Universitatea „Politehnica” din București
Facultatea de Electronică, Telecomunicații și Tehnologia Infomației

Aplicații ale rețelelor neurale pentru analiza seriilor temporale

PROIECT DE DIPLOMĂ

Prezentat ca cerință parțială pentru obținerea titlului de *Inginer* în domeniul

Electronică Aplicată și Ingineria Informației

Programul de studii: Ingineria Informației

Conducători științifici:
Ș.l. Dr. Ing. Anamaria RĂDOI,
Prof. Dr. Ing. Corneliu BURILEANU

Student:
Simion Alexandru-Marius

București
2018

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

Anexa 1

TEMA PROIECTULUI DE DIPLOMĂ
a studentului **SIMION I. Alexandru-Marius , 444A**

1. Titlul temei: Aplicaţii ale reţelelor neurale pentru analiza seriilor temporale

2. Descrierea contribuţiei originale a studentului (în afara părţii de documentare):
Proiectul de diploma constă în implementarea unor algoritmi de analiză a seriilor temporale folosind reţele neurale. Scopul proiectului este acela de a crea un sistem automat de analiză a seriilor temporale și de predicție a anumitor evenimente. În primul rând, studentul va crea o bază de date formată din serii temporale (de exemplu, date seismice). Totodată, studentul va realiza o sinteză a algorimilor existenți în literatură pentru analiza seriilor temporale, cu accent pe algoritmii ce au la bază rețele neurale. Un număr de minim 3 arhitecturi de rețele neurale vor fi propuse, cu ajutorul cărora vor fi efectuate experimente pe baza de date creată. Rezultatele obținute vor fi comparate și va fi analizat modul în care arhitecturile propuse permit o cât mai bună estimare a unui eventual eveniment. În acest sens, vor fi măsurate erorile de predicție obținute pentru fiecare arhitectură în parte.

3. Resurse folosite la dezvoltarea proiectului:
Limbaje de programare: Python si C++ Biblioteci: Tensorflow, Theano, MatplotLIB, Numpy, Scipy

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

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

6. Data înregistrării temei: 2017-11-28 08:17:27

Conducător(i) lucrare,
Ş. L. Dr. Ing. Anamaria RĂDOI
semnătura: .....

Student,
semnătura: .....

Dl. Prof. Dr. Ing. Corneliu Burileanu
semnătura: .....

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

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

Cod Validare: **58a44f9929**

Anexa 5

Declaraţie de onestitate academică

Prin prezenta declar că lucrarea cu titlul "*Aplicații ale rețelelor neurale pentru serii temporale*", prezentată în cadrul Facultăţii de Electronică, Telecomunicaţii şi Tehnologia Informaţiei a Universităţii "Politehnica" din Bucureşti ca cerinţă parţială pentru obţinerea titlului de *Inginer* în domeniul *Calculatoare și Tehnologia Informației*, programul de studii *Ingineria Informației* este scrisă de mine şi nu a mai fost prezentată niciodată la o facultate sau instituţie de învăţământ superior din ţară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referinţe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar şi în traducere proprie din altă limbă, sunt scrise între ghilimele şi fac referinţă la sursă. Reformularea în cuvinte proprii a textelor scrise de către alţi autori face referinţă la sursă. Înţeleg că plagiatul constituie infracţiune şi se sancţionează conform legilor în vigoare.

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

Bucureşti, *29.06.2018*

Absolvent *Alexandru-Marius Simion*

(semnătura în original)

Cuprins

Listă Figuri

Listă Tabele

Introducere

Fizicianul Neils Bohr a spus o dată: „Predicția este foarte dificilă, mai ales în ceea ce privește viitorul”, astfel previziunea reprezintă o sarcină extrem de dificilă.

Deși reprezintă una dintre provocările științei moderne, atunci când este efectuată corect, aceasta poate avea un impact enorm.

Predicțiile joacă un rol esențial în cadrul proceselor decizionale din cadrul unei companii aeriene cum ar fi alocarea de resurse, a unui buget, în cazul undelor seismice permit detectarea anomaliilor mișcarilor suprafeței solului, iar în cazul datelor din cadrul cursului valutar BNR, acestea permit crearea unor previziuni monetare și ajută guvernanții să adopte legislații cu privire la politica monetară.

Rețelele neurale sunt folosite adesea pentru predicție utilizând învățarea automată a dependențelor din cadrul datelor măsurate, fără a necesita informații adiționale (așa cum este cazul regresiei) având o rată de succes variabilă de la caz la caz. O rețea neurală poate fi reprezentată asemenea unei cutii negre, ce are drept scop memorarea dependențelor ascunse pentru a le putea utiliza în prezicerea datelor ulterioare.

Predicțiile sunt realizate pe mai multe baze de date (unde seismice, vânzări bilete în cadrul unei companii aeriene, curs valutar BNR) constituind fiecare o mulțime de observații măsurate în perioade succesive de timp, numite serii temporale, prin intermediul rețelelor neurale, mai exact prin intermediul LSTM-urilor, fapt ce reprezinta scopul acestei lucrări.

Domeniul rețelelor neurale își are originea în modelarea sistemelor biologice neurale, în urma dezvoltării acestuia pe scară largă, reliefându-și importanța în inginerie, mai ales în procedeele de învățare mecanică (engl.: Machine Learning)..

Rețelele neurale reprezintă modelarea matematică a procesului de învățare a unei ființe ce dispune de un număr de neuroni, aceștia aflându-se în cadrul unui organ ce permite controlul funcțiilor vitale.

Rețelele neurale recurente reprezintă un tip aparte de rețele neurale, făcând parte din categoria celor mai puternici și robuști algoritmi existenți, dispunând de memorie internă și având aplicabilitate vastă, printre care se pot aminti: serii temporale, date financiare, recunoașterea vocii și a scrisului.

LSTM-urile sau Rețelele cu memorie de scurtă durată persistentă reprezintă o categorie aparte a rețelelor neurale recurente, ce dispun de o memorie extinsă, fapt ce le face ideale pentru utilizarea în diferite experiențe ce se desfășoară pe o perioadă mai lungă de timp, asemenea aplicațiilor ce vor fi prezentate ulterior.

1. Rețele neurale

Unitatea computațională de a bază a creierului este reprezentată de neuron. În sistemul nervos uman se întâlnesc aproximativ 86 de miliarde de neuroni care sunt conectați cu aproximativ 10^{14} - 10^{15} sinapse. În Figura 1.1.1.1 este reprezentat un neuron, iar în Figura 1.1.1.2, un model matematic al acestuia.

Fiecare neuron primește semnal de intrare de la dendridele sale și produce un semnal de ieșire de-a lungul axonului său. Acesta din urmă se ramifică și se conectează prin intermediul sinapselor sale la dendridele altor neuroni.

În modelul computațional al neuronului, semnalul ce se deplasează de-a lungul axonilor (x_0) interacționează în mod multiplicativ **(w_0x_0)** cu dendridele unui alt neuron bazându-se pe puterea sinaptică din acea sinapsă **(w_0)**.

Principiul este acela că puterile sinaptice (ponderile *w*) pot fi învățate și acestea controlează cât de mult influențează un neuron pe altul (ponderi pozitive sau negative). În modelul de bază, dendridele transportă semnalul către corpul celulei, unde acestea sunt însumate. Dacă suma finală este mai mare de un anumit prag, neuronul trimite un puls de-a lungul axonului său.[5]

În cadrul modelului computațional, presupunem că timpii pulsurilor nu contează și că doar frecvența de transmitere a acestora ofera informație. Bazându-se pe această interpretare a ratei de informație, se va modela rata de transmisie a neuronului cu o funcție de activare $\boldsymbol{f}$, ce reprezintă frevența pulsurilor de-a lungul axonului.

Din punct de vedere istoric, o alegere adesea întâlnită funcției de activare este **funcția sigmoid σ**, deoarece aceasta ia ca intrare o valoare reală (puterea semnalului după însumare) și o introduce într-un interval [0,1].

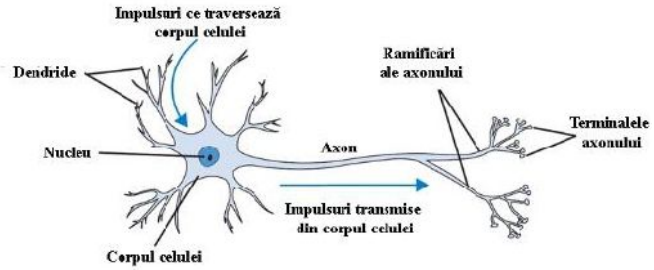


Figura 1.1.1.1 Neuron biologic

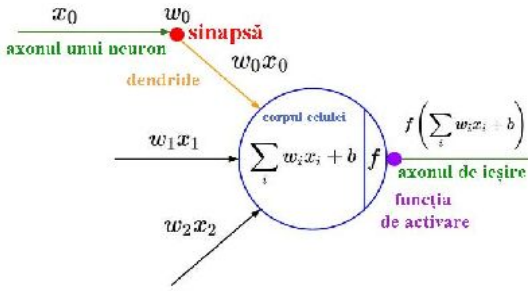


Figura 1.1.1.2 Modelul matematic al neuronului

Cu alte cuvinte, fiecare neuron produce o înmulțire punct la punct între intrarea și ponderile sale, adaugă un bias și apoi aplică neliniaritatea (sau funcția de activare), în acest caz fiind reprezentată de funcția sigmoid $\boldsymbol{\sigma(x) = \frac{1}{1+e^{-x}}}$

Modelul grosier al neuronului este unul simplificat. Spre exemplu, există o mulțime de neuroni, fiecare având proprietăți diferite. Dendridele din neronii biologici execută operații neliniare complexe. Sinapsele nu reprezintă o singură pondere, ele reprezintă un sistem dinamic complex și neliniar. Timpii exacți de producere a pulsurilor în multe sisteme au o importanță majoră, sugerând că aproximarea ratei informației nu este suficientă.

1.1 Neuronul – clasificator liniar

Asemenea clasificatorilor liniari, un neuron are capacitatea de a-i „plăcea” (funcția de activare în 1) sau „displăcea” (funcția de activare în 0) anumite regiuni din spațiul de intrare. Prin urmare, cu o funcție de pierdere adecvată aplicată asupra ieșirii neuronului, un neuron se poate transforma într-un clasificator liniar. [5]

1.1.1 Clasificatorul Softmax binar

Vom considera $\boldsymbol{\sigma(\sum_i w_i x_i + b)}$ ca fiind probabilitatea uneia din clasele $\boldsymbol{P(y_i = 1|x_i; w)}$. Probabilitatea unei alte clase va fi $\boldsymbol{P(y_i = 0|x_i; w) = 1 - P(y_i = 1|x_i; w)}$ deoarece suma acestora trebuie să fie 1. Folosind această interpretare, se poate formula pierderea de entropie încrucișată , iar prin optimizarea acesteia putem ajunge la un clasificator binar Softmax (cunoscut ca și ***regresie logistică***). Din moment ce funcția sigmoid este restrânsă la intervalul [0,1], predicțiile acestui clasificator sunt bazate pe faptul că ieșirea neuronului este mai mare de 0.5. [5]

1.1.2 Clasificatorul binar SVM

Alternativ, se poate atașa o marjă de pierdere marginală la ieșirea neuronului și se poate antrena pentru a deveni un SVM binar.

Regularizarea interpretării. Regularizarea pierderii atât în cazul SVM, cât și Softmax pot fi interpretate din punct de vedere biologic ca o *omisiune graduală*, deoarece va avea ca efect egalarea ponderilor $\boldsymbol{w}$ cu 0 după fiecare actualizare a parametrilor.[5]

Un singur neuron poate fi folosit pentru a implementa un clasificator binar. (SVM binar sau Softmax binar).

1.2 Funcții de activare adesea folosite

Fiecare funcție de activare (sau neliniaritate) preia un număr și efectuează asupra acestuia o anumită operație matematică. În practică se pot întâlni mai multe funcții de activare:

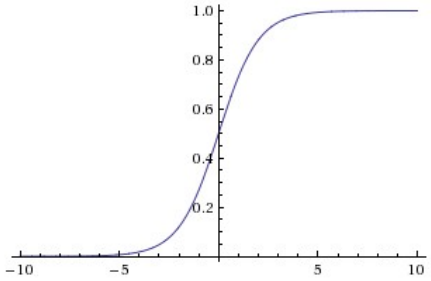


Figura 1.1.2.1 Funcția sigmoid înghesuie valorile în intervalul [0,1]

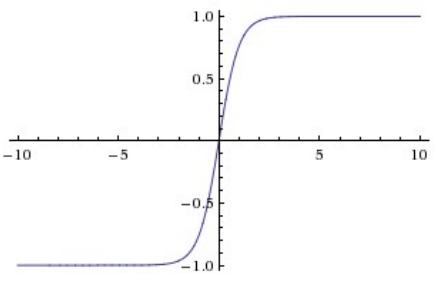


Figura 1.1.2.2 Funcția tangentă hiperbolică înghesuie valorile în intervalul [-1,1]

1.2.1 Funcția sigmoid

Funcția sigmoid are următoarea formă matematică $\boldsymbol{\sigma(x) = \frac{1}{1+e^{-x}}}$ și este reprezentată în Figura 1.1.2.1. După cum s-a menționat anterior, aceasta recepționează un număr real și îl scalează în intervalul [0,1]. În cazurile extreme, numerele negative foarte mari devin 0, iar numerele pozitive foarte mari devin 1. Funcția sigmoid a fost folosită frecvent de-a lungul timpului deoarece are o interpretarea buna a ratei de emisie a neuronului: de la o emisie nulă (0) la o emisie completă la o anumită frecvență maximă (1). În practică, funcția sigmoid a decăzut în popularitate și este foarte puțin folosită. Aceasta are două dezavantaje majore:

- ***Funcțiile sigmoid conduc la saturația și distrugerea gradientului***. O proprietate nedorită a neuronului sigmoid este când activarea neuronului se saturează în 0 sau 1, gradientul acestor regiuni fiind aproape 0. În urma procesului de ***„backpropagation”***, acest gradient local va fi multiplicat cu gradientul ieșirii porții pentru întregul obiectiv. Așadar, dacă gradientul local este foarte mic, acesta distruge gradientul și aproape niciun semnal nu va trece prin neuron către ponderile sale și în mod recursiv către datele sale. În plus, trebuie acordată o atenție deosebită la inițializarea ponderilor neuronilor sigmoizi pentru a preveni saturația. Spre exemplu, dacă ponderile inițiale sunt foarte mari, neuronii se vor satura, iar rețeaua neuronală abia va fi capabilă să învețe.
- ***Ieșirile funției sigmoid nu sunt centrate în zero.*** Acest lucru este nedorit deoarece neuronii, în straturile ulterioare procesării într-o rețea neuronală vor recepționa date ce nu sunt centrate în zero. Acest lucru are implicații asupra dinamicii în timpul scăderii gradientului, deoarece datele care intră în neuron sunt întotdeauna pozitive ($x > 0, f = w^T x + b$) , atunci gradientul ponderilor $\boldsymbol{w}$ în timpul procesului de ***„backpropagation”*** va deveni fie complet pozitiv, fie complet negativ (fiind dependent de gradientul întregii funcții $\boldsymbol{f}$). Acest lucru ar introduce dinamici în zig-zag nedorite în urma actualizării gradientului pentru ponderi.[5] Cu toate acestea, o dată ce acești gradienți sunt adunați de-a lungul unui lot de date, actualizarea finală a ponderilor poate avea semne diferite, astfel ameliorându-se problema. Prin urmare, deși acesta este inconvenient, are consecințe mai puțin severe comparativ cu saturarea funcției de activare de mai sus.

1.2.2 Tangenta hiperbolică.

Tangenta hiperbolică este ilustrată în Figura 1.1.2.2. Aceasta introduce valori reale în intervalul [-1,1]. Asemenea neuronului sigmoid, funcția de activare se saturează, dar spre deosebire de neuronul sigmoid, ieșirea sa este centrată în zero. Așadar, în practică, tangenta hiperbolică este întotdeaună preferată față de funcția sigmoid. De asemenea, neuronul tanh, este un neuron sigmoid scalat, în acest caz având forma $\tanh(x) = 2\sigma(2x) - 1$

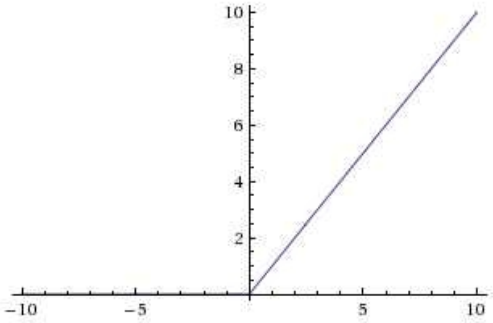


Figura 1.2.2.1 Funcția de activare ReLU, care este 0 când x < 0 și apoi liniară cu panta = 1 când x > 0

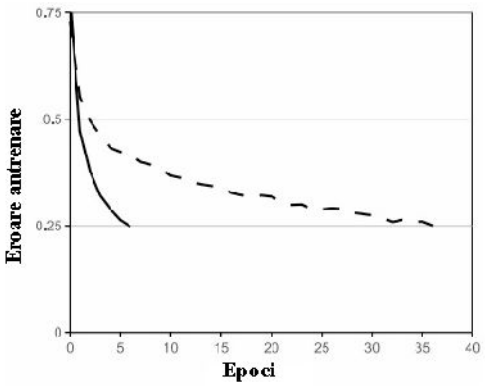


Figura 1.2.2.2 Cconvergență de 6 ori mai bună a funcției ReLU comparativ cu tangenta hiperbolică

1.2.3 Funcția ReLU

Popularitatea funcției ReLU a crescut exponențial în ultimii ani. Aceasta are forma $\boldsymbol{f = \max(0, x)}$. Cu alte cuvinte, funcția este prăguită în zero (Figura 1.2.2.1). Această funcție are anumite avantaje, dar și dezavantaje:

- (+) Are o convergență mai bună a gradientului stochastic comparativ cu funcția sigmoid/tangentă hiperbolică. Acest fapt se datorează formei sale liniare ce nu se saturează.
- (+) Spre deosebire de neuronii formați prin intermediul tangentei hiperbolică / funcției sigmoid, ce presupun operații complexe, funcția ReLU, poate fi implementată prin simpla prăguire a matricii de activare în 0.
- (-) Din nefericire, unitățile ReLU sunt fragile în timpul antrenamentului și pot muri. Spre exemplu, un gradient foarte mare ce trece printr-un neuron ReLU poate cauza actualizarea ponderilor în așa fel încât acel neuron nu se va activa în niciun punct vreodată. Dacă se întâmplă acest lucru, gradientul ce trece prin celulă va fi întotdeauna zero de la acel punct în colo. Astfel, celulele ReLU pot muri în mod ireversibil în timpul procesului de antrenare deoarece pot fi complești de varietatea datelor de intrare. Spre exemplu, se poate ajunge la un procent de 40% din rețea să fie moartă (neuronii să nu se poată activa de-a lungul întregului set de antrenare), dacă rata de învățare este mult prea mare. Prin setarea corectă a ratei de învățare acesta lucru nu va mai fi un impediment.

1.2.4 Funcția ReLU cu pierderi

Funcția ReLU cu pierderi reprezintă o încercare de a rezolva problema „morții" pentru funcția ReLU obișnuită. Astfel, în loc ca funcția să fie zero când $x < 0$, o funcție ReLU cu pierderi va avea o pantă negativă foarte mică (aproximativ 0.01). [5] Funcția va avea următoarea formă: $\boldsymbol{f(x) = 1(x < 0)(\alpha x) + 1(x \geq 0)(x)}$, unde $\boldsymbol{\alpha}$ este o constantă foarte mică. Panta în regiunea negativă poate constitui un parametru pentru fiecare neuron, așa cum este cazul neuronilor PreLU. Cu toate acestea, consecvența beneficiului în cadrul sarcinilor este în prezent neclară.

1.2.5 Funcția Maxout

Alte tipuri de celule care au fost propuse nu au formă funcțională $\boldsymbol{f(w^T x + b)}$ unde este aplicată o neliniaritate asupra produsului punct la punct dintre ponderi și date. Una dintre alegerile cele mai populare este neuronul Maxout care generalizează ReLU și varianta sa cu pierderi. Neuronul Maxout se folosește de funcția $\boldsymbol{\max(w_1^T + b_1, w_2^T + b_2)}$. [5] Se poate observa că ReLU și ReLU cu pierderi reprezintă un caz special al acestei forme (spre exemplu, ReLU are $w_1, b_1 = 0$). Neuronul Maxout, prin urmare, se bucură de toate beneficiile unei celule ReLU (regimul liniar al operației, fără saturație) și nu are dezavantajele acesteia (ReLU care mor). Cu toate acestea, spre

deosebire de neuronii ReLU, aceştia îşi dublează numărul de parametri pentru fiecare neuron în parte, conducând la un număr foarte mare de parametri.[5]

1.3 Arhitecturi de rețele neuronale. Organizarea la nivel de straturi

1.3.1 Rețelele neurale ca neuroni în grafuri

Rețelele neuronale sunt modelate asemenea unor colecții de neuroni, ce sunt conectați într-un graf aciclic. Cu alte cuvinte, ieșirile unor neuroni pot deveni intrările altor neuroni. Ciclurile nu sunt permise din moment ce acestea ar implica o buclă infinită în trecerea frontală a unei rețele. ***(engl. : forward pass).*** În locul unor conglomerate amorfe de neuroni conectați, modelele de rețele neurale sunt de asemenea organizate în diverse straturi de neuroni. Pentru rețelele neurale obișnuite, modelul de strat cel mai cunoscut este ***„stratul complet conectat”*** în cadrul căruia neuronii dintre două straturi adiacente sunt conectați complet în perechi, iar neuronii dintr-un singur strat nu împărtășesc nicio conexiune. În cadrul figurilor Figura 1.3.1.1 , Figura 1.3.1.2 se pot observa două exemple de topologii de rețele neuronale ce folosesc o stivă de straturi complet conectate.

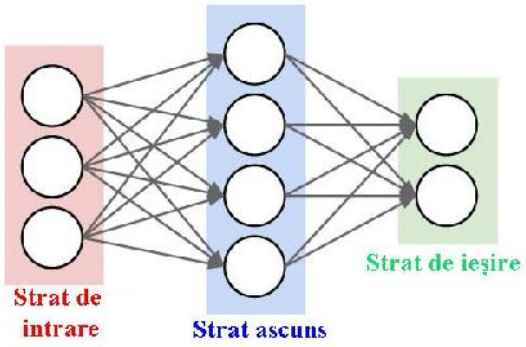


Figura 1.3.1.1 O rețea neuronală cu două straturi (un strat ascuns cu 4 neuroni și un strat de ieșire cu 2 neuroni), având trei intrări

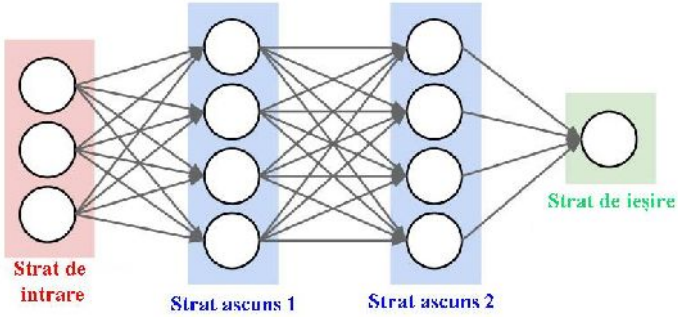


Figura 1.3.1.2 O rețea neuronală cu trei intrări, două straturi ascunse a câte 4 neuroni fiecare și un strat de ieșire. Se poate observa faptul că există conexiuni interstraturi , dar nu în cadrul aceluiași strat.

Convenții de numire. Atunci când se menționează o rețea neuronală cu n straturi, nu se ia în considerare stratul de intrare. Așadar, o rețea cu un singur strat descrie o rețea cu niciun strat ascuns (intrările sunt legate direct la ieșiri).

Stratul de ieșire. Spre deosebire de toate straturile dintr-o rețea neuronală, neuronii din stratul de ieșire nu dispun de o funcție de activare (sau putem considera că au o funcție de activare liniară). Acest fapt se datorează ultimului strat de ieșire care este ales să reprezinte scorul de clasă (ex: în cadrul clasificării), care este reprezentat prin numere reale sau printr-un obiectiv de valoare reală (ex: regresie).

