convoluţionale, Big Data şi Deep Learning în analiza automată de imagini.* Revista Română de Informatică şi Automatică (Romanian Journal of Information Technology and Automatic Control), ISSN 1220-1758, vol. 29(1), pp. 91-114.
17. Yenter, A., Verma, A. (2027). *Deep CNN-LSTM for IMDB reviews*.

Paul TEODORESCU de profesie inginer mecanic, a lucrat în domeniul IT încă din anul 1990, în Romania și în Canada. Primul său proiect a fost de diagnosticare a tramvaielor electrice cu ajutorul calculatoarelor. S-a specializat în dezvoltarea de software pentru aplicații inginerești și în baze de date Oracle. În Canada a proiectat si dezvoltat un masiv proces de data warehousing, real-time 24/7. Actualmente este pasionat de Inteligența Artificială, Machine Learning, Deep Learning, Natural Language Processing, rețele neuronale artificiale.

Paul TEODORESCU a mechanical engineer by profession but with a solid IT background (graduating the Computer Science high school), he has been working in the IT field since 1990, in Romania and Canada. His first project in Romania was about the electric tram diagnostics using computers. Throughout his Canadian career, he specialized in software development for engineering applications using especially Oracle databases. One of the most important projects was to design and develop a massive data warehousing process, real-time 24/7. He is currently passionate of Artificial Intelligence, Machine Learning, Deep Learning, Natural Language Processing, artificial neural networks.