Dimensionarea rețelelor neurale. Cele două metrici folosite adesea de către oameni pentru a măsura dimensiunea rețelelor neuronale sunt numărul de neuroni, sau mai comun, numărul de parametri. Folosind imaginile de mai sus vom avea:

- Rețeaua din Figura 1.3.1.1 are 4 + 2 = 6 neuroni (nu se iau în considerare intrările), [3x4]+[4x2] = 20 ponderi și 4+2 = 6 bias-uri, având un total de 26 de parametri de învățare.
- Rețeaua din Figura 1.3.1.2 are 4 + 4 + 1 = 9 neuroni, [3x4]+[4x4]+[4x1] = 12 + 16 + 4 = 32 ponderi și 4+4+1 = 9 bias-uri, având un total de 41 de parametri de învățare.

Rețelele convoluționale moderne conțin peste 100 de milioane de parametri de învățare și sunt constituite din 10-20 de straturi. Cu toate acestea, numărul de conexiuni efective este mult mai mare datorită partajării parametrilor.

1.3.2 Exemplu de calculare înainte și înapoi (engl: Example of feed-forward computation)

Înmulțiri repetate de matrici întrețesute cu funcție de activare. Unul din principalele motive pentru care rețelele neuronale sunt organizate în straturi este acela că această structură ajută la evalurea rețelelor neuronale utilizând operații vectoriale de matrici într-un mod simplu și eficient. Utilizând exemplul cu rețeaua neuronală cu trei starturi din Figura 1.3.1.2, intrarea va fi un vector [3x1]. Toate puterile de conectare pentru un strat pot fi stocate într-o singură matrice. Spre exemplu, ponderile primului strat ascuns ***W1*** vor avea dimensiunea [4x3], iar bias-ul pentru toate celulele va fi un vector ***b1*** de dimensiune [4x1]. În cazul de față, fiecare neuron va avea ponderile pe un singur rând de dimensiune ***W1,*** așadar vectorul multiplicării matricilor **np.dot(W1,x)** evaluează activările pentru toți neuronii din acel strat. Similar, **W2** va fi o matrice [4x4] ce stocheză conexiunile din al doilea strat ascuns, iar **W3** va fi o matrice [1x4] pentru stratul de ieșire. Adâncimea acestei rețele neuronale cu 3 straturi este reprezentată de înmulțirea a trei matrici, întrețesută cu aplicarea funcției de activare:

```
f = lambda x: 1.0/(1.0 + np.exp(-x)) # funcția de activare (se va
folosi funcția sigmoid)
x = np.random.randn(3, 1) # un vector de intrare aleator de 3
elemente (3x1)
h1 = f(np.dot(W1, x) + b1) # calcularea activărilor primului strat
ascuns (4x1)
h2 = f(np.dot(W2, h1) + b2) # calcularea activărilor celui de-al
doilea strat ascuns (4x1)
out = np.dot(W3, h2) + b3 # neuronul de ieșire (1x1)
```

În codul de mai sus ***W1,W2,W3,b1,b2,b3*** reprezintă parametrii de învățare ai rețelei. De asemenea, în loc să avem un singur vector coloană de intrare, variabila ***x*** poate ține întregul lot de antrenare (unde fiecare intrare va fi o coloană de dimensiune ***x***) și apoi toate exemplele vor fi evaluate în paralel în mod eficient. Se poate observa că stratul de ieșire nu are o funcție de activare (ex: acesta reprezintă scorul de clasă în vederea clasificării).[5]

Trecerea frontală a unui strat conectat complet corespunde unei multiplicări de matrice urmată de o deplasare de bias și de o funcție de activare.

1.3.3 Puterea reprezentantă

O altă modalitate de a privi rețelele neuronale cu straturi complet conectate este aceea că ele definesc o familie de funcții ce sunt parametrizate de ponderile rețelei.

S-a dovedit că rețelele neuronale cu cel puțin un singur strat de ieșire reprezintă aproximatori universali. Acest lucru a fost demonstrat: dându-se o funcție continuă ***f(x)*** și un $\epsilon > 0$, există o rețea neuronală **g(x)** cu un strat ascuns astfel încât $\forall x, |f(x) - g(x)| < \epsilon$. Cu alte cuvinte, rețeaua neuronală poate aproxima orice funcție continuă.

Dacă un singur strat ascuns este suficient pentru a aproxima orice funcție, de ce se folosesc mai multe straturi? Răspunsul este acela că deși o rețea neuronală cu două straturi este un aproximator universal, în practică acest lucru este complet inutil. În unidimensional, funcția „suma creșterilor

indicatorilor"$\boldsymbol{g(x) = \sum_i c_i 1(a_1 < x < b_i)}$ unde ***a,b,c*** sunt vectorii de parametri, reprezintă un aproximator universal. Rețelele neuronale lucrează bine în practică deoarece acestea exprimă bine funcții ce se potrivesc cu proprietățile statistice ale informațiilor adesea întâlnite și sunt ușor de învățat folosind algoritmi de optimizare. Similar, faptul că rețelele cu mai multe straturi ascunse lucrează mai bine decât rețelele cu un singur strat ascuns este o observație empirică, în ciuda faptului că puterea reprezentantă este aceeași.

În practică, rețelele cu 3 straturi tind să aibă o performanță sporită față de rețelele cu două straturi, dar pe măsură ce se mărește numărul de straturi diferența de performanță nu mai este atât de evidentă. Acest lucru se află într-o contradicție puternică cu rețelele convoluționale, unde adâncimea (numărul de straturi ascunse) s-a dovedit a fi o componentă importantă într-un sistem ce realizează o bună recunoaștere. Un argument pentru această observație este că imaginile conțin o structură ierarhică, așadar câteva straturi de procesare sunt utile în acest domeniu.

1.3.4 Alegerea numărului de straturi și dimensiunea acestora

Pe măsură ce dimensiunea și numărul de straturi dintr-o rețea neuronală cresc, **capacitatea** rețelei crește. Astfel , spațiul de reprezentare al funcțiilor crește din moment ce neuronii pot colabora pentru a exprima multe funcții diverse. Spre exemplu, presupunem că avem o problemă de clasificare binară în bidimensional. Se vor antrena trei rețele neuronale separate, fiecare cu un strat ascuns de o anumită dimensiunea și se vor obține următorii clasificatori:

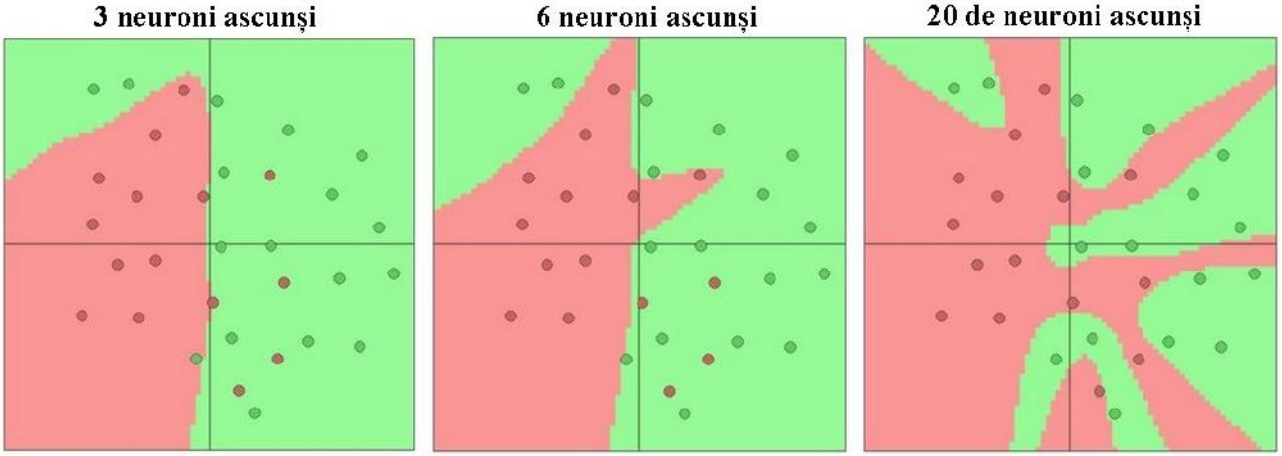


Figura 1.3.4.1 Rețelele Neuronale mai mari pot reprezenta funcții mult mai complexe. Informațiile sunt afișate ca cercuri colorate de clasa de care aparțin și regiunile de decizie de o rețea neuronală antrenată sub acestea.

În Figura 1.3.4.1 se poate observa că rețelele neuronale cu mai mulți neuroni pot exprima funcții mult mai complicate. Cu toate acestea, acest fapt poate constituie un avantaj (putem clasifica date mult mai complicate), dar și un dezavantaj (deoarece este mai ușor să suprascriem datele antrenate). Suprapunerea (***engl: overfitting***) are loc atunci când un model cu capacitate mare se potrivește cu zgomotul din date în loc de relația (presupusă) de bază. Spre exemplu, modelul cu 20 de neuroni potrivește bine datele de antrenare, dar are ca dezavantaj segmentarea spațiului în multe regiuni de decizie roșii și verzi disjuncte. Modelul cu 3 neuroni are puterea reprezentantă de a clasifica datele prin mișcări largi. Aceasta modelează datele ca două regiuni și interpretează câteva puncte roșii din interiorul grupului verde ca excedente (zgomot). În practică, acest lucru ar putea conduce la o generalizare mai bună pe setul de test.[5]

Este indicată utilizarea acestor metode pentru a preveni overfitting-ul în locul controlului numărului de neuroni, motivul fiind acela că rețelele mai mici sunt mai greu de antrenat cu metode locale, cum ar fi Gradient Descent. Este clar că funcțiile lor de pierdere au relativ puține minime locale, dar se pare că multe dintre aceste minime converg mai ușor și că sunt nesatisfăcătoare (adică cu pierderi mari). Pe de altă parte, rețelele neuronale mai mari conțin mult mai multe minime locale, dar aceste minime se dovedesc a fi mult mai bune în ceea ce privește pierderea lor reală. Deoarece rețelele neuronale sunt neconvexe, este greu de studiat matematic aceste proprietăți, dar au fost făcute câteva încercări de a înțelege aceste funcții obiectiv.

Așadar, ceea ce se obține ,dacă se antrenează o rețea de dimensiuni mici, pierderea finală poate avea o variație bună - în unele cazuri, având o convergență bună, iar în alte cazuri se poate pierde în minime locale.

Pe de altă parte, dacă se antrenează o rețea dimensiuni mari, se vor găsi multe soluții diferite, însă varianța în pierderea finală obținută va fi mult mai mică. Cu alte cuvinte, toate soluțiile sunt la fel de bune și se bazează mai puțin pe inițializarea aleatorie.

Așadar, forța de regularizare este modul ideal de a controla suprapunerea unei rețele neurale. Putem vedea rezultatele obținute de trei situații diferite:

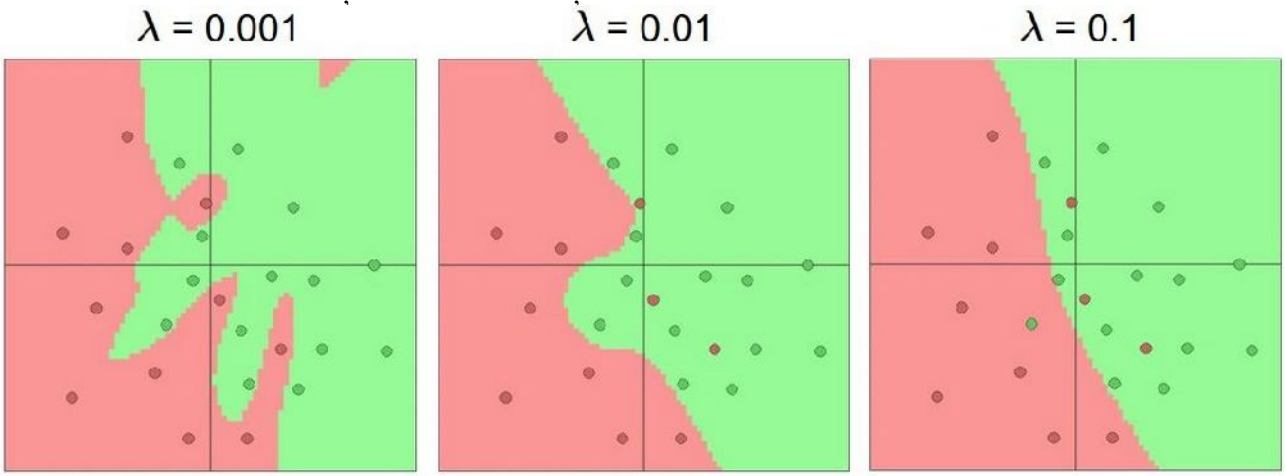


Figura 1.3.4.2 Efectele forței de regularizare: Fiecare rețea neuronală de mai sus are 20 de neuroni ascunși, dar schimbarea puterii de regularizare face ca regiunile sale decizionale finale să fie mai netede, cu o regularizare mai înaltă.

În concluzie, nu este indicată folosirea rețelelor mai mici din cauza procesului de overfitting. În schimb, este indicat să se utilizeze o rețea neurală pe cât de mare este posibil și tehnici de regularizare pentru a controla suprapunerea.

1.4 Backpropagation și Scăderea Gradientului Stochastic

Învățarea în profunzime reprezintă o clasă de modele de rețele neurale. Un astfel de model este constituit dintr-un strat de intrare, un strat de ieșire și un număr arbitrar de straturi ascunse. Aceste straturi la rândul lor sunt constituite din neuroni sau unități neurale. Aceștia din urmă poartă denumirea de neuroni deoarece prezintă un comportament similar cu neuronii din creierul uman.

În scopul învățării în profunzime, neuronii pot fi considerați ca o funcție neliniară de sumă ponderată a intrărilor.

1.4.1 Regresie logistică

Un neuron reprezintă o funcție ce conectează un vector de intrare $\{x_1,...,x_K\}$ la un scalar y prin intermediul unui vector de ponderi $\{w_1,...,w_K\}$ și o funcție neliniară f.

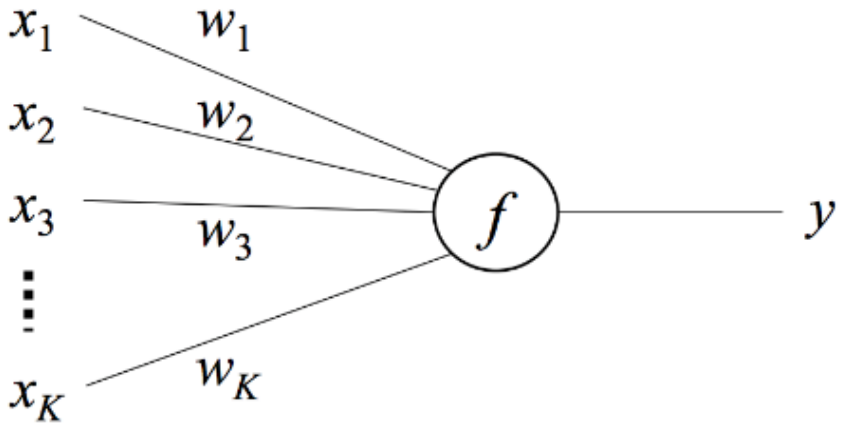


Figura 1.4.1.1 Funcția f utilizează suma ponderată a intrărilor și returnează y.

$$y = f\left(\sum_{i=0}^{K} w_i x_i\right) = f(w^t x)$$

În majoritatea cazurilor un element adițional este adăugat vectorului de intrări, ce are valoarea egală cu **1** cu o pondere corespunzătoare elementului adițional ce joacă rol de **bias** [2]. Funcția ***f*** poartă denumirea de funcție de legătură ce asigură neliniaritatea dintre intrare și ieșire. **Funcția logistică** este adesea aleasă ca funcție de legătură și este definită astfel:

$$f(u) = \frac{1}{1 + e^u}$$

După o serie de substituții corespunzătoare, formula pentru modelare unui singur neuron devine:

$$y = \frac{1}{1 + e^{(w^T x)}}$$

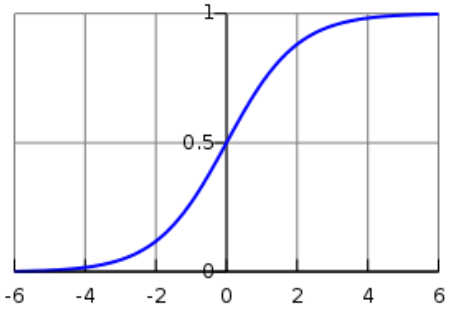


Figura 1.4.1.2 Graficul funcției sigmoid

Din graficul acesteia, se poate observa că este o funcție netedă, diferențiabilă, mărginită între 0 și 1, iar derivata sa va avea următoarea formă:

$$\frac{df(u)}{du} = f(u)\big(1 - f(u)\big) = f(u)f(-u)$$

Derivata va fi utilizată când se va învăța vectorul de ponderi ***w*** prin intermediul scăderii gradientului stochastic (***engl: stochastic gradient descend***). [2]

Asemenea oricărei probleme de optimizare, țelul este acela de a minimiza o funcție obiectiv. În mod tradițional, funcția obiectiv măsoară diferența dintre ieșirea actuală ***t*** și rezultatul estimat ***f(w^{T}x)***. În acest caz, se va folosi funcția de pierdere pătratică:

$$E = \frac{1}{2}(t - y)^2 = \frac{1}{2}\big(t - f(w^T x)\big)^2$$

Se dorește obținerea ponderilor ***w*** astfel încât funcția obiectiv amintită anterior să fie minimă, fapt realizat prin intermediul scăderii gradientului stochastic (***engl.: stochastic gradient descend***). În cadrul metodei de scădere a gradientului stochastic, se vor actualiza în mod iterativ parametrii pondere în direcția gradientului funcției de pierdere până se atinge minimul. Spre deosebire de scăderea gradientului obișnuit, nu se va folosi întregul lot pentru a calcula gradientul la fiecare iterație. În schimb, în fiecare iterație se va selecta în mod aleator un singur punct din lot și se va face o deplasare în direcția gradientului relativ la acel punct. Desigur, aceasta este o aproximare a gradientului real, dar se poate demonstra că în cele din urmă va atinge minimul. Utilizarea scăderii gradientului stochastic prezintă o serie de avantaje față de gradientul obișnuit:

1. Scăderea gradientului necesită încărcarea întregului lot în memorie. În cazul unui lot de

dimensiuni mari, acesta poate cauza probleme. Scăderea gradientului stochastic necesită doar un singur punct la fiecare moment de timp, fapt ce conduce la un consum de memorie mult mai scăzut.

2. Majoritatea loturilor de date prezintă redundanță. Scăderea gradientului tradițional necesită o trecere completă prin date până când se va realiză actualizarea. Datorită redundanței, o actualizare majoră se realiza de cele mai multe ori fără a itera prin întreg lotul după cum este cazul scăderii gradientului stochastic.
3. Ca o consecință a punctului anterior, scăderea gradientului stochastic poate converge adesea mai rapid decât scăderea gradientului tradițional. De asemenea, garantează găsirea unui minim global dacă funcția de pierdere este convexă.[2]

Având funcția obiectiv ***E*** deja definită în funcție de un singur punct de date, se poate calcula gradientul relativ la un element arbitrar al vectorului de ponderi, w_i.

$$\frac{\partial \mathrm{E}}{\partial w_i} = \frac{\partial \mathrm{E}}{\partial y} * \frac{\partial \mathrm{y}}{\partial \mathrm{u}} * \frac{\partial \mathrm{u}}{\partial w_i} = (y - t) * y(1 - y) * x_i$$

În continuare, se poate obține relația de actualizare a scăderii gradientului stochastic (se vor folosi notații vectoriale):

$$w^{nou} = w^{vechi} - \eta * (y - t) * y(1 - y) * x$$

$\eta > 0$ reprezintă dimensiunea pasului. Punctele ***(x,y)*** sunt introduse în funcția de actualizare până când ponderile ***w*** converg la valoarea lor optimă.

Acest procedeu poartă denumirea de ***regresie logistică*** , iar dacă în locul funcției logistice ar fi fost folosită funcția treaptă unitate, s-ar fi realizat un perceptron.

1.4.2 Backpropagation

O rețea neurală este constituită dintr-un strat de intrare, un strat de ieșire și un strat ascuns. Stratul de intrare este constituit din vectorul de intrare $x=\{x_1,...,x_K\}$. Stratul ascuns este reprezentat de un vector de N neuroni $h=\{h_1,...,h_N\}$. Iar în cele din urmă, stratul de ieșire ce conține un neuron pentru fiecare element al vectorului de ieșire $y=\{y_1,...,y_M\}$. Fiecare element din stratul de intrare este conectat la fiecare neuron din stratul ascuns având w_{ki} , care indică ponderea asociată conexiunii dintre al ***k****-lea* element al intrării și al ***i****-lea* neuron ascuns. Aceeași structură a conexiunii este prezentă între stratul ascuns și cel de ieșire cu w'_{ij} indicând ponderea asociată conexiunii dintre al ***i****-lea* neuron ascuns și al ***j****-lea* neuron de ieșire.[2] Structura neuronului este ilustrată în Figura 1.4.2.1.

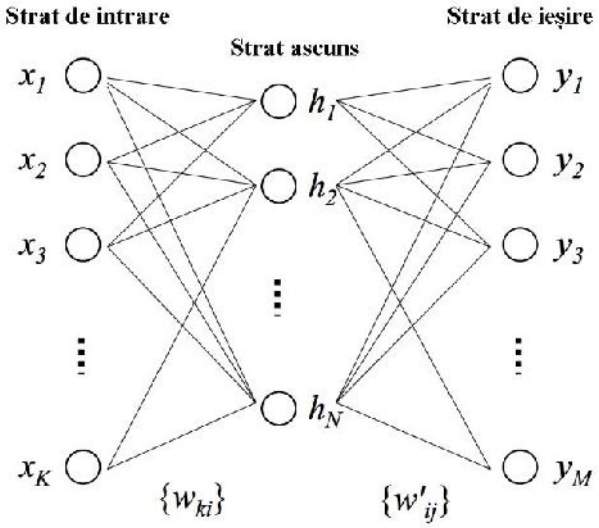


Figura 1.4.2.1 Structura neuronului

O abordare utilă se dovedește a fi vizionarea ponderilor w_{ki} ca cea de-a ***(k,i)*** intrare în matrice de ponderi ***W*** de dimensiune **K x N** și în mod similar ponderea w'_{ij} ca cea de-a ***(i,j)*** intrare într-o matrice de ponderi ***W'*** de dimensiune ***N x M.*** Ieșirea fiecărui neuron din stratul ascuns și cel de ieșire este calculată în aceeași manieră. Se aplică funcția logistică asupra sumei ponderate a intrărilor neuronului. Spre exemplu, ieșirea unui neuron arbitrar din stratul ascuns, h_i este:

$$h_i = f(u_i) = f(\sum_{k=1}^{K} w_{ki}x_k)$$

Și în mod similar ieșirea unui neuron arbitrar de ieșire, y_j este:

$$y_j = f(u'_j) = f(\sum_{i=1}^{N} w'_{ij}h_i)$$

Funcția obiectiv este aceeași ca mai devreme, exceptând faptul că este sumată peste toate elementele din stratul de ieșire:

$$E = \frac{1}{2}\sum_{j=1}^{M}(y_j - t_j)^2$$

Următorul pas îl reprezintă construirea ecuațiilor de actualizare pentru ambele seturi de ponderi: ponderile stratului de intrare – strat ascuns w_{ki} și ponderile stratului ascuns – strat de ieșire w'_{ij}. Pentru a realiza acest lucru trebuie să se calculeze gradientul funcției obiectiv E relativ la w_{ki} și gradientul relativ la w'_{ij}. Se va începe prin calculul gradientului relativ la w'_{ij} (ponderile strat ascuns – strat de ieșire).[2]

$$\frac{\partial \mathrm{E}}{\partial w'_{ij}} = \frac{\partial \mathrm{E}}{\partial y_j} * \frac{\partial y_j}{\partial u'_j} * \frac{\partial u'_j}{\partial w'_{ij}}$$

Derivata lui E în raport cu y_j este:

$$\frac{\partial \mathrm{E}}{\partial y_j} = y_j - t_j$$

Funcția logistică f în raport cu intrarea sa u are forma $f(u)(1-f(u))$, iar prin introducerea acesteia în relația anterioară se va obține:

$$\frac{\partial \mathrm{E}}{\partial w'_{ij}} = (y_j - t_j) * y_j(1 - y_j)h_i$$

Cu ajutorul acestui gradient se poate obține funcția de actualizare pentru w'_{ij}

$$w'^{nou}_{ij} = w'^{vechi}_{ij} - \eta * (y_j - t_j) * y_j(1 - y_j)h_i$$

Pentru a obține gradientul funcției obiectiv relativ la ponderile stratului de intrare – strat de ieșire w_{ki} se va utiliza gradientul complet:

$$\frac{\partial E}{\partial w_{ki}} = \sum_{j=1}^{M}\left(\frac{\partial E}{\partial y_j} * \frac{\partial y_j}{\partial u'_j} * \frac{\partial u'_j}{\partial h_i}\right) * \frac{\partial h_i}{\partial u_i} * \frac{\partial u_i}{\partial w_{ki}}$$

Suma se datorează faptului că celula ascunsă la care w_{ki} se conectează este conectată la fiecare celulă de ieșire, astfel fiecare din acești gradienți trebuie luat în considerare. Având deja calculate atât $\frac{\partial E}{\partial y_j}$ cât și $\frac{\partial y_j}{\partial u'_j}$ se va obține:

$$\frac{\partial E}{\partial y_j} * \frac{\partial y_j}{\partial u'_j} = (y_j - t_j) * y_j(1 - y_j)$$

Următorul pas implică calcularea derivatelor : $\frac{\partial u'_j}{\partial h_i}, \frac{\partial h_i}{\partial u_i}, \frac{\partial u_i}{\partial w_{ki}}$

$$\frac{\partial u'_j}{\partial h_i} = \frac{\partial \sum_{i=1}^{N} w'_{ij} h_i}{\partial h_i} = w'_{ij}$$

Apoi folosind derivata funcției logistice:

$$\frac{\partial h_i}{\partial u_i} = h_i(1 - h_i)$$

Iar în cele din urmă se ajunge la:

$$\frac{\partial u_j}{\partial w_{ki}} = \frac{\partial \sum_{i=1}^{N} w_{ki} x_k}{\partial w_{ki}} = x_k$$

După efectuarea substituțiilor corespunzătoare se va ajunge la următorul gradient:

$$\frac{\partial E}{\partial w_{ki}} = \sum_{j=1}^{M} \left[(y_j - t_j) * y_j(1 - y_j) * w'_{ij}\right] * h_i(1 - h_i) * x_k$$

Acest proces este cunoscut sub denumirea de **backpropagation** deoarece se începe cu o eroare finală de ieșire $y_j - t_j$ pentru neuronul de ieșire j și această eroare se propagă înapoi în cadrul rețelei pentru a actualiza ponderile.

2. Rețele neurale recurente și LSTM

Rețelele neurale recurente reprezintă un tip de rețele neurale artificiale pentru recunoașterea modelelor în secvențe de date, cum ar fi text, genomi, scris de mână, vorbire sau serii numerice temporale obținute de la senzori, cursuri valutare. Acești algoritmi iau în considerare momentele de timp și secvențele,astfel aceștia având dimensiune temporală.

Cercetătorii au demonstrat că acestea constituie cel mai puternic și util model de rețele neurale alături de mecanismul de atenție și rețelele cu memorie. RNR-urile (***engl: RNN***) se aplică chiar și pe imagini care pot fi împărțite pe zone și tratate ca o secvență.

Din moment ce rețelele posedă un anumit tip de memorie, iar memoria este o parte a condiției umane, vor exista nenumărate analogii cu memoria din creierul uman.

2.1 Rețele Feedforward

Pentru a înțelege rețelele recurente, în primul rând trebuie înțelese principiile de bază ale rețelelor ***feedforward***. Denumirea ambelor tipuri de rețele își are originea în modul în care acestea propagă informația printr-o serie de operații matematice efectuate în nodurile rețelei. Una dintre acestea propagă informația direct (neatingând un anumit nod de două ori), în timp ce cealaltă propagă informația în cadrul unei bucle (cea din urmă purtând numele de rețea neuronală recurentă).

În cazul rețelelor feedforward, valorile de intrare sunt introduse în rețea și trasfomate în ieșiri; prin intermediul învățării supervizate, ieșirile vor deveni etichete (un nume aplicat intrărilor). Așadar, acestea mapează informația brută în categorii, recunoscând modele care pot semnala, spre exemplu, că o imagine folosită ca intrare ar trebui clasificată drept „pisică” sau „elefant”.

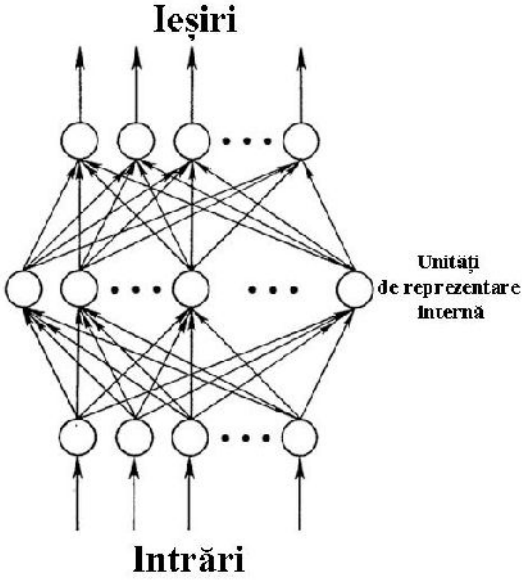


Figura 2.1.1 Rețea feedforward

O rețea feedforward este antrenată pe imagini etichetate până când minimizează eroarea de clasificare a imaginilor. Având un set de parametri (sau ponderi, cunoscuți drept model), rețeaua încearcă să clasifice date pe care nu le-a mai întâlnit. O rețea feedforward antrenată poate fi expusă oricărei colecții de imagini, iar prima fotografie la care este expusă nu va afecta neapărat modul în care o va clasifica pe cea de-a doua. Dacă se observă imaginea unei pisici, acesta nu va conduce la faptul că următoarea imagine va fi percepută ca un elefant.

Așadar, rețeaua feedforward nu are nicio noțiune a ordinii temporale, iar singura intrare pe care o ia în considerare este exemplul curent la care a fost expusă. Rețelele feedforward suferă de amnezie în ceea ce privește trecutul recent: acestea își amintesc doar momentele formative ale procesului de antrenare.

Fenomenul de „backpropagation” în timp

Scopul rețelelor neuronale este acela de a clasifica intrarea secvențială. Astfel, un rol esențial îl are propagarea înapoi (***engl: backpropagation***) a erorii și scăderea gradientului.

Fenomenul de backpropagation în rețelele feedforward se mișcă înapoi de la eroarea finală către ieșirile, ponderile și intrările fiecărui start ascuns, alocând acelor ponderi responsabilitatea unei porțiuni de eroare prin calculul derivatei parțiale - $\partial E/\partial w$, sau relația între ratele lor de schimb. Acele derivate sunt apoi utilizate folosind regula de învățare (scăderea gradientului) pentru a ajusta ponderile în direcția scăderii erorilor. [1]

Rețelele recurente se bazează pe o extindere a procesului de ***backpropagation*** , numit ***backpropagation în timp.*** Timpul, în acest caz, este exprimat ca o serie de calcule ordonate și bine definite ce conectează un moment de timp de următorul, constituind astfel tot ceea ce este nevoie pentru ca procedeul de ***backpropagation*** să funcționeze.

Rețelele neurale, fie ele recurente sau nu, reprezintă funcții compuse de forma ***f(g(h(x)))***. Prin adăugarea unui element se va ajunge la extinderea seriei de funcții pentru care se va calcula derivata.

Fenomenul de backpropagation trunchiat în timp

Fenomenul de ***backpropagation*** trunchiat în timp reprezintă o aproximare a fenomenului de ***backpropagation*** în timp, care are un avantaj în utilizarea secvențelor foarte lungi, deoarece în cazul primului menționat costul pentru actualizarea parametrilor devine mult prea mare de-a lungul mai multor pași.

Dezavantajul este constituit de faptul că gradientul poate crește datorită trunchierii, astfel rețeaua neputând memora dependențe la fel de lungi ca în cazul fenomenului de ***backpropagation în timp***. [1]

2.2 Rețele neuronale recurente

Rețelele recurente, pe de altă parte, primesc ca intrare nu doar intrarea curentă, dar și ceea ce au perceput anterior (de-a lungul timpului).

În Figura 2.2.1 este reprezentată o rețea simplă recurentă, unde *BTSXPE* din partea de jos a acesteia reprezintă intrarea la momentul curent, în timp ce unitatea de context (***engl: context unit***) reprezintă ieșirea la moemntul anterior.

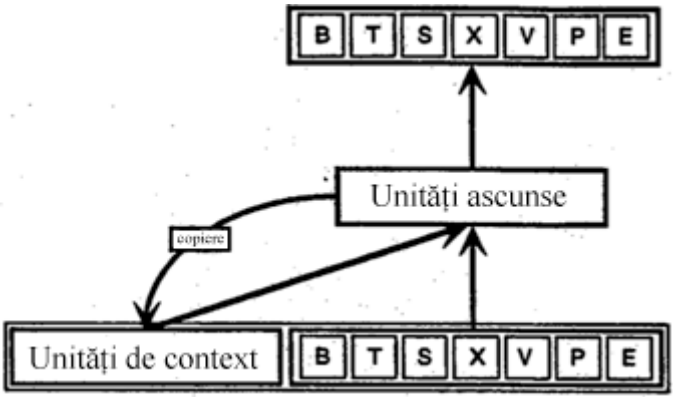


Figura 2.2.1 Rețea simplă recurentă

Decizia luată de o rețea recurentă la momentul ***t-1*** va afecta decizia luată la momentul ***t***. Așadar, rețelele recurente au două surse de intrare, prezentul și trecutul recent, care se combină pentru a răspunde noilor informații.

Rețelele recurente se disting de rețelele feedforward prin faptul că bucla de feedback este conectată la deciziile anterioare, folosind ieșirea pe post de intrare la fiecare moment.

Se menționează adesea că rețelele recurente au memorie. Adăugarea de memorie rețelelor neurale are un scop: rețelele recurente se folosesc de informația din secvențe pentru a efectua sarcini pe care rețelele feedforward nu le pot îndeplini.

Acea informație secvențială este menținută în stratul ascuns al rețelei recurente, reușind să se întindă de-a lungul mai multor etape în timp, trecând în cascadă și afectând procesarea fiecărui exemplu introdus. Descoperirea corelațiilor dintre evenimente separate de mai multe momente de timp este esențială, iar aceste corelații se numesc „dependențe de lungă durată” , deoarece un eveniment la un anumit moment de timp este reprezentat ca o funcție de unul sau mai multe evenimente ce au avut loc înaintea acestuia. RNR-urile pot fi privite astfel: existența unei modalități de a împărtăși ponderile de-a lungul timpului.[1]

Asfel, în cazul oamenilor, procesul gândirii nu începe de la 0 în fiecare secundă, așadar înțelerea fiecărui cuvânt se bazează pe înțelegerea cuvintelor anterioare. Prin urmare, nu se renunță la ceea ce s-a înțeles, începându-se procesul de gândire de fiecare dată, gândurile având persistență.

Acesta este unul din dezavantajele rețelelor neuronale tradiționale, ele fiind incapabile de acest lucru. Spre exemplu, se dorește clasificarea tipurilor de evenimente care se petrec în fiecare moment dintr-un film. Nu este clar cum o rețea neuronală tradițională si-ar putea folosi mecanismul său de înțelegere pentru momentele anterioare din film astfel încât să le poată prevedea pe cele viitoare.

Rețele neuronale recurente rezolvă acest inconvenient, ele reprezentând rețele cu bucle în cadrul acestora, ce permit informației să persiste.

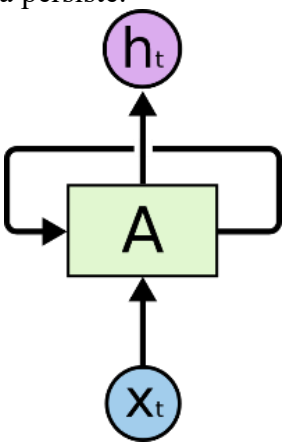


Figura 2.2.2 Rețelele neuronale recurente prezintă bucle

Figura 2.2.2 reprezintă o parte dintr-o rețea neuronală, A, ce primește ca intrare $\boldsymbol{X_t}$ și are drept ieșire $\boldsymbol{h_t}$. O buclă permite informației să fie transmisă de la un punct al rețelei neuronale la altul.

Cu toate acestea, dacă aruncăm o privire mai în ansamblu , se poate observa că acestea nu sunt așa diferite de o rețea neurală normală. O rețea neurală poate fi considerată ca un set de mai multe copii alea aceleiași rețele, fiecare transmițând un mesaj succesorului său. În urma desfășurării buclei se va obține:

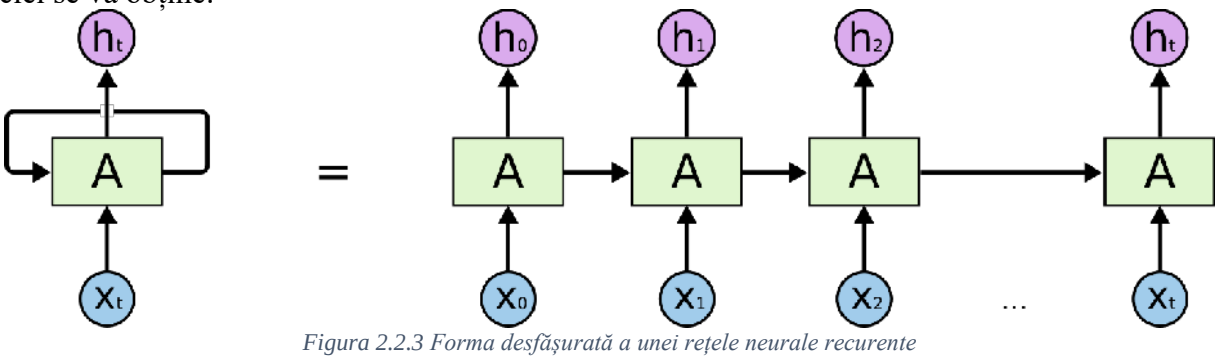


Figura 2.2.3 Forma desfășurată a unei rețele neurale recurente

Datorită acestei forme înlănțuite, rețelele neuronale sunt strâns legate de secvențe și liste, acestea reprezentând arhitectura naturală a rețelelor neuronale ce se utilizează pentru asemenea date.

În ultimii ani, aplicarea RNR-urilor asupra unor multitudini de probleme cum ar fi: recunoașterea vorbirii, modelarea limbii, traduceri, captarea de imagini, ect.. s-a dovedit a fi un adevărat succes.

Ceea ce este cu adevărat important, în urma acestor succese, este utilizarea LSTM-urilor (***engl: Long Short Term Memory***, ce constituie un model special de rețele neuronale), pentru un număr impresionant de sarcini unde acestea se descurcă mult mai bine decât rețelele neuronale clasice. Aproape toate rezultatele interesante în urma utilizării rețelelor neurale recurente sunt obținute folosind LSTM-uri.[4]

2.3 Problema dependențelor de lungă durată

Un lucru fascinant în ceea ce privește RNR-urile este constituit de faptul că acestea pot conecta informație anterioară la sarcina curentă, cum ar fi utilizarea cadrelor anterioare dintr-un film pentru a înțelege cadrul curent. Dacă RNR-urile ar putea face acest lucru, ar fi extrem de utile.

Uneori avem nevoie să ne uităm la informația anterioară pentru a efectua sarcina curentă. Spre exemplu, se consideră un model de limbă care încearcă să prezică următorul cuvânt bazându-se pe cele anterioare. Dacă se încearcă prezicerea ultimul cuvânt în fraza „Norii sunt pe *cer*”, nu avem nevoie de nicio informație ulterioară – se poate deduce foarte simplu că următorul cuvânt este *cer*. În unele cazuri, unde spațiul dintre informația relevantă și locul în care este nevoie de această este mic, RNR-urile pot învăța din informația anterioară.

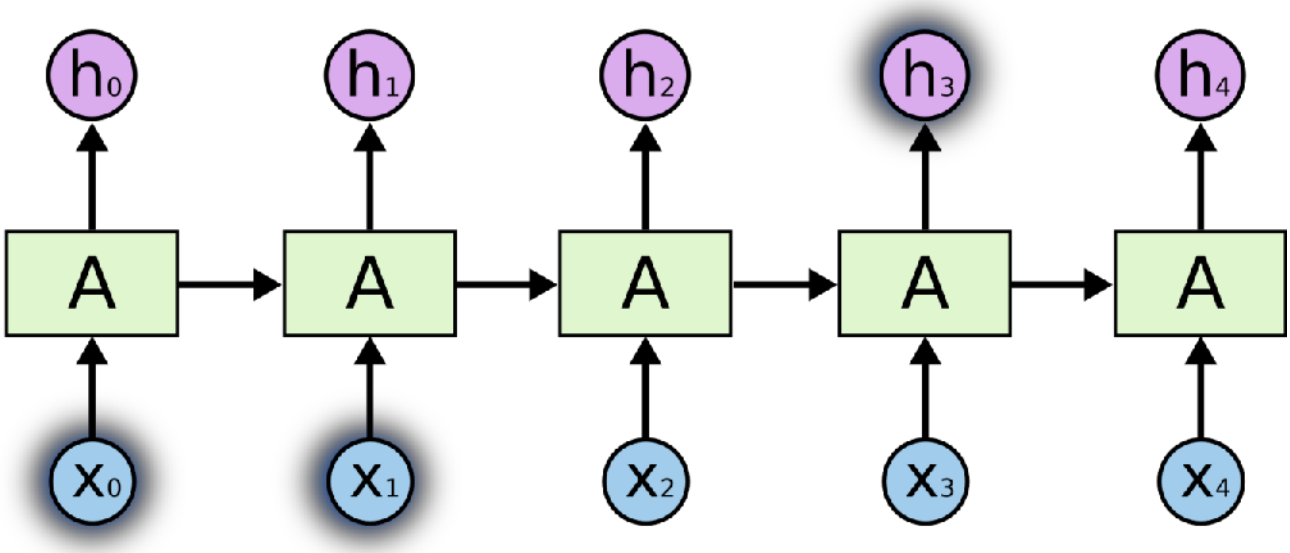


Figura 2.3.1 Dependențe de scurtă durată

Dar există și cazuri în care avem nevoie de informație suplimentară. Spre exemplu, dacă se dorește prezicerea ultimului cuvânt din fraza „Am crescut în Franța... Vorbesc fluent limba *franceză*”. Informația anterioară sugerează că următorul cuvânt este probabil numele unei limbi, dar dacă se dorește limitarea acestei căutări la o singură limbă, este nevoie de contextul „Franța”, ce se află mult mai în urmă. Astfel, se poate ca spațiul dintre informația relevantă și locul unde este necesară să devină foarte mare.

Din nefericire, pe măsură ce acel spațiu crește, RNR-urile devin incapabile să conecteze informația.

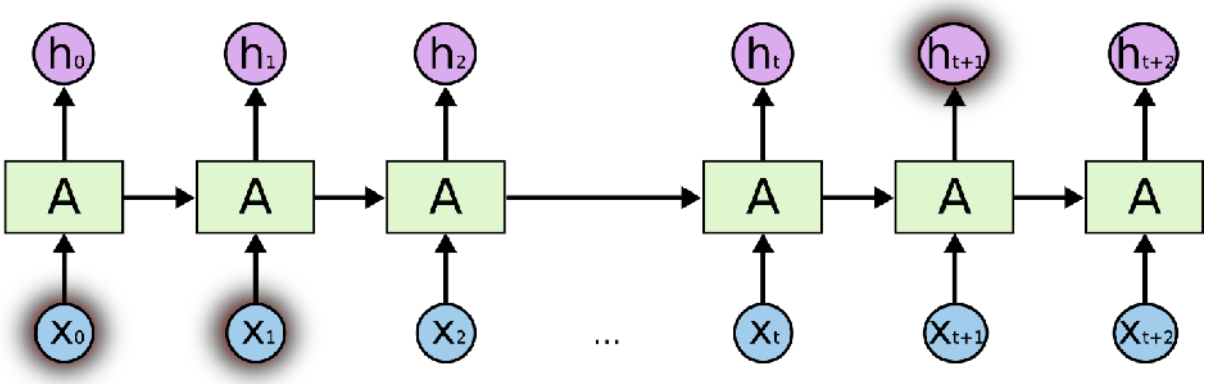


Figura 2.3.2 Dependențe de lungă durată

În teorie, RNR-urile sunt capabile să controleze asemenea dependențe de lungă durată. Un om poate prelua cu ușurință parametrii pentru ele cu scopul de a rezolva o problemă de acest fel. Din păcate, în practică, RNR-urile nu sunt capabile să îi învețe. Această problemă a fost studiată pe larg de Hochreiter (1991) și Bengio (1994), care a descoperit câteva motive fundamentale pentru care acest lucru ar putea fi dificil. Din fericire, LSTM-urile nu au această problemă.

2.4 Rețele LSTM

Rețelele cu memorie de scurtă durată persistentă (**engl.: Long Short Term Memory**) - sunt o specie de RNR-uri , capabile să învețe dependențele de lungă durată. Acestea au fost introduse de Hochreiter și Schmidhuber în anul 1997 și au fost rafinate și popularizate de mulți alții. Ele funcționează extrem de bine pe o plajă foarte largă de probleme și sunt utilizate pe scară largă.

LSTM-urile permit conservarea erorii ce poate fi propagată înapoi în timp și în straturi. Prin menținerea unei erori constante, acestea permit rețeleler neurale recurente să învețe de-a lungul mai multor momente de timp (peste 1000 de astfel de momente), astfel deschizând de la distanță un canal între cauze și efecte. Acest fapt reprezintă una din dificultățile principale ale rețelelor neurale.

LSTM-urile mențin informația din afara fluxului normal al rețelei recurente într-o celulă închisă. Informația poate fi stocată, scrisă sau citită dintr-o celulă, asemenea datelor din memoria calculatorului. Celula ia deciziile cu privire la informațiile ce trebuie stocate și când să permită citirea, scrierea și ștergerea acestora, prin intermediul unor porți care se deschid și se închid. Spre deosebire de stocarea digitală pe computere,aceste porți sunt analogice, implementate prin înmulțirea element cu element de către sigmoide, care se află în intervalul 0-1. Analogul are avantajul asupra digitalului de a fi diferențiat și, prin urmare, este potrivit pentru backpropagation.

Aceste porți acționează asupra semnalelor pe care le primesc și, asemănător cu nodurile rețelei neuronale, blochează sau transmit informații bazate pe forța și importul lor, pe care le filtrează cu propriile seturi de ponderi. Acele ponderi, precum ponderile care modulează stările de intrare și stările ascunse, sunt ajustate prin procesul de învățare a rețelelor recurente. Astfel celulele învață când să permită introducerea, ieșirea sau ștergerea datelor prin intermediul procesului iterativ de a face presupuneri, de a împiedica erorile și de a regla ponderilor prin scăderea gradientului.

LSTM-urile sunt create special pentru a evita problema dependenței de lungă durată. Reamintirea informației pe perioade foarte lungi de timp reprezintă comportamentul lor standard, nefiind ceva pe care se chinuie sa o învețe.

Toate rețelele neuronale sunt formate dintr-un lanț de module de rețele neuronale care se repetă. În cadrul RNR-urilor standard, acest modul repetitiv va avea o structură foarte simplă, cum ar fi un singur strat de tangentă hiperbolică.

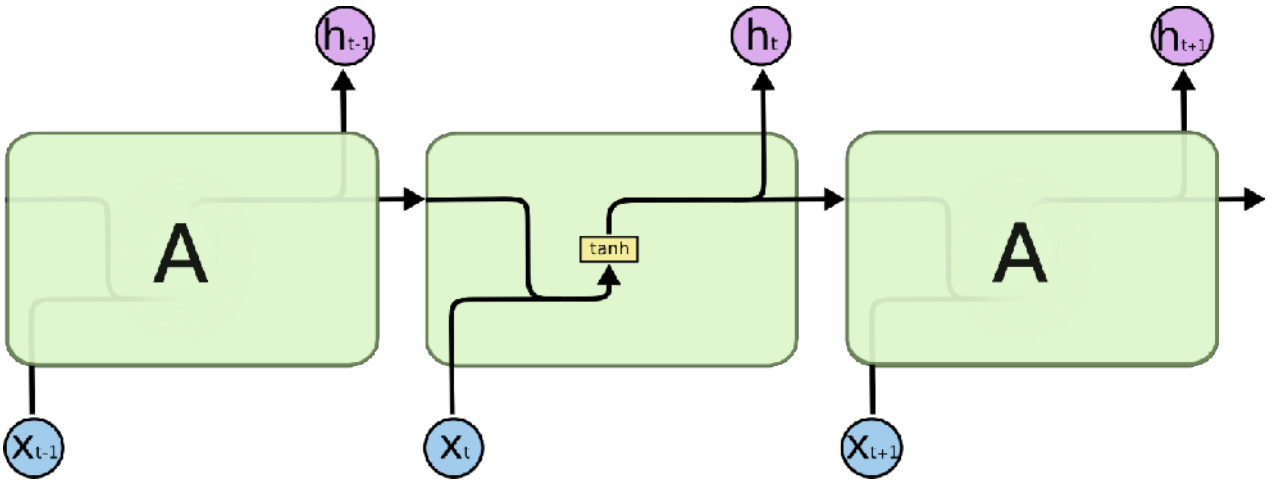


Figura 2.4.1 Modulul repetitiv intr-o RNR standard conține un singur strat

LSTM-urile au de asemenea o structură înlănțuită, dar modulul repetitiv are o structură diferită. În loc să aibă o rețea neuronală cu un singur strat, rețeaua are patru straturi, ce interacționează într-un mod special.

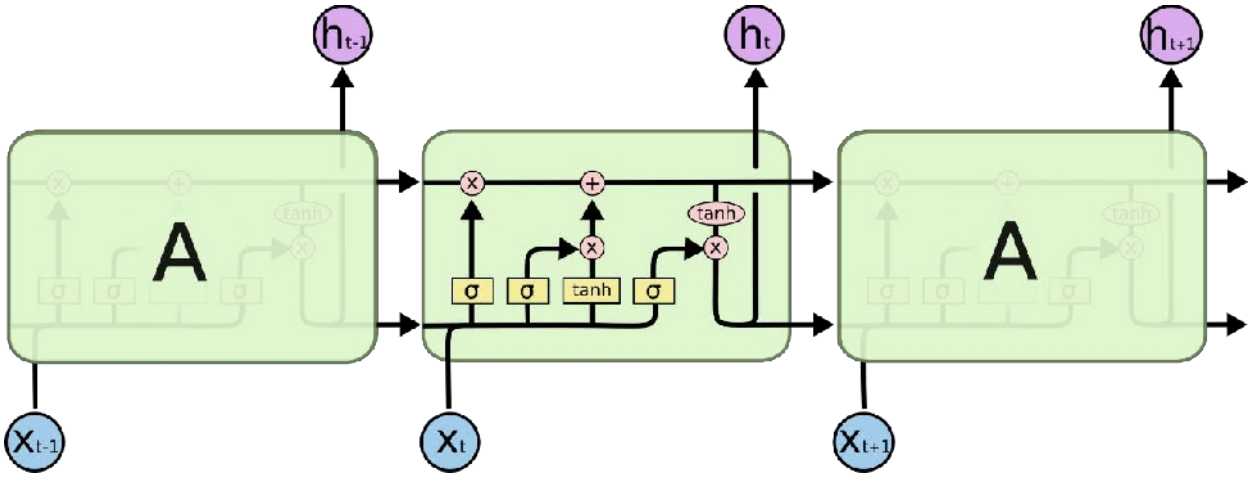


Figura 2.4.2 Modulul repetitiv în cadrul unui LSTM conține patru straturi care interacționează între ele

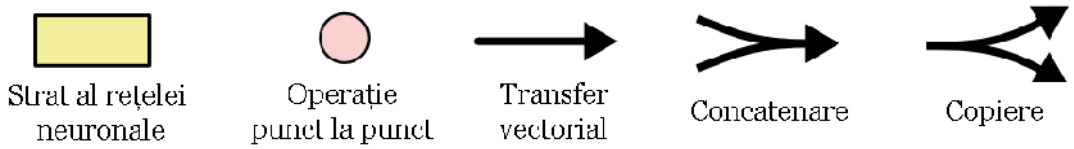


Figura 2.4.3 Notații utilizate

În Figura 2.4.3, fiecare linie poartă un vector de la ieșirea unui nod la itrarea altora. Cercurile roz reprezintă operații punct la punct, cum ar fi adunarea de vectori, în timp ce dreptunghiurile galbene reprezintă straturile învățate din rețeaua neuronală. Liniile care se îmbină reprezintă concatenare, în timp ce liniile care se bifurcă reprezintă faptul că conținutul este copiat și copiile sunt transmise în diferite locații.

2.5 Ideea centrală din spatele LSTM-urilor

Principiul de funcționare al LSTM-urilor are la bază stările celulelor (de exemplu, C_{t-1} și C_t), legate prin linia orizontală ce traversează Figura 2.5.1 în partea de sus.

Această linie orizontală poate fi asociată modului de funcționare a unei benzi rulante. Informația va trece prin lanțul de prelucrare, suferind numai câteva modificări minore datorate existenței unor funcții liniare.

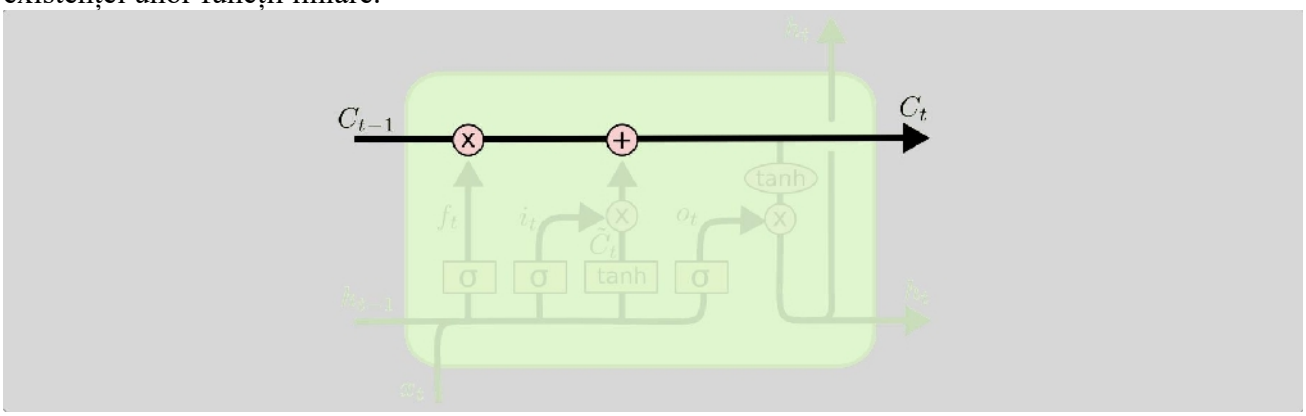


Figura 2.5.1 Celula de stare

LSTM-ul are capacitatea de a adăuga sau elimina informație din celula de stare, fapt realizat prin intermediul unor structuri numite porți. Porțile reprezintă o cale prin care informația este lăsată să treacă. Acestea sunt compuse dintr-un strat sigmoid de rețea neuronală și a operație pozitivă de înmulțire punct la punct.[4]

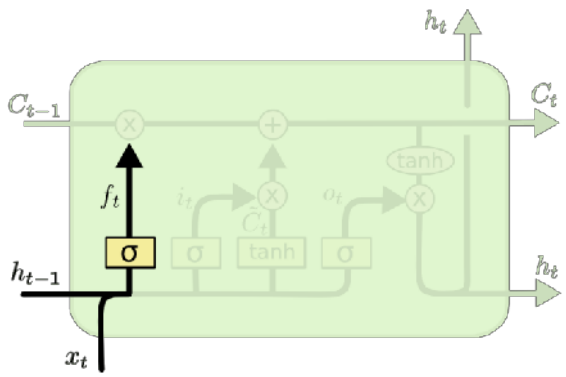
Stratul sigmoid are la ieșire numere între 0 și 1, astfel descriind cantitatea din fiecare componentă care este lăsată să treacă mai departe. Pentru 0 nu va trece nimic, în timp ce pentru 1 va trece tot.

O LSTM are trei astfel porți pentru a proteja și controla celula de stare.

2.6 Principiul de funcționare al LSTM-urilor

Primul pas în cadrul LSTM-ului este constituit de decizia cu privire la ce informație va fi eliminată din celula de stare. Această decizie este luată de stratul sigmoid, numit și „stratul porții de omisiune” (**engl.: forget state gate**). Acesta privește la ieșirea anterioară h_{t-1} și intrarea curentă x_t și trimite la ieșire un număr înre **0** și **1** pentru fiecare număr din celula de stare C_{t-1}. Un 1 reprezintă păstrarea informației, în timp ce un 0 reprezintă eliminarea completă a acesteia.

Revenind la exemplul anterior cu modelul de limbă ce încearcă să prezică următorul cuvânt bazându-se pe cele anterioare. Într-o astfel de problema, celula de stare poate include genul subiectului prezent, astfel încât pronumele corect să poată fi folosit. Când se întâlnește un nou subiect, se oprește uitarea genului subiectului anterior.



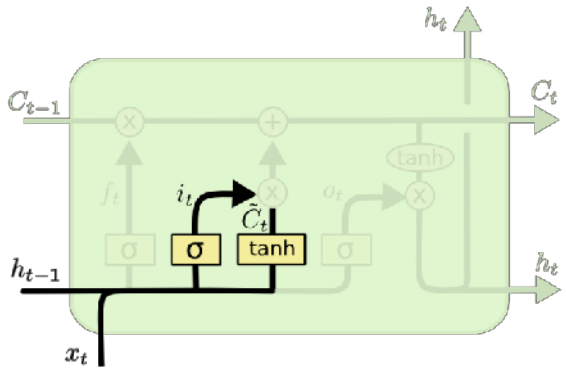
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figura 2.6.1 Poarta de uitare

Următorul pas este să decidem ce informație nouă dorim să stocăm în celula de stare. Acest lucru se realizează în doi pași:

1) Un strat de intrare numit „stratul de intrare al porții” (***engl.: input gate layer***) decide ce valori vor fi actualizate. Apoi, un strat de tip tangentă hiperbolică crează un vector de valori candidate $\tilde{C}_t$, ce poate fi adăugat stării.
2) Se vor combina cele două straturi pentru a realiza o actualizare a stării.

În cadrul exemplului cu modelul de limbă, se dorește adăugarea genului noului subiect în cadrul celulei de stare, pentru a-l înlocui pe cel care se uită.



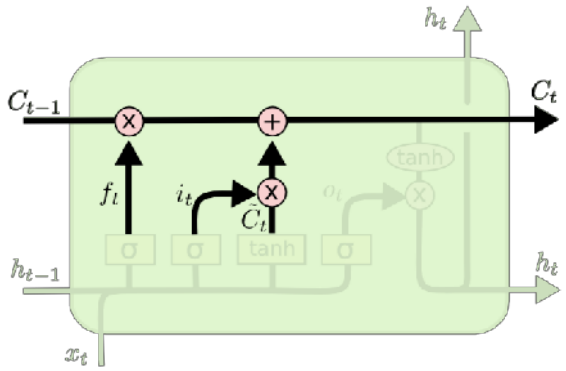
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figura 2.6.2 Stratul de intrare și tanh

În continuare se va actualiza vechea celulă de stare C_{t-1} într-o nouă celulă de stare C_t. Pașii anteriori au decis deja ce este de făcut, acum fiind nevoie de realizarea practică a acestora.

Se va înmulți vechea stare cu f_t, eliminând informația nedorită. Apoi se va adăuga $i_t * \tilde{C}_t$. Acestea reprezintă noile valori candidate, scalate în funcție de cât de mult se dorește actualizarea fiecărei valori de stare.

În cazul modelului de limbă, aici este punctul unde aruncăm informația despre genul subiectului vechi și adăugăm noua informație, așa cum s-a decis la pașii anteriori.

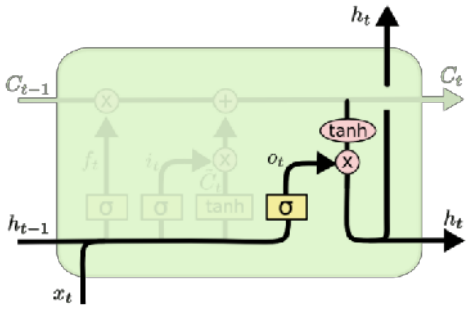


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figura 2.6.3 Actualizare C_t

În cele din urmă, va trebui să se decidă ce informații vor fi disponibile la ieșire. Această ieșire se va baza pe celula de stare, dar va constitui o versiune filtrată a acesteia. În primul rând, se va folosi un strat sigmoid ce va decide care componente ale celulei de stare vor contribui la ieșire. Apoi celula de stare va fi supusă unei operații de tangentă hiperbolică (pentru a obține valori între **-1** și **1**) și ieșirea acesteia va fi înmulțită cu ieșirea porții sigmoide, astfel ieșirea constituind numai informațiile dorite.

Pentru exemplul cu modelul de limbă, din moment ce abia a văzut subiectul, ar putea oferi ca ieșire un verb, în cazul în care acesta ar urma în frază. Spre exemplu, va oferi la ieșire datorită faptului că subiectul este la singular sau plural, conjugarea corectă a verbului, în cazul în care acesta urmează în conversație.



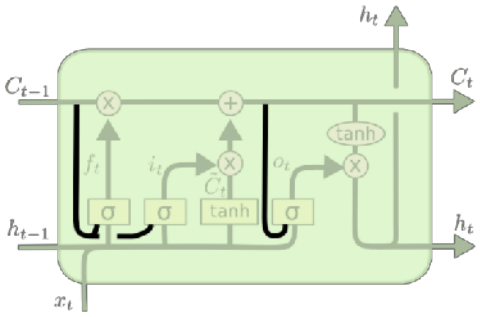
$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figura 2.6.4 Ieșirea LSTM-ului

2.7 Tipuri de LSTM-uri întâlnite în practică

Aproape fiecare abordare a LSTM-urilor este diferită de alta, cea anterioară reprezentând o rețea LSTM obișnuită. Diferențele sunt minore, dar trebuie totuși menționate.

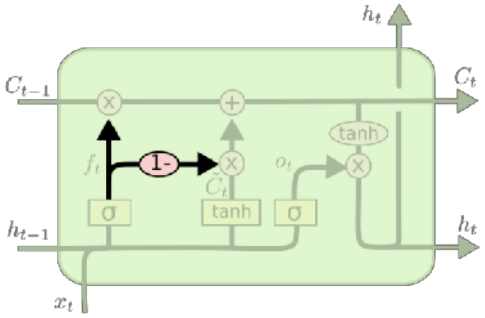
O variantă populară de LSTM, introdusă de Gers și Schmidhuber în anul 2000, este caracterizată de adăugarea „conexiunilor de tip vizor". (**engl.: peephole connections**). Acestea permit straturilor porții să privească către celula de stare.[4]



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t]\ +\ b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t]\ +\ b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t]\ +\ b_o\right)$$

Figura 2.7.1 Adaugarea de conexiuni de tip vizor tuturor porților.

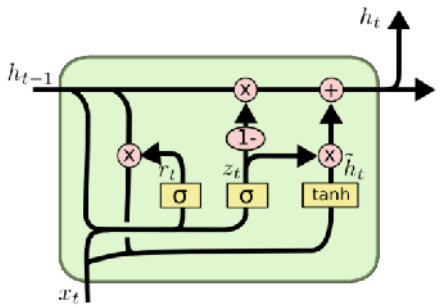
O altă variantă este reprezentată de cuplare porților de intrare și omisiune (**engl.: input gate, forget gate**). Astfel, deciziile cu privire la ce trebuie omis și ce trebuie adăugat nu vor mai fi luate separat, ci vor fi efectuate împreună. Astfel se va uita numai atunci când se vor introduce date. Se vor insera valori noi în stare doar în momentul în care se uită o informație mai veche.



$$C_t = f_t * C_{t-1} + (1 - \boldsymbol{f_t}) * \tilde{C}_t$$

Figura 2.7.2 LSTM cu input gate și forget gate cuplate

O altă variantă mai impresionantă a LSTM-urilor este reprezentat de GRU (***engl: Gated Recurrent Unit***), introdusă de Cho în anul 2014. Aceasta combină porțile de omisiune și intrare într-o singură „poartă de actualizare". De asemnea unește celula de stare și stratul ascuns plus alte câteva modificări. Modelul rezultat este unul mai simplu decât modelele LSTM standard și a devenit din ce în ce mai popular.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figura 2.7.3 Gated Recurrent Unit

Se mai pot remarca câteva variante de LSTM, printre care reamintim „Depth Gated RNNs" introdusă de Yao în anul 2014. De asemenea există abordări diferite care rezolvă dependențele de lungă durată cum ar fi „Clockwork RNNs " introdusă de Koutnik în anul 2014.

Cu toate acestea, LSTM-urile tind să se comporte similar, acestea diferențiindu-se în funcție de sarcina care se dorește a fi îndeplinită.

2.8 Descriere pe larg a celulei LSTM

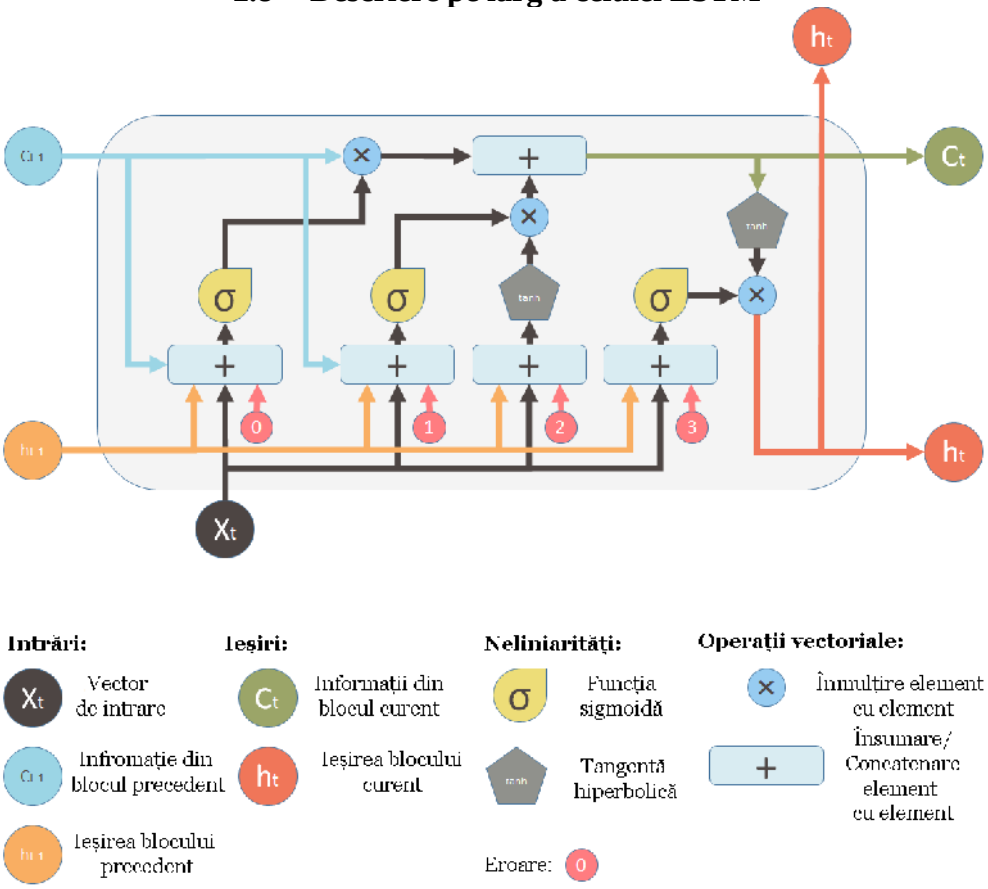


Figura 2.8.1 Diagrama unei rețele LSTM

Rețeaua neuronală dispune de trei intrări:

- X_t - intrarea la momentul curent de timp.
- h_{t-1} – ieșirea din celula LSTM precedentă
- C_{t-1} – informația din celula anterioară (extrem de importantă)

Informația – ceea ce s-a memorat anterior.
Astfel această celulă ia decizii considerând intrările curentă, anterioară și informația precedentă.

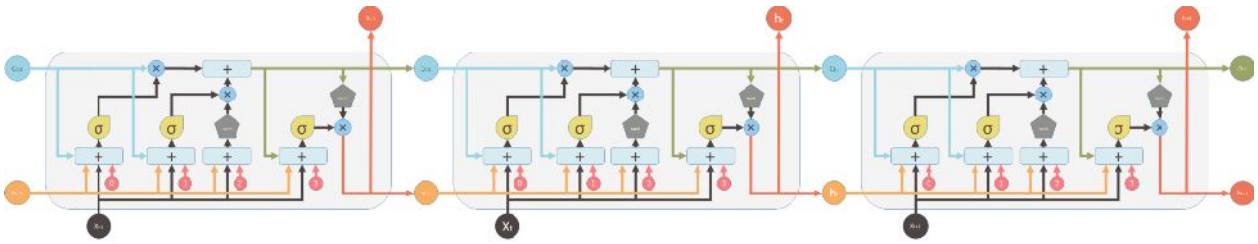


Figura 2.8.2 Celule LSTM

Informația C_t variază similar cu pomparea apei printr-o conductă. Astfel dacă se va asocia cu apa ce curge prin conductă, curgerea acesteia este controlată de două porți:

Figura 2.8.3 Conductă

1) Poarta de uitare (***engl: forget gate***)
   - daca este închisă, informația veche este înlăturată
   - daca este deschisă, informația veche este păstrată
2) poarta informației noi:

Figura 2.8.4 Articulație sub formă de T

- Informația nouă intră printr-o articulație în formă de T similară cu cea din Figura 2.8.4 și se amestecă cu informația veche. Cantitatea de informație nou introdusă este controlată de cea de-a doua poartă.

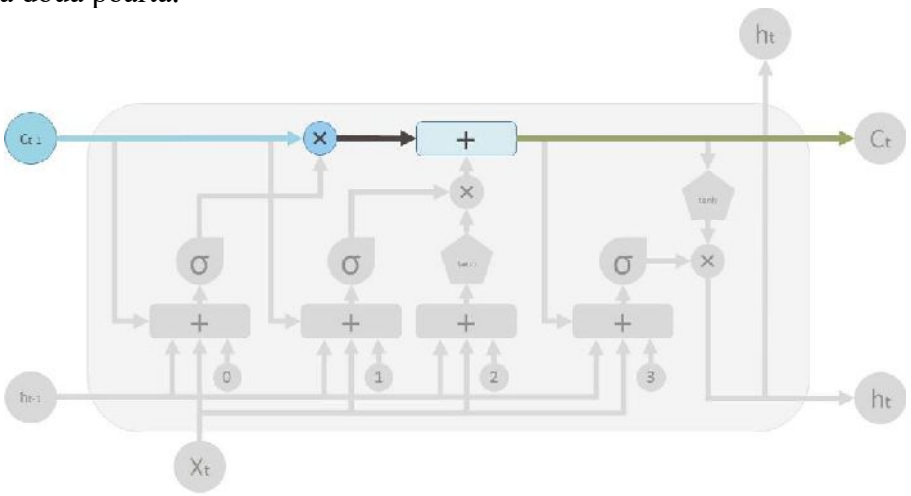


Figura 2.8.5 Traseul informației

În Figura 2.8.5, calea din zona superioară a LSTM-ului reprezintă traseul informației. Intrarea este reprezentată de informația veche (un vector). Primul X din cadrul acesteia reprezintă poarta de uitare, reprezentând de fapt o multiplicare element cu element. Astfel dacă se inmulțește informația veche C_{t-1} cu un vector de valoare aproximativă egală cu 0, atunci toată informația veche va fi uitată. În schimb dacă vectorul are valoarea egală cu 1 atunci toată informația veche este procesată. [3]

Cea de-a doua operație prin care trece informația este reprezentată de operatorul „+” (însumare pe bucăți). Aceasta se aseamănă cu o articulație în formă de T. Informația veche și cea nouă se vor contopi prin intermediul acestei operații. Cantitatea de informație nouă care trebuie adăugată informației vechi este controlată de o altă poartă, „**X**” aflat sub semnul „+”.

După aceste două operații, informația veche C_{t-1} este schimbată în noua informație C_t.

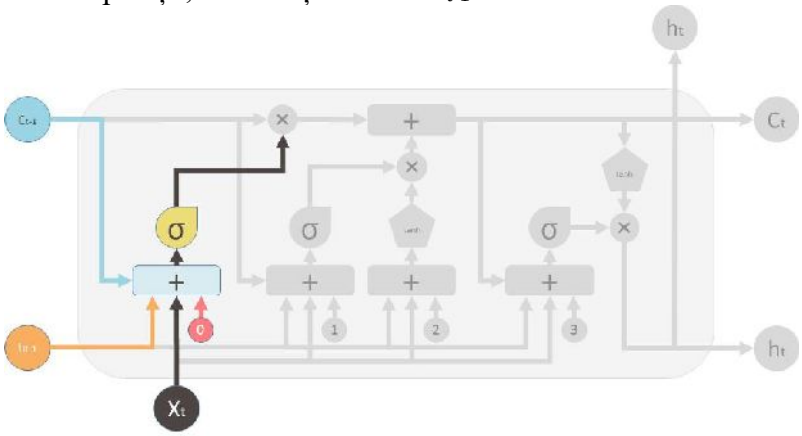


Figura 2.8.6 Contopirea infromației vechi cu informația nouă

În urma analizei celor două porți, prima dintre acestea reprezintă poarta de uitare. Aceasta este controlată de o rețea neuronală cu un singur strat, iar intrările acestei rețelei sunt reprezentate de:

1) h_{t-1} – ieșirea blocului LSTM anterior
2) X_t – intrarea blocului LSTM curent
3) C_{t-1} – informația din blocul anterior
4) b_0 – vectorul de eroare (bias)

Această rețea neuronală utilizează ca funcție de activare , funcția sigmoidă, iar ieșirea acesteia este reprezentată de poarta de uitare (engl: **forget gate**), care se va aplica informației vechi C_{t-1} prin intermediul unei înmulțiri element cu element.

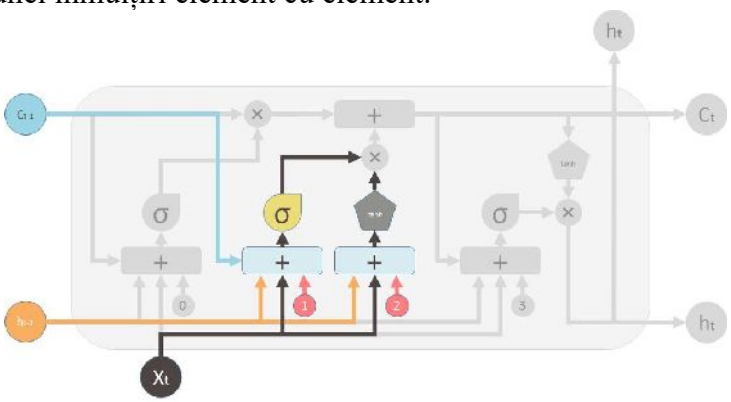


Figura 2.8.7 Controlul informației vechi asupra informației noi

Cea de-a doua poartă este denumită poarta informației noi, iar aceasta la rândul ei este reprezentată de o rețea neuronală simplă cu un singur strat ce are aceleași intrări ca și poarta de uitare. Aceasta va controla în ce măsură informația veche va influența informația nouă. [3]

Informația nouă este generată de o rețea neuronală uni-strat ce folosește ca activator funcția tangenta hiperbolică. Ieșirea acestei rețele va înmulți element cu element noua valoare a porții de informație și va adăuga vechea informație pentru a forma noua informație.

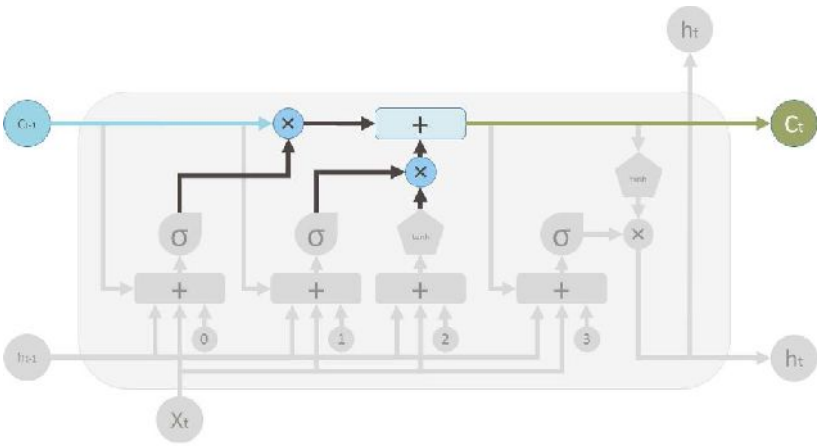


Figura 2.8.8 Poarta de uitare și poarta informației noi

Cele două simboluri ✖ reprezintă poarta de uitare și poarta informației noi.

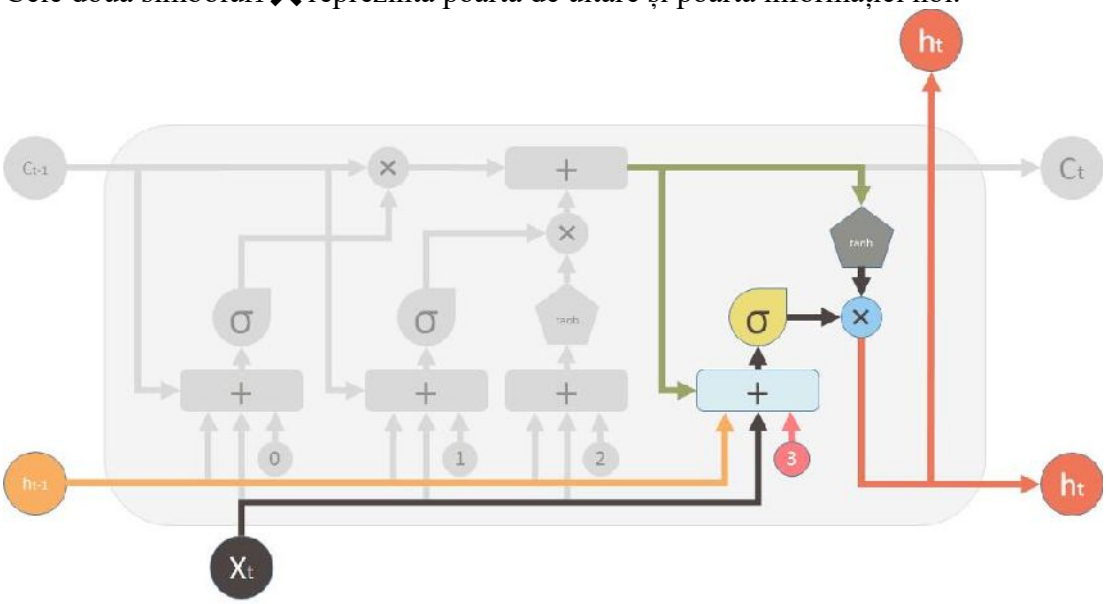


Figura 2.8.9 Ieșirea LSTM-ului

În final va trebui generată ieșirea celulei LSTM. Acest pas are o poartă de ieșire care este controlată de informația nouă, de ieșirea anterioară h_{t-1} , intrarea X_t și vectorul eroare. Această poartă controlează cantitatea de informație ce va fi transmisă următoarei celule LSTM.[3]

C_{t-1} face parte din celula anterioară, în timp ce C_t este parte constituentă a ieșirii.

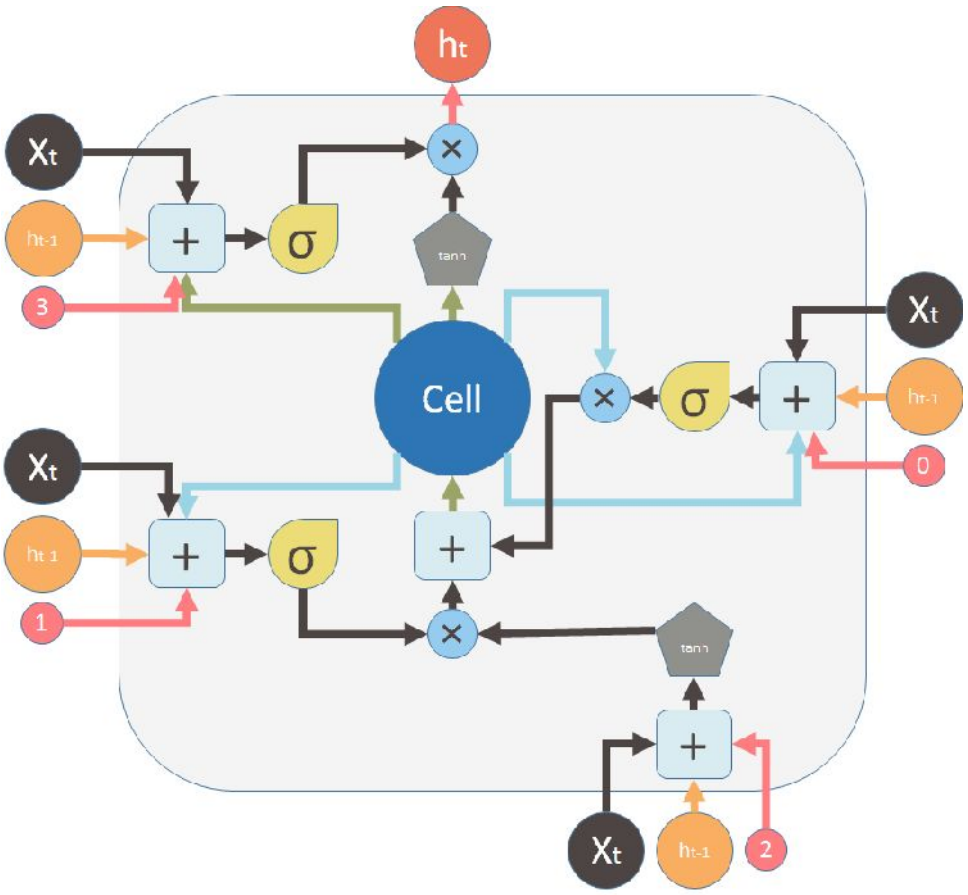


Figura 2.8.10 Diagrama celulei LSTM

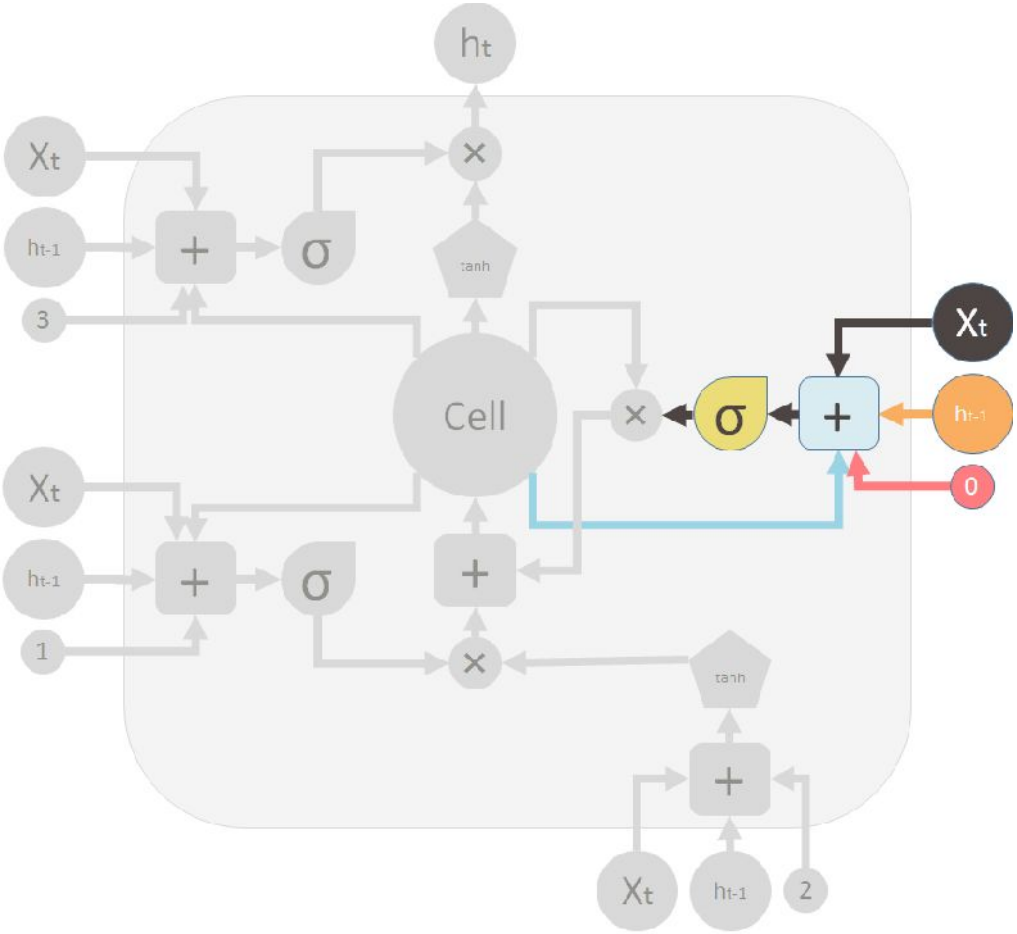


Figura 2.8.11 Poarta de uitare ce elimină din informația veche:

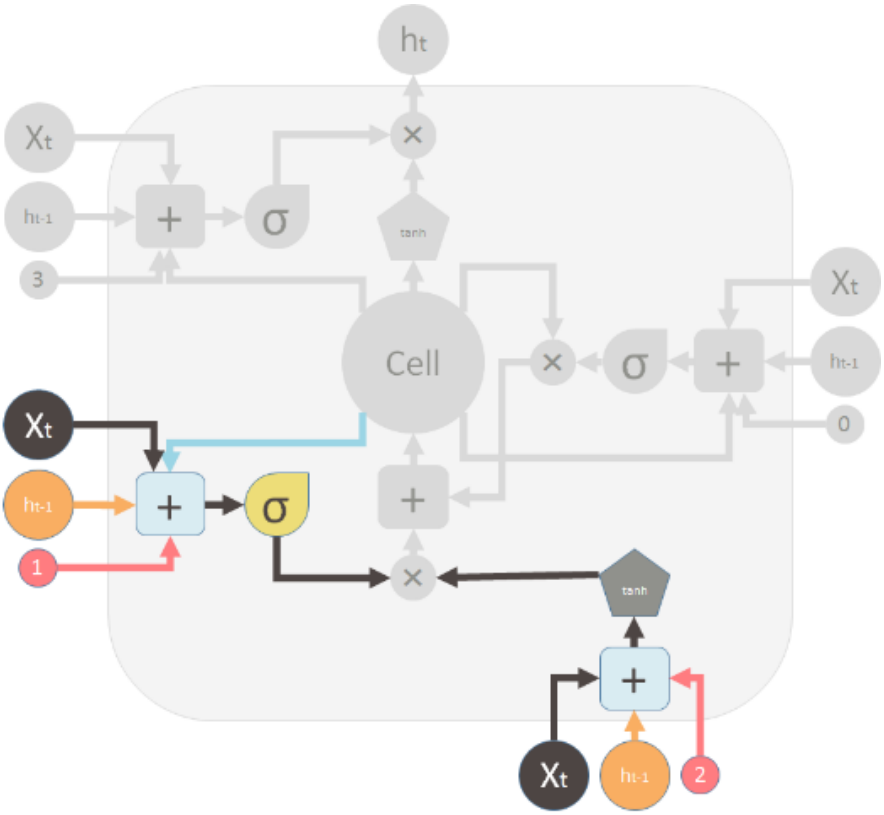


Figura 2.8.12 Informația veche și informația nouă

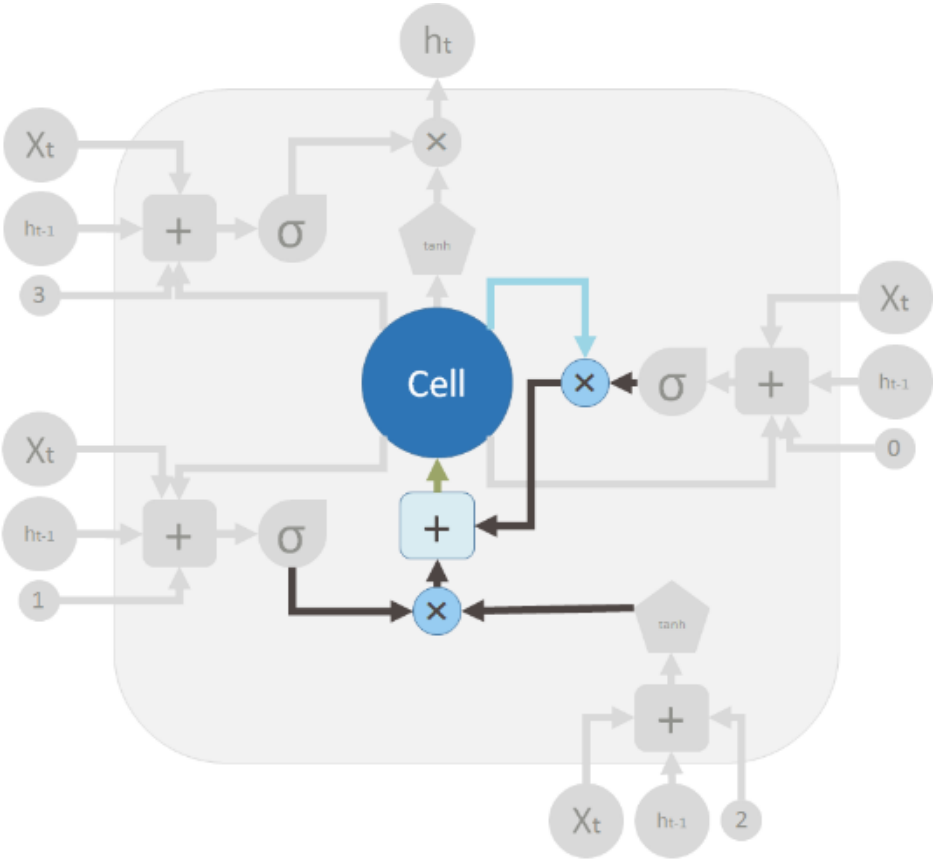


Figura 2.8.13 Contopirea informației noi și a celei vechi în cadrul celulei

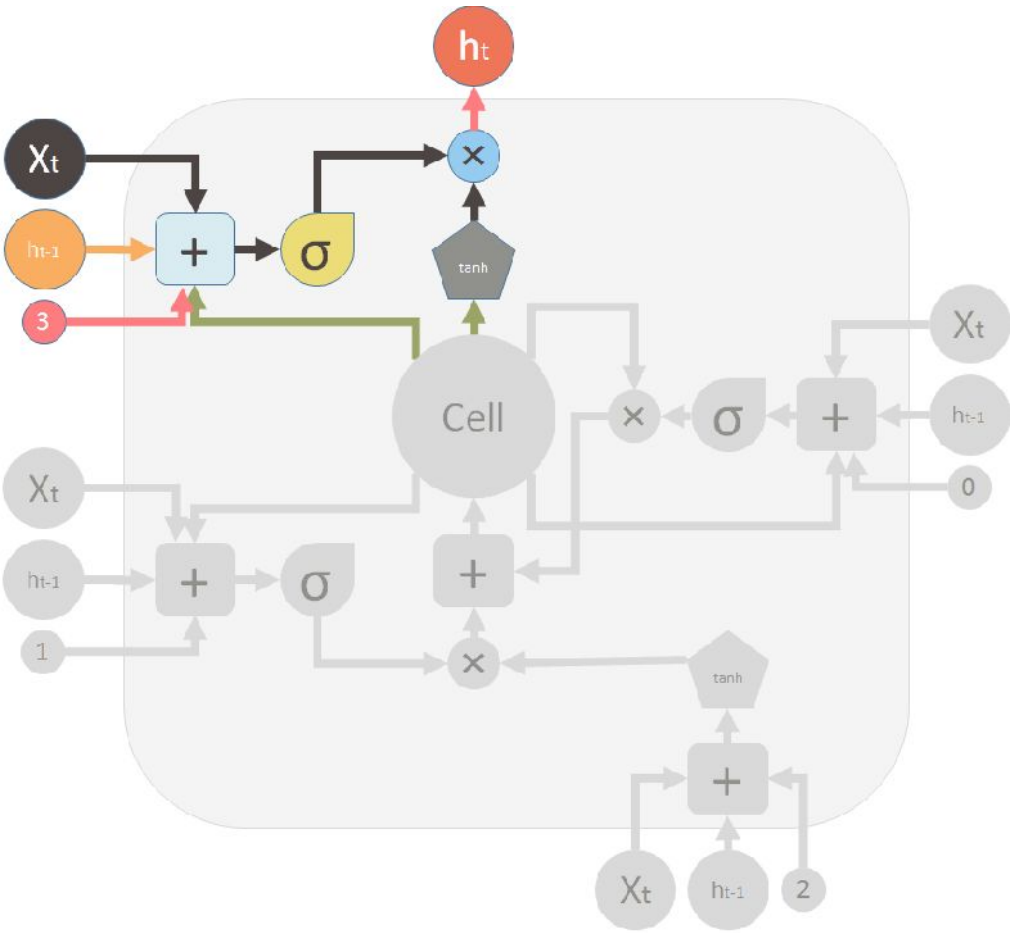


Figura 2.8.14 Poarta de ieşire şi ieşirea celulei LSTM

3. Rezultate experimentale

În cadrul experimentelor s-au folosit 3 baze de date (serii temporale), după cum urmează:

1) Date seismice – constituită din 2 caracteristici ale cutremurului din Oklahoma din 6.11.2011, ce avut o magnitudine de 5.6 pe scara Richter:

1.1) **Accelerația solului** – având **34200** de eșantioane exprimate sub formă de numere reale (float)

1.2) **Viteza solului** – având **34200** de eșantioane exprimate sub formă de numere reale (float)

Aceste date au fost preluate de la National Earthquake Information Center (**NEIC**)

2) Vânzări – constituită din numărul de bilete vândute de o agenție de turism în decursul fiecărei luni pe o perioadă de **12 ani** începând cu ianuarie 1949 până în decembrie 1960, fiind constituită din **144** de eșantioane și preluată de la **Time Series Data Library.**

3) Curs Valutar BNR – prezintă evoluția cursului Euro-Lei pe o perioadă de un an și este constituită din **249** de eșantioane și este preluată de la Banca Națională a României

Folosind un lot de 67% din numărul total de eșantioane pentru antrenare și 33% pentru testare a unor rețele neurale de tip LSTM, având un anumit număr de epoci (iterații) asupra loturilor și respectiv un look back variabil (numărul de eșantioane precedente ce ajută la realizarea predicției) s-au obținut următoarele rezultate (exprimate sub forma unor abateri medii pătratice) și grafice.

În realizarea acestor exprimente un rol important l-a jucat structura și modul de funcționare al LSTM-urilor [3].

3.1 Date seismice - Accelerație

3.1.1 Rezultate all-in-one:

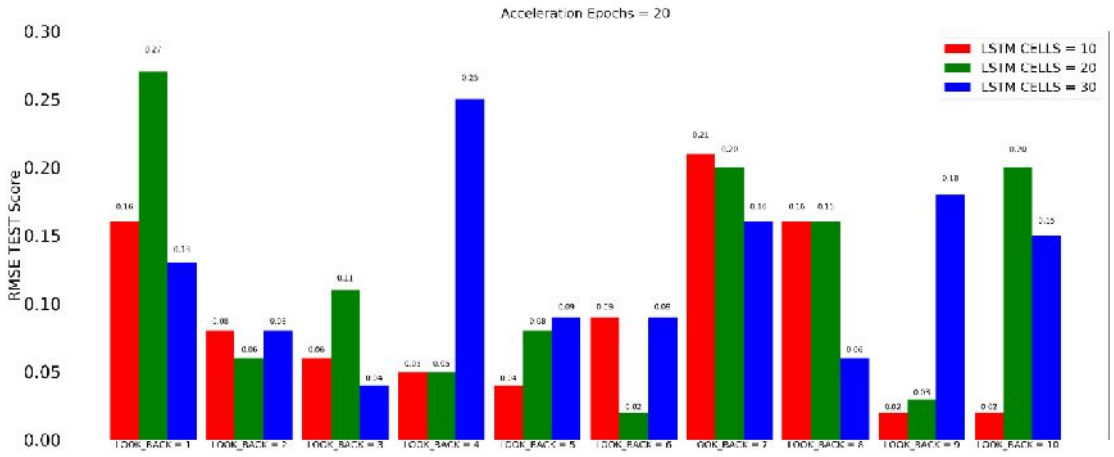


Figura 3.1.1.1 Rezultate Date seismice – Accelerație: 10, 20, 30 celule LSTM, 20 epoci, un look back variabil (1-10)

În Figura 3.1.1.1 sunt ilustrate rezultatele în urma antrenării unor rețele cu 10, 20, 30 celule LSTM și folosind un număr de **20 de epoci**, variind pentru fiecare caz look back-ul de la 1 la 10. Se observă, faptul că rezultatul cel mai bun a fost obținut în cazul în care **look back-ul = 6** și numărul de celule **LSTM = 20**.

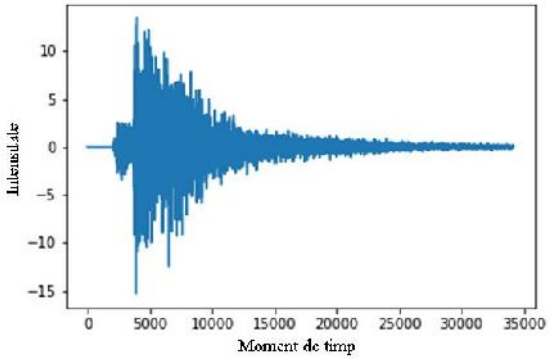


Figura 3.1.1.2 Grafic – intensitate accelerație la fiecare moment de timp (34200 momente de timp)

În Figura 3.1.1.2 este reprezentată evoluția în timp a accelerației în cele 34200 momente de timp.

Deoarece s-a obținut un rezultat foarte bun anterior, se dorește varierea numărului de epoci (iterații) asupra bazei de date și a se observa influența acestei variații asupra rezultatelor experimentale.

3.1.2 Rezultate „best-case”

Acceleration best case

Epochs	LSTM cells	look back	Train RMSE Score	Test RMSE Score
10	20	6	0.22	0.03
20	20	6	0.28	0.21
30	20	6	0.19	0.1
40	20	6	0.16	0.01
50	20	6	0.17	0.08

Tabel 3.1.2.1 Rezultate optime folosind o rețea cu 20 celule LSTM, look-back = 6 și variind numărul de epoci.

Astfel, se observă că prin majorarea numărului de epoci, se obține un rezultat mult mai bun (cazul cu 40 de epoci), ceea ce facilitează predicția.

Un număr de epoci mult prea mare conduce la rezultate mult mai slabe (over-fitting, modelul nu învață datele, doar le memorează).

În figurile Figura 3.1.2.1 - Figura 3.1.2.5 sunt reprezentate datele învățate (cu portocaliu în proporție de 67 %), iar cu verde datele previzionate (în proporție de 33%).

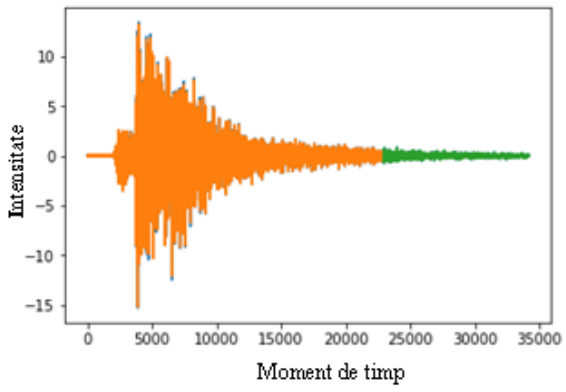


Figura 3.1.2.1 Look back = 6, 20 celule LSTM, 10 Epoci

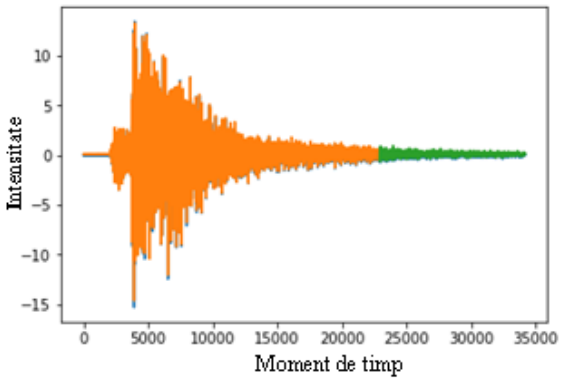


Figura 3.1.2.2 Look back = 6, 20 celule LSTM, 20 Epoci

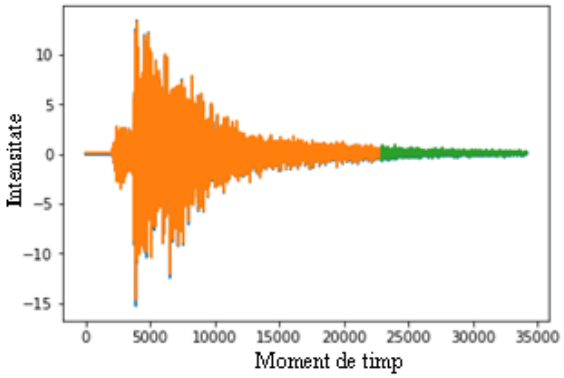


Figura 3.1.2.3 Look back = 6, 20 celule LSTM, 30 Epoci

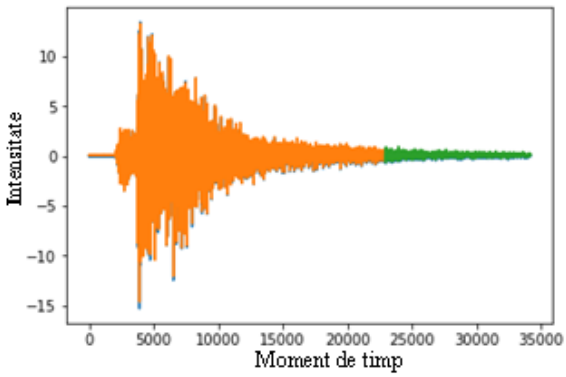


Figura 3.1.2.4 Look back = 6, 20 celule LSTM, 40 Epoci

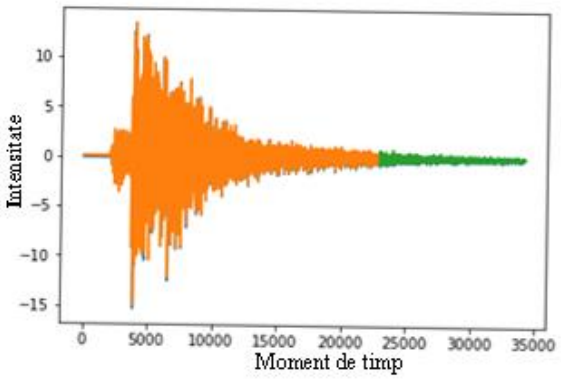


Figura 3.1.2.5 Look back = 6, 20 celule LSTM, 50 Epoci

3.2 Date seismice – Viteză

3.2.1 Rezultate all-in-one:

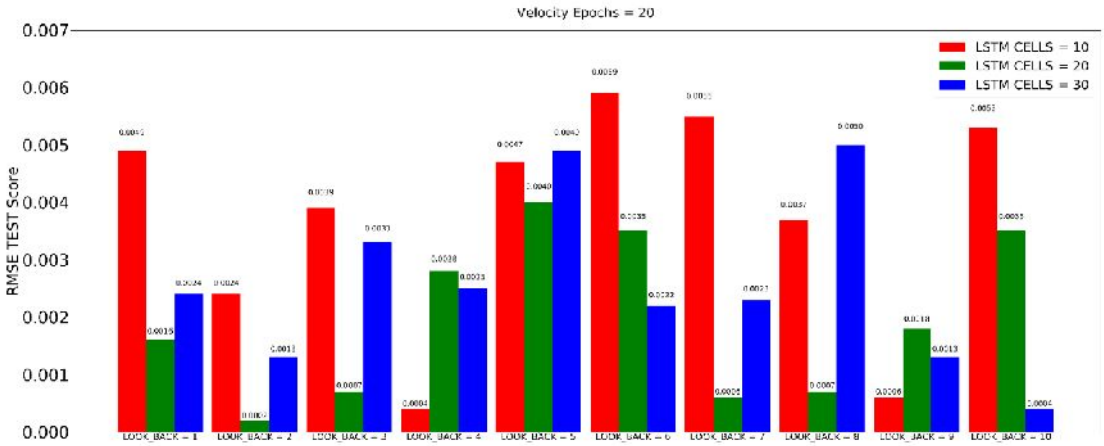


Figura 3.2.1.1 Grafic erori predicție folosind o rețea cu 10,20,30 celule LSTM, 20 de epoci și un look back variabil (1-10)

În Figura 3.2.1.1 sunt ilustrate rezultatele în urma antrenării unor rețele cu 10, 20, 30 celule LSTM și folosind un număr de **20 de epoci**, variind pentru fiecare caz look back-ul de la 1 la 10. Se observă, faptul că rezultatul cel mai bun a fost obținut în cazul în care **look back-ul = 2** și numărul de celule **LSTM = 20.**

Deoarece s-a obținut un rezultat foarte bun anterior, se dorește varierea numărului de epoci (iterații) asupra bazei de date și a se observa influența acestei variații asupra rezultatelor experimentale.

Astfel, se observă că prin majorarea numărului de epoci, nu se obține nicio îmbunătățire asupra rezultatului.

Un număr de epoci mult prea mare conduce la rezultate mult mai slabe (over-fitting, modelul nu învață datele, doar le memorează).

3.2.2 Rezultate „best-case”

Velocity best case				
Epochs	**LSTM cells**	**look back**	**Train RMSE Score**	**Test RMSE Score**
10	20	2	3×10^{-3}	8×10^{-4}
20	20	2	2.6×10^{-3}	1.9×10^{-4}
30	20	2	4.3×10^{-3}	3×10^{-3}
40	20	2	3.8×10^{-3}	2×10^{-3}
50	20	2	5×10^{-3}	4×10^{-3}

Tabel 3.2.2.1 Rezultate optime date seismice - viteză , 20 de celule LSTM, look back = 2, număr de epoci variabil

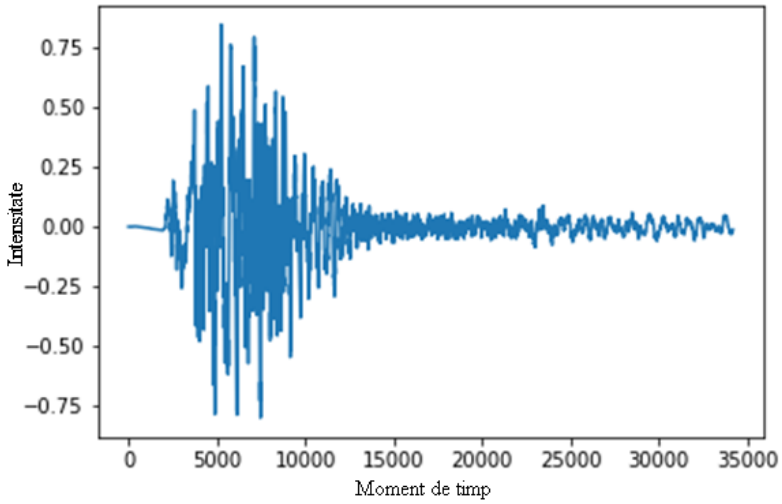


Figura 3.2.2.1 Grafic viteza solului

În Figura 3.2.2.1 este reprezentată evoluția în timp a vitezei în cele 34200 momente de timp.

În figurile Figura 3.2.2.2 - Figura 3.2.2.6 sunt reprezentate datele învățate (cu portocaliu în proporție de 67 %), iar cu verde datele previzionate (în proporție de 33%).

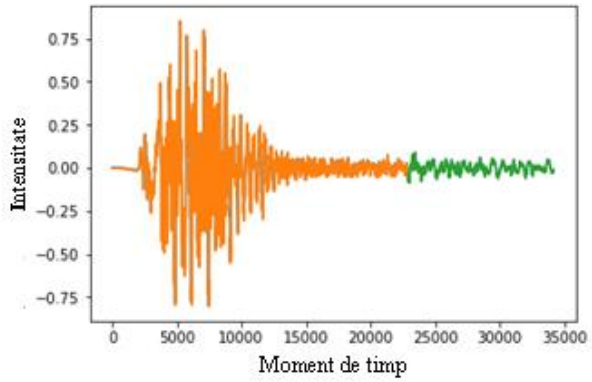


Figura 3.2.2.2 Look back = 2, 20 celule LSTM, 10 Epoci

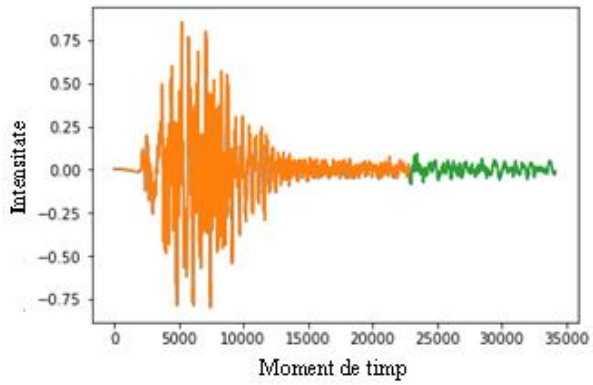


Figura 3.2.2.3 Look back = 2, 20 celule LSTM, 20 Epoci

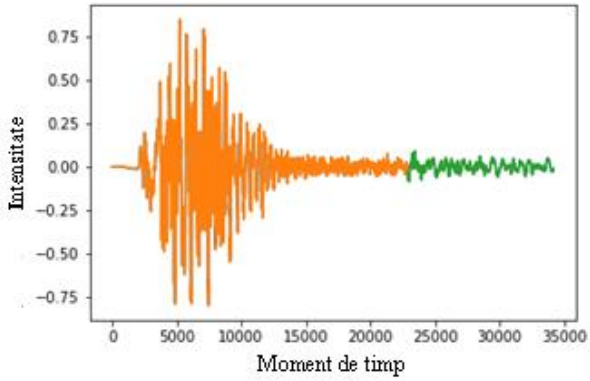


Figura 3.2.2.4 Look back = 2, 20 celule LSTM, 30 Epoci

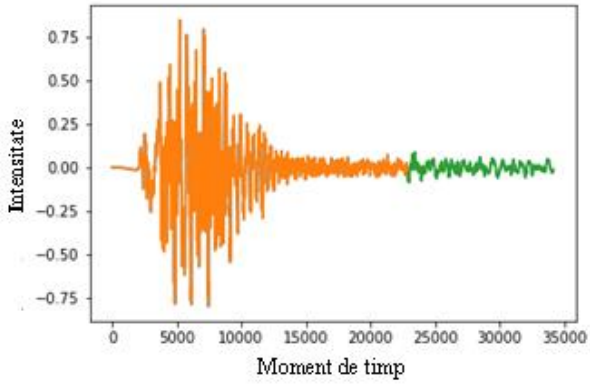


Figura 3.2.2.5 Look back = 2, 20 celule LSTM, 40 Epoci

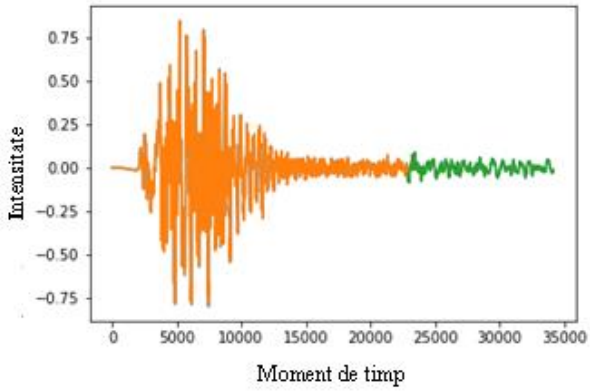


Figura 3.2.2.6 Look back = 2, 20 celule LSTM, 50 Epoci

3.3 Companie aeriană

3.3.1 Rezultate all-in-one

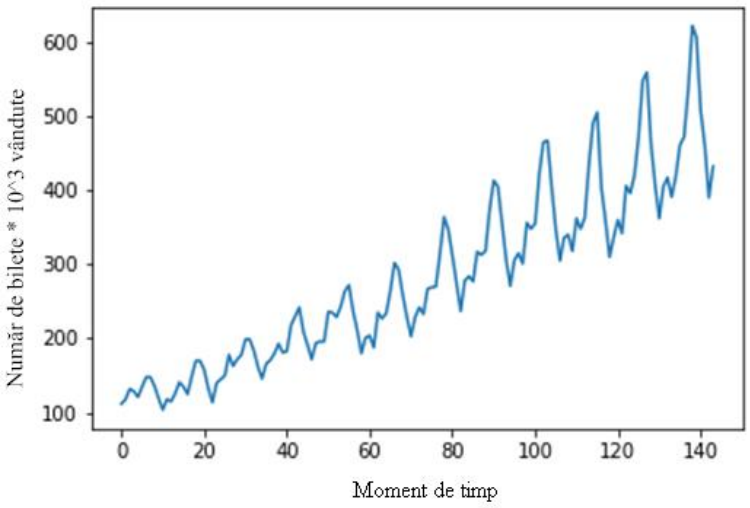


Figura 3.3.1.1 Grafic bilete vândute în 12 ani de către companie

În Figura 3.3.1.1 este reprezentată evoluția în timp a numărului de bilete vândute exprimată în unități de ordinul miilor în cadrul celor 144 de luni (momente de timp)

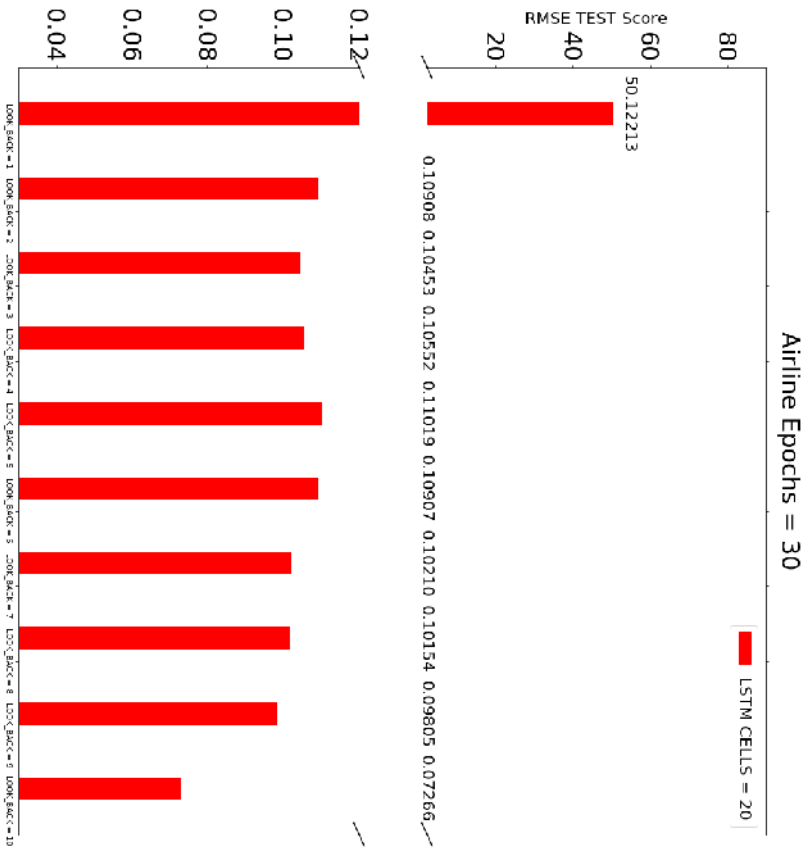


Figura 3.3.1.2 Grafic erori predicție, 20 celule LSTM , 30 epoci, look_back variabil (1-10)

În Figura 3.3.1.2 sunt reprezentate rezultatele obținute în urma antrenării unei rețele cu 20 de celule LSTM, utilizând 30 de epoci și având un look back variind între 1 și 10. Se poate observa o eroare foarte mare în cadrul utilizării unui look back = 1 datorită caracterului fluctuant al datelor (se observă în figura că datele nu variază liniar) și astfel utilizarea unei singure valori anterioare conduce la rezultate eronate.

În cazul acestei serii temporale, un look_back = 10 și un număr de 30 de epoci s-au dovedit suficiente pentru o obține o predicție bună.

Airline				
LSTM Cells	Look_back	Epochs	Train Score RMSE	Test Score RMSE
20	1	30	23,46	50,12
20	2	30	4.98×10^{-2}	1.0907×10^{-1}
20	3	30	4.9×10^{-2}	1.04×10^{-1}
20	4	30	4.859×10^{-2}	1.05×10^{-1}
20	5	30	4.88×10^{-2}	1.1×10^{-1}
20	6	30	4.72×10^{-2}	1.0906×10^{-1}
20	7	30	4.6×10^{-2}	1.02×10^{-1}
20	8	30	4.854×10^{-2}	1.01×10^{-1}
20	9	30	4.49×10^{-2}	9.8×10^{-2}
20	10	30	3.77×10^{-2}	7.2×10^{-2}

Tabel 3.3.1.1 Rezultate predicție, 20 celule LSTM, look back variabil (1-10), 30 epoci

În cadrul tabelului Tabel 3.3.1.1 sunt prezentate erorile în obținute în cadrul rulării celor 10 antrenări, iar în figurile Figura 3.3.1.3 - Figura 3.3.1.12 :

- Cu albastru estre reprezentată evoluția normală a seriei temporale
- Cu portocaliu este reprezentat lotul de antrenare
- Cu verde sunt reprezentate datele previzionate

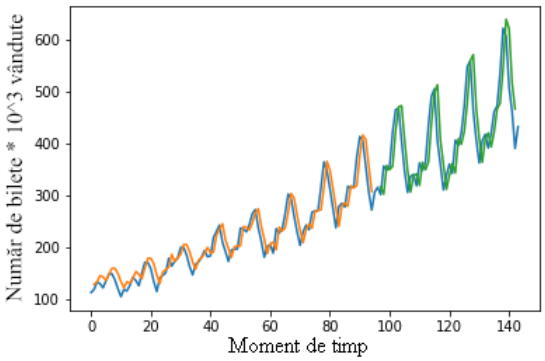


Figura 3.3.1.3 Look back = 1, 20 celule LSTM, 30 Epoci

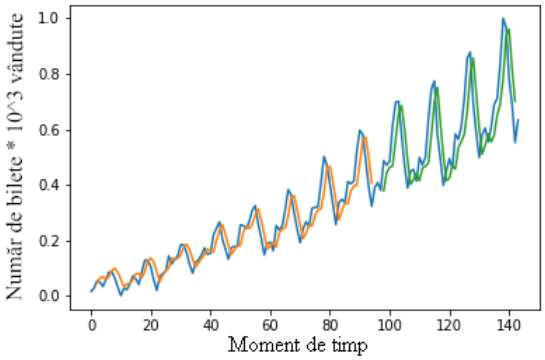


Figura 3.3.1.4 Look back = 2, 20 celule LSTM, 30 Epoci

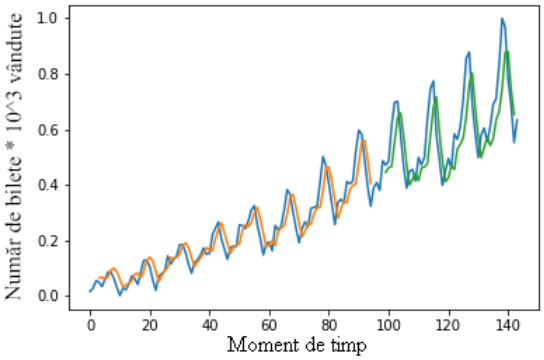


Figura 3.3.1.5 Look back = 3, 20 celule LSTM, 30 Epoci

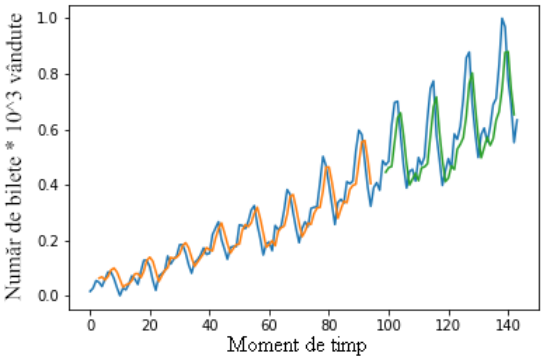


Figura 3.3.1.6 Look back = 4, 20 celule LSTM, 30 Epoci

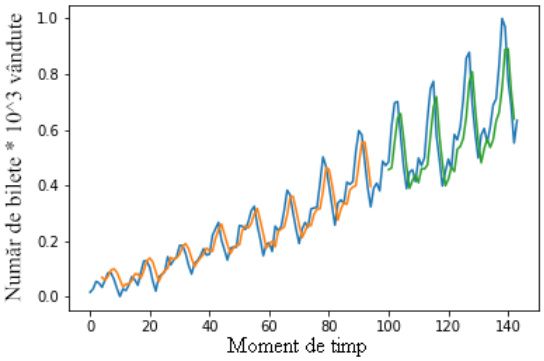


Figura 3.3.1.7 Look back = 5, 20 celule LSTM, 30 Epoci

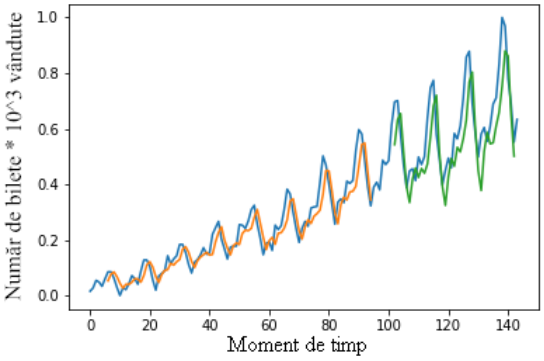


Figura 3.3.1.8 Look back = 6, 20 celule LSTM, 30 Epoci

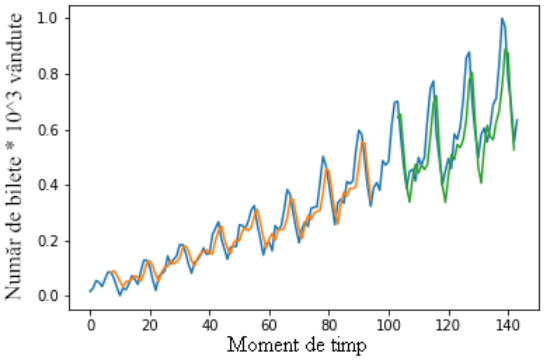


Figura 3.3.1.9 Look back = 7, 20 celule LSTM, 30 Epoci

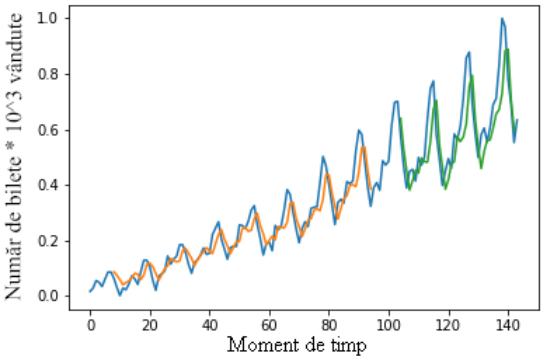


Figura 3.3.1.10 Look back = 8, 20 celule LSTM, 30 Epoci

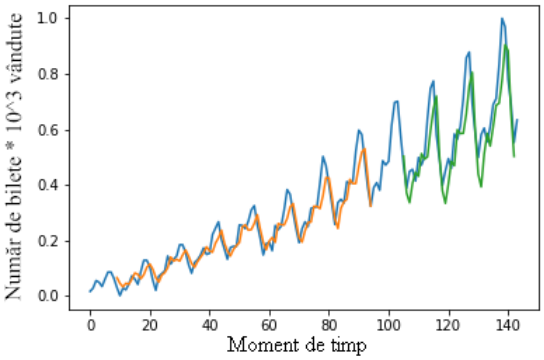


Figura 3.3.1.11 Look back = 9, 20 celule LSTM, 30 Epoci

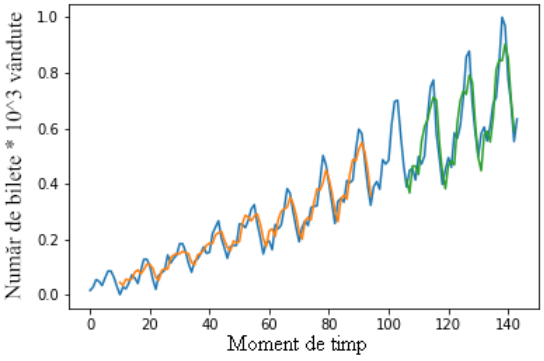


Figura 3.3.1.12 Look back = 10, 20 celule LSTM, 30 Epoci

3.4 Date Finaciare

3.4.1 Rezultate all-in-one

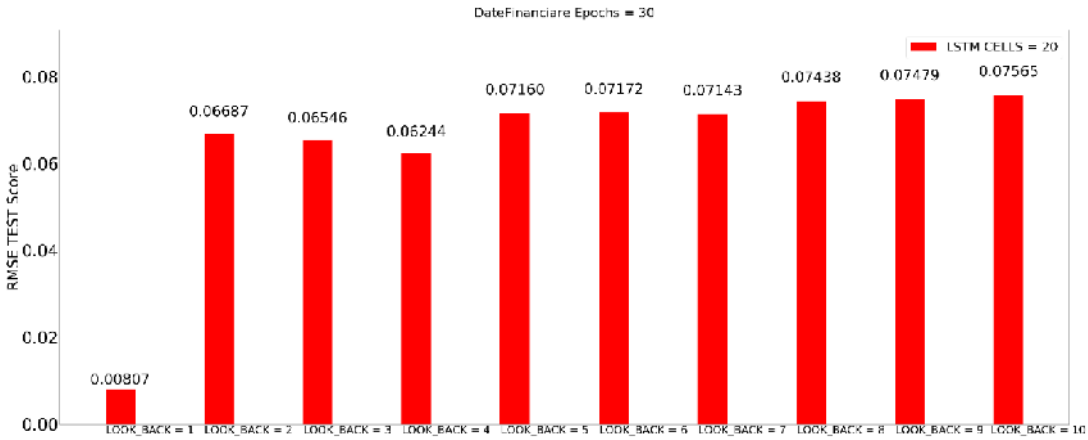


Figura 3.4.1.1 Rezultate Curs Valutar BNR folosind 20 celule LSTM, 30 de epoci și look back variabil (1-10)

În Figura 3.4.1.1 sunt reprezentate rezultatele obținute folosind o rețea cu 20 de celule LSTM de-a lungul a 30 de epoci și având un look back variabil (1-10). Se poate observa că în cazul în care se utilizează un look back = 1 se obține o eroare minimă.

Deoarece baza de date conține valori foarte apropiate, folosirea unui lookback mare conduce la o eroare mai mare.

Curs Valutar BNR				
LSTM Cells	Look_back	Epochs	Train Score RMSE	Test Score RMSE
20	1	30	0.0067	0.008
20	2	30	0.061	0.066
20	3	30	0.0619	0.065
20	4	30	0.0638	0.062
20	5	30	0.0615	0.0715
20	6	30	0.0625	0.0717
20	7	30	0.059	0.0714
20	8	30	0.0621	0.0743
20	9	30	0.0645	0.0747
20	10	30	0.0633	0.0756

Tabel 3.4.1.1 Rezultate Curs Valutar BNR folosind 20 celule LSTM, 30 de epoci și look back variabil (1-10)

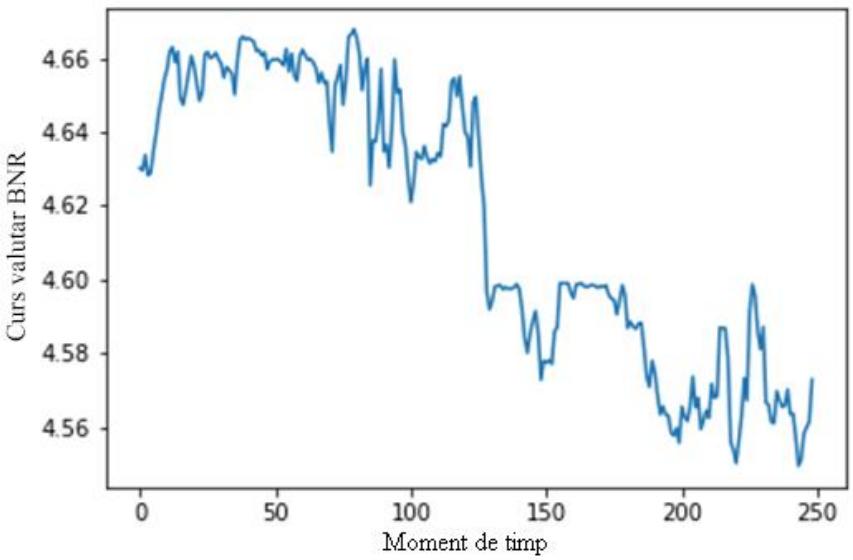


Figura 3.4.1.2 Grafic curs valutar Euro-Lei 18.05.2017 - 18.05.2018

În cadrul graficului din Figura 3.4.1.2 este prezentată evoluția în timp a cursului valutar euro-lei de-a lungul celor 249 momente de timp.

În figurile Figura 3.4.1.3 - Figura 3.4.1.12 sunt reprezentate:

- Cu albastru: evoluția normală a cursului valutar
- Cu portocaliu: lotul de antrenare
- Cu verde: datele previzionate

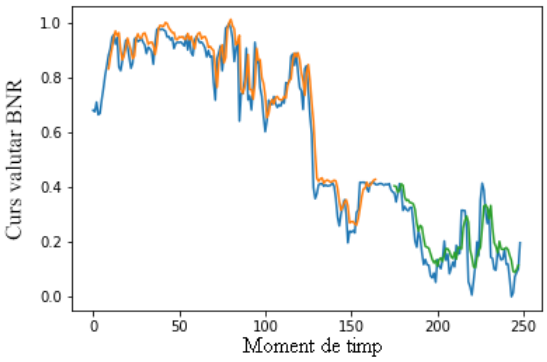


Figura 3.4.1.3 Grafic Curs Valutar BNR, 20 celule LSTM, 30 de epoci, look back = 1

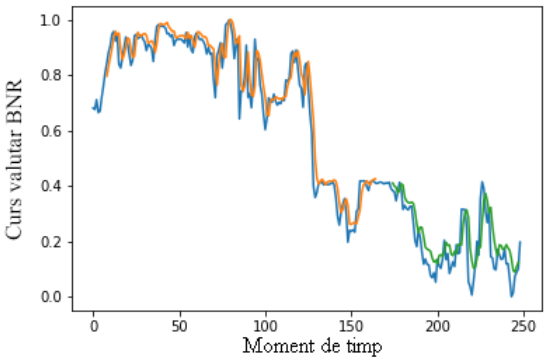


Figura 3.4.1.4 Grafic Curs Valutar BNR, 20 celule LSTM, 30 de epoci, look back = 2

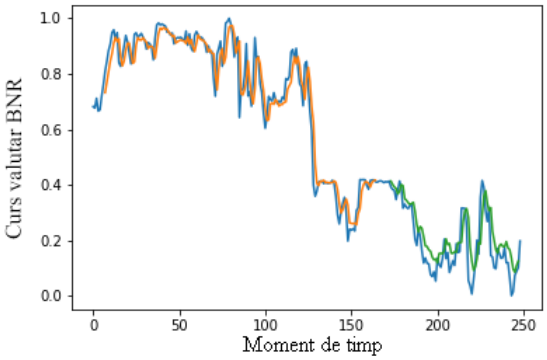


Figura 3.4.1.5 Grafic Curs Valutar BNR, 20 celule LSTM, 30 de epoci, look back = 3

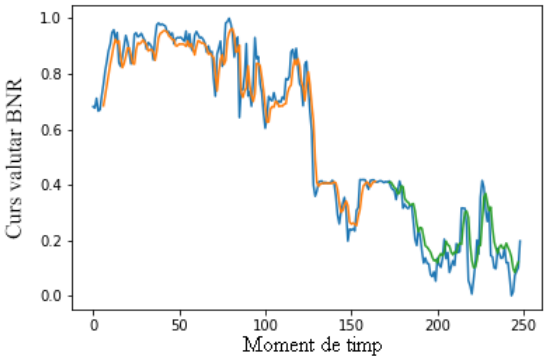


Figura 3.4.1.6 Grafic Curs Valutar BNR, 20 celule LSTM, 30 de epoci, look back = 4

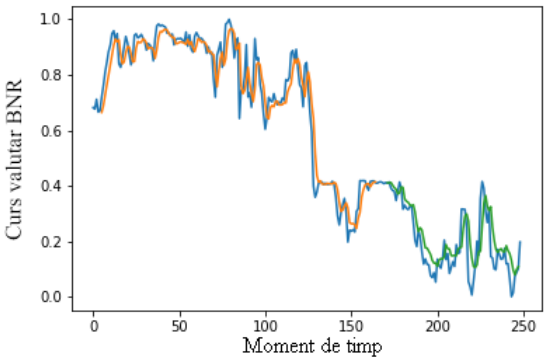


Figura 3.4.1.7 Grafic Curs Valutar BNR, 20 celule LSTM, 30 de epoci, look back = 5

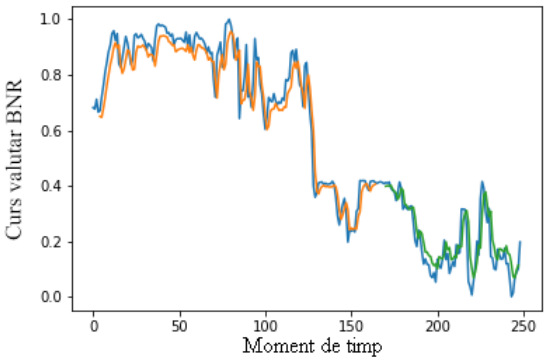


Figura 3.4.1.8 Grafic Curs Valutar BNR, 20 celule LSTM, 30 de epoci, look back = 6

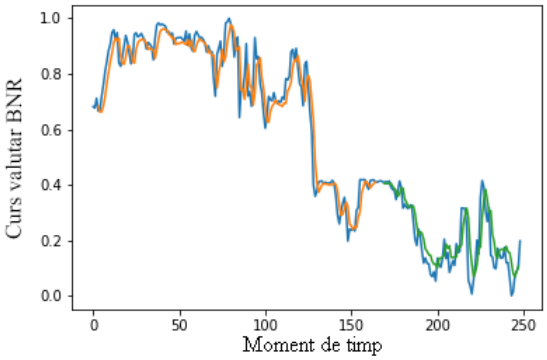


Figura 3.4.1.9 Grafic Curs Valutar BNR, 20 celule LSTM, 30 de epoci, look back = 7

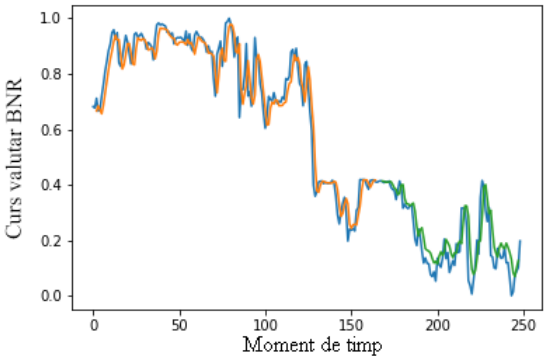


Figura 3.4.1.10 Grafic Curs Valutar BNR, 20 celule LSTM, 30 de epoci, look back = 8

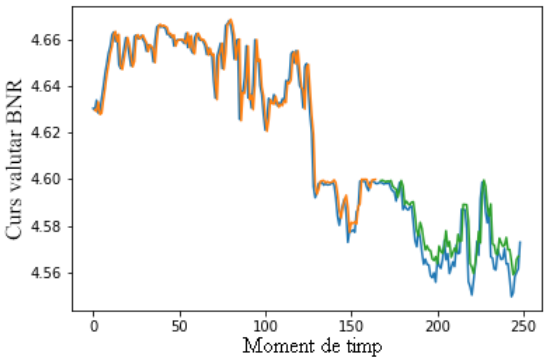


Figura 3.4.1.11 Grafic Curs Valutar BNR, 20 celule LSTM, 30 de epoci, look back = 9

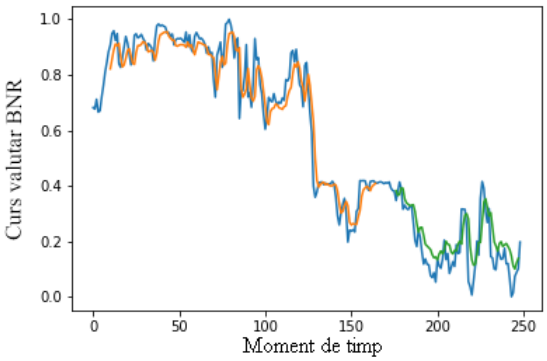


Figura 3.4.1.12 Grafic Curs Valutar BNR, 20 celule LSTM, 30 de epoci, look back = 10

4. Concluzii

În cadrul acestei teze au fost expuse capabilitățile de predicție a unor rețele neurale de tip LSTM, asupra 3 baze de date, structurate sub formă de serii temporale.

Rețelele neurale au o plajă de aplicații foarte mare, acestea ajutând la prevenirea dezastrelor naturale, a controlului cursului valutar, dar și a managementului unei companii.

Prin construirea și antrenarea corectă a rețelelor neurale se pot realiza predicții cât mai apropiate de realitate, acestea ajutând în procesul de luare a unor decizii cheie.

Astfel în urma analizei rezultatelor obținute, putem conchide că un număr foarte mare de epoci, utilizat pe o bază de date nu foarte mare conduce la overfitting, astfel rețeaua neurală este incapabilă să mai extragă datele.

În cazul în care baza de date prezintă fluctuații neliniare (baza de date cu vânzări) este indicată folosirea unei valori de look back cât mai mari, pentru a realiza o predicție cât mai bună, spre deosebire de cazul în care valorile seriei temporale sunt similare (cazul cursului valutar BNR).

Contribuții personale

Proiectul de licență este constituit din aplicații ale rețelelor neurale recurente de tip LSTM asupra seriilor temporale: date seismice (componenta de accelerație și componenta de viteză), vânzări bilete, curs valutar BNR. În cadrul acestei lucrări am avut următoarele contribuții:

- Am efectuat o serie de teste folosind valori diferite de look_back și variind numărul de epoci și respectiv de celule LSTM pentru cele 3 seturi de date pentru observa influența acestor variații asupra datelor previzionate.
- Am facilitat execuția în paralel a testelor prin crearea de sesiuni separate ce folosesc separat memoria plăcii video
- Am creat o serie de algoritmi pentru realizarea graficelor experimentale (all_in_one, best_case), respectiv de spargere a axelor și adăugarea unui semnal sonor la terminarea rulării acestora
- Am exemplificat pentru fiecare serie temporală lotul de antrenare, datele previzionate, evoluția normală a întregului, dar și erorile obținute

5. Bibliografie

[1] „*A Beginner's Guide to Recurrent Networks and LSTMs*" https://deeplearning4j.org/lstm.html. accesat la data: 05.04.2018.

[2] A. Minnaar, „*Deep Learning Basics: Neural Networks, Backpropagation and Stochastic Gradient Descent*," http://alexminnaar.com/deep-learning-basics-neural-networks-backpropagation-and-stochastic-gradient-descent.html. accesat la data 12.05.2018.

[3] S. Yan, „*Understanding LSTM and its diagrams*," https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714. accesat la data 10.03.2018.

[4] „*Understanding LSTM Networks*," http://colah.github.io/posts/2015-08-Understanding-LSTMs/. accesat la data 05.03.2018.

[5] „*Neural Networks Part 1: Setting up the Architecture*," http://cs231n.github.io/neural-networks-1/. accesat la data: 27.04.2018.

6. Anexe

6.1 Cod look_back variabil

6.1.1 Cod look_back variabil accelerație:

```
1. import numpy as np
2. import numpy
3. import matplotlib.pyplot as plt
4. from pandas import read_csv
5. import math
6. from keras.models import Sequential
7. from keras.layers import Dense
8. from keras.layers import LSTM
9. from sklearn.preprocessing import MinMaxScaler
10. from sklearn.metrics import mean_squared_error
11. import pandas as pd
12. from numpy  import array
13. import re
14. 
15. data_file = open("acc.smc")
16. block = list()
17. found = False
18. #(-?\d.\d+E-\d)
19. for line in data_file:
20.     if found:
21.         z=re.findall('-?\d.\d+E-\d|-?\d.\d+E\+\d', line)
22.         for i  in range(0,len(z)):
23.             block.append(z[i]);
24.     else:
25.         if line.strip() == "| Source location: USGS NEIC (WDCS-D)":
26.             found = True
27. 
28. data_file.close()
29. block = array(block)
30. #block = block.reshape(8*len(block),1)
31. block = block[~pd.isnull(block)]
32. block = block.reshape(-1, 1)
33. block = block.astype(float)
34. print(block,len(block))
35. 
36. 
37. def create_dataset(dataset, look_back=1):
38.     dataX, dataY = [], []
39.     for i in range(len(dataset)-look_back-1):
40.         a = dataset[i:(i+look_back), 0]
41.         dataX.append(a)
42.         dataY.append(dataset[i + look_back, 0])
43.     return numpy.array(dataX), numpy.array(dataY)
44. 
45. numpy.random.seed(7)
46. scaler = MinMaxScaler(feature_range=(0, 1))
47. block = scaler.fit_transform(block)
48. 
49. train_size = int(len(block) * 0.67)
50. test_size = len(block) - train_size
51. train, test = block[0:train_size,:], block[train_size:len(block),:]
52. look_back = 1
53. 
54. trainX, trainY = create_dataset(train, look_back)
55. testX, testY = create_dataset(test, look_back)
56. 
57. trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
58. testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
59. 
60. 
61. model = Sequential()
62. model.add(LSTM(20, input_shape=(1, look_back)))
63. model.add(Dense(1))
```

```
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=20, batch_size=1, verbose=2)
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))

trainPredictPlot = numpy.empty_like(block)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict

testPredictPlot = numpy.empty_like(block)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(block)-1, :] = testPredict

plt.plot(scaler.inverse_transform(block))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```

6.1.2 Cod look_back variabil viteză:

```
t_config.gpu_options.allow_growth = True
with tf.Session(config=t_config) as sess:
    import numpy as np
    import numpy
    import matplotlib.pyplot as plt
    from pandas import read_csv
    import math
    from keras.models import Sequential
    from keras.layers import Dense
    from keras.layers import LSTM
    from sklearn.preprocessing import MinMaxScaler
    from sklearn.metrics import mean_squared_error
    import pandas as pd
    from numpy  import array
    import re

    data_file = open("v.smc")
    block = list()
    found = False
    #(-?\d.\d+E-\d)
    for line in data_file:
        if found:
            z=re.findall('-?\d.\d+E-\d|-?\d.\d+E\+\d', line)
            for i  in range(0,len(z)):
                block.append(z[i]);
        else:
            if line.strip() == "| Source location: USGS NEIC (WDCS-D)":
                found = True


    data_file.close()
    block = array(block)
    #block = block.reshape(8*len(block),1)
    block = block[~pd.isnull(block)]
    block = block.reshape(-1, 1)
    block = block.astype(float)
    print(block,len(block))

```

```
    def create_dataset(dataset, look_back=1):
        dataX, dataY = [], []
        for i in range(len(dataset)-look_back-1):
            a = dataset[i:(i+look_back), 0]
            dataX.append(a)
            dataY.append(dataset[i + look_back, 0])
        return numpy.array(dataX), numpy.array(dataY)

    numpy.random.seed(7)
    scaler = MinMaxScaler(feature_range=(0, 1))
    block = scaler.fit_transform(block)

    train_size = int(len(block) * 0.67)
    test_size = len(block) - train_size
    train, test = block[0:train_size,:], block[train_size:len(block),:]
    look_back = 10

    trainX, trainY = create_dataset(train, look_back)
    testX, testY = create_dataset(test, look_back)

    trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
    testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))


    model = Sequential()
    model.add(LSTM(10, input_shape=(1, look_back)))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    model.fit(trainX, trainY, epochs=20, batch_size=1, verbose=2)


    trainPredict = model.predict(trainX)
    testPredict = model.predict(testX)


    trainPredict = scaler.inverse_transform(trainPredict)
    trainY = scaler.inverse_transform([trainY])
    testPredict = scaler.inverse_transform(testPredict)
    testY = scaler.inverse_transform([testY])


    trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
    print('Train Score: %.10f RMSE' % (trainScore))
    testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
    print('Test Score: %.10f RMSE' % (testScore))


    trainPredictPlot = numpy.empty_like(block)
    trainPredictPlot[:, :] = numpy.nan
    trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict


    testPredictPlot = numpy.empty_like(block)
    testPredictPlot[:, :] = numpy.nan
    testPredictPlot[len(trainPredict)+(look_back*2)+1:len(block)-
1, :] = testPredict

    plt.plot(scaler.inverse_transform(block))
    plt.plot(trainPredictPlot)
    plt.plot(testPredictPlot)
    plt.show()

        import winsound
        frequency = 1400  # Set Frequency To 2500 Hertz
        duration = 2000  # Set Duration To 1000 ms == 1 second
        winsound.Beep(frequency, duration)
```

6.1.3 Cod airline look_back variabil:

```
appendFile = open('RMSE_airline.txt','a')
import tensorflow as tf
t_config = tf.ConfigProto()
t_config.gpu_options.allow_growth = True
with tf.Session(config=t_config) as sess:
    import numpy as np
    import numpy
    import matplotlib.pyplot as plt
    from pandas import read_csv
    import math
    from keras.models import Sequential
    from keras.layers import Dense
    from keras.layers import LSTM
    from sklearn.preprocessing import MinMaxScaler
    from sklearn.metrics import mean_squared_error
    import pandas as pd
    from numpy  import array
    import re


    def create_dataset(dataset, look_back=1):
        dataX, dataY = [], []
        for i in range(len(dataset)-look_back-1):
            a = dataset[i:(i+look_back), 0]
            dataX.append(a)
            dataY.append(dataset[i + look_back, 0])
        return numpy.array(dataX), numpy.array(dataY)

    dataframe = read_csv('airline.csv', usecols=[1], engine='python', skipfooter=3
)
    dataset = dataframe.values
    dataset = dataset.astype('float32')
    numpy.random.seed(7)

    for i in range (1,11):

        scaler = MinMaxScaler(feature_range=(0, 1))
        dataset = scaler.fit_transform(dataset)

        train_size = int(len(dataset) * 0.67)
        test_size = len(dataset) - train_size
        train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]


        look_back = i

        trainX, trainY = create_dataset(train, look_back)
        testX, testY = create_dataset(test, look_back)

        trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
        testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))


        model = Sequential()
        model.add(LSTM(20, input_shape=(1, look_back)))
        model.add(Dense(1))
        model.compile(loss='mean_squared_error', optimizer='adam')
        model.fit(trainX, trainY, epochs=30, batch_size=1, verbose=2)


        trainPredict = model.predict(trainX)
        testPredict = model.predict(testX)


        trainPredict = scaler.inverse_transform(trainPredict)
        trainY = scaler.inverse_transform([trainY])
```

```
        testPredict = scaler.inverse_transform(testPredict)
        testY = scaler.inverse_transform([testY])

        trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
        print('Train Score: %.10f RMSE' % (trainScore))
        testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
        print('Test Score: %.10f RMSE' % (testScore))

        trainPredictPlot = numpy.empty_like(dataset)
        trainPredictPlot[:, :] = numpy.nan
        trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict


        testPredictPlot = numpy.empty_like(dataset)
        testPredictPlot[:, :] = numpy.nan
        testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-
1, :] = testPredict

        plt.plot(scaler.inverse_transform(dataset))
        plt.plot(trainPredictPlot)
        plt.plot(testPredictPlot)
        plt.savefig('Airline_lookback_'+str(look_back)+'.png')
        plt.show()
        appendFile.write('\nTrain_score: ' + str(trainScore) + ' loopback ' + str(
look_back) + ' epochs: ' + str(30))
        appendFile.write('\nTest_score: ' + str(testScore) + ' loopback ' + str(lo
ok_back) + ' epochs: ' + str(30))
import winsound
frequency = 1400  # Set Frequency To 2500 Hertz
duration = 2000  # Set Duration To 1000 ms == 1 second
winsound.Beep(frequency, duration)
appendFile.close()
```

6.1.4 Cod DateFinaciare look_back variabil:

```
import tensorflow as tf
t_config = tf.ConfigProto()
t_config.gpu_options.allow_growth = True
with tf.Session(config=t_config) as sess:
    import numpy as np
    import numpy
    import matplotlib.pyplot as plt
    from pandas import read_csv
    import math
    from keras.models import Sequential
    from keras.layers import Dense
    from keras.layers import LSTM
    from sklearn.preprocessing import MinMaxScaler
    from sklearn.metrics import mean_squared_error
    import pandas as pd
    from numpy  import array
    import re


    def create_dataset(dataset, look_back=1):
        dataX, dataY = [], []
        for i in range(len(dataset)-look_back-1):
            a = dataset[i:(i+look_back), 0]
            dataX.append(a)
            dataY.append(dataset[i + look_back, 0])
        return numpy.array(dataX), numpy.array(dataY)

    dataframe = read_csv('export.csv', skipinitialspace=True, usecols=[7],sep=';',
skiprows=5,decimal=',')
    dataset = dataframe.values
    dataset = dataset.astype('float32')
```

```
    numpy.random.seed(7)

    for i in range (1,11):

        scaler = MinMaxScaler(feature_range=(0, 1))
        dataset = scaler.fit_transform(dataset)

        train_size = int(len(dataset) * 0.67)
        test_size = len(dataset) - train_size
        train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]

        look_back = i

        trainX, trainY = create_dataset(train, look_back)
        testX, testY = create_dataset(test, look_back)

        trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
        testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))


        model = Sequential()
        model.add(LSTM(20, input_shape=(1, look_back)))
        model.add(Dense(1))
        model.compile(loss='mean_squared_error', optimizer='adam')
        model.fit(trainX, trainY, epochs=30, batch_size=1, verbose=2)


        trainPredict = model.predict(trainX)
        testPredict = model.predict(testX)
        trainPredict = scaler.inverse_transform(trainPredict)
        trainY = scaler.inverse_transform([trainY])
        testPredict = scaler.inverse_transform(testPredict)
        testY = scaler.inverse_transform([testY])
        trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
        print('Train Score: %.10f RMSE' % (trainScore))
        testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
        print('Test Score: %.10f RMSE' % (testScore))

        trainPredictPlot = numpy.empty_like(dataset)
        trainPredictPlot[:, :] = numpy.nan
        trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict

        testPredictPlot = numpy.empty_like(dataset)
        testPredictPlot[:, :] = numpy.nan
        testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-
1, :] = testPredict

        plt.plot(scaler.inverse_transform(dataset))
        plt.plot(trainPredictPlot)
        plt.plot(testPredictPlot)
        plt.savefig('DateFinanciare_lookback_'+str(look_back)+'.png')
        plt.show()


        appendFile.write('\nTrain_score: ' + str(trainScore) + ' loopback ' + str(
look_back) + ' epochs: ' + str(30))
        appendFile.write('\nTest_score: ' + str(testScore) + ' loopback ' + str(lo
ok_back) + ' epochs: ' + str(30))

import winsound
frequency = 1400  # Set Frequency To 2500 Hertz
duration = 2000  # Set Duration To 1000 ms == 1 second
winsound.Beep(frequency, duration)
appendFile.close();
```

6.2 Cod All_in_One

6.2.1 Cod All_in_One Accelerație

```
1. import re
2. import numpy as np
3. import matplotlib.pyplot as plt
4. import matplotlib.patches as mpatches
5. from pylab import *
6. from __future__ import division
7.
8. data_file = open("graphs_Acceleration.txt")
9. block_one = list()
10. block_two = list()
11. block_three = list()
12. found_one = False
13. found_two = False
14. found_three = False
15. def autolabel(rects):
16.     for rect in rects:
17.         height = rect.get_height()
18.         ax.text(rect.get_x() + rect.get_width()/2., 1.05*height,
19.                 '%.2f' % height,ha='center', va='bottom')
20. for line in data_file:
21.     if not (line.startswith('10') or line.startswith('20') or line.startswith('30'
    )):
22.         if found_one:
23.
24.             z= re.findall(r'\S+', line)
25.             for i  in range(0,len(z)):
26.                 block_one.append(z[i]);
27.         if found_two:
28.
29.             z= re.findall(r'\S+', line)
30.             for i  in range(0,len(z)):
31.                 block_two.append(z[i]);
32.         if found_three:
33.
34.             z= re.findall(r'\S+', line)
35.             for i  in range(0,len(z)):
36.                 block_three.append(z[i]);
37.     else:
38.         if line.strip() == "10":
39.             found_one = True
40.             found_two = False
41.             found_three = False
42.
43.         if line.strip() == "20":
44.             found_one = False
45.             found_two = True
46.             found_three = False
47.
48.         if line.strip() == "30":
49.             found_one = False
50.             found_two = False
51.             found_three = True
52. block_one = np.asarray(block_one)
53. block_two = np.asarray(block_two)
54. block_three = np.asarray(block_three)
55. block_one = block_one.astype(float)
56. block_two = block_two.astype(float)
57. block_three = block_three.astype(float)
58. data_file.close()
59. n_groups = 10
60. index = np.arange(n_groups)
61.
62. t = []
63. tone = []
```

```
ttwo = []
fig,ax = plt.subplots(figsize=(100,40))
ylim(ymax=0.3)
ylim(ymin=0)


plt.rc('font', size=40)
plt.rc('axes', labelsize=80)  # controls default text sizes
plt.rc('axes', titlesize=70)     # fontsize of the axes title
  # fontsize of the x and y labels
plt.rc('xtick', labelsize=50)    # fontsize of the tick labels
plt.rc('ytick', labelsize=90)    # fontsize of the tick labels
plt.rc('legend', fontsize=70)    # legend fontsize
plt.rc('figure', titlesize=120)  # fontsize of the figure title

for i in range(0, 10):


    t.append(block_one[i])
    tone.append(block_two[i])
    ttwo.append(block_three[i])
    freq = t
    freqone = tone
    freqtwo = ttwo
    bar_width = 0.3


rects1 = plt.bar(index-bar_width, freq, bar_width, color='r')
rects2 = plt.bar(index, freqone, bar_width, color='g')
rects3 = plt.bar(index+bar_width, freqtwo, bar_width, color='b')
ax.set_ylabel('RMSE TEST Score')
ax.set_title('Acceleration Epochs = 20', y=1.03)
ax.set_xticks(np.add(index,(bar_width/1000))) # set the position of the x ticks
ax.set_xticklabels(('LOOK_BACK = 1', 'LOOK_BACK = 2', 'LOOK_BACK = 3', 'LOOK_BACK = 4', 'LOOK_BACK = 5', 'LOOK_BACK = 6',
                    'LOOK_BACK = 7', 'LOOK_BACK = 8', 'LOOK_BACK = 9', 'LOOK_BACK = 10'))
      autolabel(rects1)
      autolabel(rects2)
      autolabel(rects3)
      T = mpatches.Patch(color='red', label='LSTM CELLS = 10')
      TW = mpatches.Patch(color='green', label='LSTM CELLS = 20')
      TH = mpatches.Patch(color='blue', label='LSTM CELLS = 30')
      plt.legend(handles=[T,TW,TH], loc=1)
      plt.savefig('all_in_one_acceleration.png')

      plt.show()
```

6.2.2 Cod All_in_One Viteză

```
import re
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from pylab import *
from __future__ import division

data_file = open("graph_velocity.txt")
block_one = list()
block_two = list()
block_three = list()
found_one = False
found_two = False
found_three = False
def autolabel(rects):
    for rect in rects:
```

```
        height = rect.get_height()
        ax.text(rect.get_x() + rect.get_width()/2., 1.05*height,
                '%.4f' % height,ha='center', va='bottom')
for line in data_file:
    if not (line.startswith('10') or line.startswith('20') or line.startswith('30'
):
        if found_one:

            z= re.findall(r'\S+', line)
            for i  in range(0,len(z)):
                block_one.append(z[i]);
        if found_two:

            z= re.findall(r'\S+', line)
            for i  in range(0,len(z)):
                block_two.append(z[i]);
        if found_three:

            z= re.findall(r'\S+', line)
            for i  in range(0,len(z)):
                block_three.append(z[i]);
    else:
        if line.strip() == "10":
            found_one = True
            found_two = False
            found_three = False

        if line.strip() == "20":
            found_one = False
            found_two = True
            found_three = False

        if line.strip() == "30":
            found_one = False
            found_two = False
            found_three = True
block_one = np.asarray(block_one)
block_two = np.asarray(block_two)
block_three = np.asarray(block_three)
block_one = block_one.astype(float)
block_two = block_two.astype(float)
block_three = block_three.astype(float)
data_file.close()
n_groups = 10
index = np.arange(n_groups)

t = []
tone = []
ttwo = []
fig,ax = plt.subplots(figsize=(100,40))
ylim(ymax=0.007)
ylim(ymin=0)


plt.rc('font', size=40)
plt.rc('axes', labelsize=80)  # controls default text sizes
plt.rc('axes', titlesize=70)     # fontsize of the axes title
  # fontsize of the x and y labels
plt.rc('xtick', labelsize=50)    # fontsize of the tick labels
plt.rc('ytick', labelsize=90)    # fontsize of the tick labels
plt.rc('legend', fontsize=70)    # legend fontsize
plt.rc('figure', titlesize=120)  # fontsize of the figure title

for i in range(0, 10):


    t.append(block_one[i])
    tone.append(block_two[i])
```

```
    ttwo.append(block_three[i])
    freq = t
    freqone = tone
    freqtwo = ttwo
    bar_width = 0.3


rects1 = plt.bar(index-bar_width, freq, bar_width, color='r')
rects2 = plt.bar(index, freqone, bar_width, color='g')
rects3 = plt.bar(index+bar_width, freqtwo, bar_width, color='b')
ax.set_ylabel('RMSE TEST Score')
ax.set_title('Velocity Epochs = 20', y=1.03)
ax.set_xticks(np.add(index,(bar_width/1000))) # set the position of the x ticks
ax.set_xticklabels(('LOOK_BACK = 1', 'LOOK_BACK = 2', 'LOOK_BACK = 3', 'LOOK_BACK = 4', 'LOOK_BACK = 5', 'LOOK_BACK = 6',
                    'LOOK_BACK = 7', 'LOOK_BACK = 8', 'LOOK_BACK = 9', 'LOOK_BACK = 10'))
      autolabel(rects1)
      autolabel(rects2)
      autolabel(rects3)
      T = mpatches.Patch(color='red', label='LSTM CELLS = 10')
      TW = mpatches.Patch(color='green', label='LSTM CELLS = 20')
      TH = mpatches.Patch(color='blue', label='LSTM CELLS = 30')
      plt.legend(handles=[T,TW,TH], loc=1)
      plt.savefig('lookback.png')

      plt.show()
```

6.2.3 Cod All_in_One Airline

```
#RMSE(look_back)
import re
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from pylab import *
from __future__ import division
rmse = list()
look_back = list()
epochs = 30
cells = 20
def autolabel(rects):
    for rect in rects:
        height = rect.get_height()
        ax.text(rect.get_x() + rect.get_width()/2., 1.05*height,
                '%.5f' % height,ha='center', va='bottom')

data_file = open("RMSE_airline_for_graphs.txt")
for line in data_file:
    #print(line)
    strs = (re.sub('[^\d\.\s+]', '', line))
    rmse.append(strs.split(' ')[1].replace(" ", ""))
    look_back.append(strs.split(' ')[3].replace(" ", ""))
rmse = np.asarray(rmse)
look_back = np.asarray(look_back)
data_file.close()

n_groups = 10
index = np.arange(n_groups)

t = []

fig,ax = plt.subplots(figsize=(100,40))


```

```
36.
37. plt.rc('font', size=80)
38. plt.rc('axes', labelsize=80)  # controls default text sizes
39. plt.rc('axes', titlesize=70)     # fontsize of the axes title
40.   # fontsize of the x and y labels
41. plt.rc('xtick', labelsize=50)    # fontsize of the tick labels
42. plt.rc('ytick', labelsize=90)    # fontsize of the tick labels
43. plt.rc('legend', fontsize=70)    # legend fontsize
44. plt.rc('figure', titlesize=120)  # fontsize of the figure title
45.
46. for i in range(0, 10):
47.
48.
49.     t.append(float(rmse[i]))
50.     freq = t
51.     bar_width = 0.3
52.
53.
54. npa = np.asarray(t, dtype=np.float32)
55. ylim(ymax=np.amax(npa)*110/100)
56. ylim(ymin=0)
57. rects1 = plt.bar(index-bar_width, freq, bar_width, color='r')
58. ax.set_ylabel('RMSE TEST Score')
59. ax.set_title('Airline Epochs = 30', y=1.03)
60. ax.set_xticks(np.add(index,(bar_width/1000))) # set the position of the x ticks
61. ax.set_xticklabels(('LOOK_BACK = 1', 'LOOK_BACK = 2', 'LOOK_BACK = 3', 'LOOK_BACK
    = 4', 'LOOK_BACK = 5', 'LOOK_BACK = 6',
62.                     'LOOK_BACK = 7', 'LOOK_BACK = 8', 'LOOK_BACK = 9', 'LOOK_BACK =
     10'))
63. autolabel(rects1)
64. T = mpatches.Patch(color='red', label='LSTM CELLS = 20')
65. plt.legend(handles=[T], loc=1)
66. plt.savefig('airline_all_in_one.png')
67.
68. plt.show()
```

6.2.4 Cod All_in_One DateFinanciare

```
1.  #RMSE(look_back)
2.  import re
3.  import numpy as np
4.  import matplotlib.pyplot as plt
5.  import matplotlib.patches as mpatches
6.  from pylab import *
7.  from __future__ import division
8.  rmse = list()
9.  look_back = list()
10. epochs = 30
11. cells = 20
12. def autolabel(rects):
13.     for rect in rects:
14.         height = rect.get_height()
15.         ax.text(rect.get_x() + rect.get_width()/2., 1.05*height,
16.                 '%.5f' % height,ha='center', va='bottom')
17.
18. data_file = open("RMSE_DateFinanciare_for_graphs.txt")
19. for line in data_file:
20.     #print(line)
21.     strs = (re.sub('[^\d\.\s+]', '', line))
22.     rmse.append(strs.split(' ')[1].replace(" ", ""))
23.     look_back.append(strs.split(' ')[3].replace(" ", ""))
24. rmse = np.asarray(rmse)
25. look_back = np.asarray(look_back)
26. data_file.close()
27.
28. n_groups = 10
```

```
index = np.arange(n_groups)

t = []

fig,ax = plt.subplots(figsize=(100,40))


plt.rc('font', size=80)
plt.rc('axes', labelsize=80)  # controls default text sizes
plt.rc('axes', titlesize=70)     # fontsize of the axes title
  # fontsize of the x and y labels
plt.rc('xtick', labelsize=60)    # fontsize of the tick labels
plt.rc('ytick', labelsize=90)    # fontsize of the tick labels
plt.rc('legend', fontsize=70)    # legend fontsize
plt.rc('figure', titlesize=120)  # fontsize of the figure title

for i in range(0, 10):


    t.append(float(rmse[i]))
    freq = t
    bar_width = 0.3


npa = np.asarray(t, dtype=np.float32)
ylim(ymax=np.amax(npa)*120/100)
ylim(ymin=0)
rects1 = plt.bar(index-bar_width, freq, bar_width, color='r')
ax.set_ylabel('RMSE TEST Score')
ax.set_title('DateFinanciare Epochs = 30', y=1.03)
ax.set_xticks(np.add(index,(bar_width/1000))) # set the position of the x ticks
ax.set_xticklabels(('LOOK_BACK = 1', 'LOOK_BACK = 2', 'LOOK_BACK = 3', 'LOOK_BACK
  = 4', 'LOOK_BACK = 5', 'LOOK_BACK = 6',
                    'LOOK_BACK = 7', 'LOOK_BACK = 8', 'LOOK_BACK = 9', 'LOOK_BACK =
   10'))
autolabel(rects1)
T = mpatches.Patch(color='red', label='LSTM CELLS = 20')
plt.legend(handles=[T], loc=1)
plt.savefig('DateFinanciare_all_in_one.png')

plt.show()
```

6.3 Cod spargere axe

```
import re
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from pylab import *
from __future__ import division
rmse = list()
look_back = list()
epochs = 30
cells = 20

def autolabel(rects):
    for rect in rects:
        height = rect.get_height()
        ax.text(rect.get_x() + rect.get_width()/2., 1.05*height,
                '%.5f' % height,ha='center', va='bottom')

data_file = open("RMSE_airline_for_graphs.txt")
for line in data_file:
    #print(line)
    strs = (re.sub('[^\d\.\s+]', '', line))
    rmse.append(strs.split(' ')[1].replace(" ", ""))
```

```
    look_back.append(strs.split(' ')[3].replace(" ", ""))
rmse = np.asarray(rmse)
look_back = np.asarray(look_back)
data_file.close()
n_groups = 10
index = np.arange(n_groups)

t = []

#fig,ax = plt.subplots(figsize=(100,40))



plt.rc('font', size=20)
plt.rc('axes', labelsize=20)  # controls default text sizes
plt.rc('axes', titlesize=30)     # fontsize of the axes title
  # fontsize of the x and y labels
plt.rc('xtick', labelsize=10)    # fontsize of the tick labels
plt.rc('ytick', labelsize=30)    # fontsize of the tick labels
plt.rc('legend', fontsize=20)    # legend fontsize
plt.rc('figure', titlesize=120)  # fontsize of the figure title
for i in range(0, 10):


    t.append(float(rmse[i]))
    freq = t
    bar_width = 0.3


pts = np.asarray(t, dtype=np.float32)
#ylim(ymax=np.amax(pts)*110/100)
#ylim(ymin=0)


#pts[[0]] += 10
f, (ax, ax2) = plt.subplots(2, 1,figsize=(17,17), sharex=True)

ax.bar(index-bar_width, freq, bar_width, color='r')
ax2.bar(index-bar_width, freq, bar_width, color='r')
rects1 = plt.bar(index-bar_width, freq, bar_width, color='r')

ax.set_ylim(2, 90)
ax2.set_ylim(0.03, .12)
ax.spines['bottom'].set_visible(False)
ax2.spines['top'].set_visible(False)

ax.xaxis.tick_top()
ax.tick_params(labeltop='off')  # don't put tick labels at the top
ax2.xaxis.tick_bottom()

d = .015  # how big to make the diagonal lines in axes coordinates
# arguments to pass to plot, just so we don't keep repeating them
kwargs = dict(transform=ax.transAxes, color='k', clip_on=False)
ax.plot((-d, +d), (-d, +d), **kwargs)        # top-left diagonal
ax.plot((1 - d, 1 + d), (-d, +d), **kwargs)  # top-right diagonal

kwargs.update(transform=ax2.transAxes)  # switch to the bottom axes
ax2.plot((-d, +d), (1 - d, 1 + d), **kwargs)  # bottom-left diagonal
ax2.plot((1 - d, 1 + d), (1 - d, 1 + d), **kwargs)  # bottom-right diagonal
ax.set_ylabel('RMSE TEST Score')
ax.set_title('Airline Epochs = 30', y=1.03)
ax.set_xticks(np.add(index,(bar_width/1000))) # set the position of the x ticks
ax.set_xticklabels(('LOOK_BACK = 1', 'LOOK_BACK = 2', 'LOOK_BACK = 3', 'LOOK_BACK = 4', 'LOOK_BACK = 5', 'LOOK_BACK = 6',
                    'LOOK_BACK = 7', 'LOOK_BACK = 8', 'LOOK_BACK = 9', 'LOOK_BACK = 10'))
autolabel(rects1)
T = mpatches.Patch(color='red', label='LSTM CELLS = 20')
ax.legend(handles=[T], loc=1)
```

```
plt.savefig('airline_all_in_one.png')
plt.show()
```

6.4 Cod sunet finalizare execuție

```
import winsound
frequency = 1400  # Set Frequency To 2500 Hertz
duration = 2000  # Set Duration To 1000 ms == 1 second
winsound.Beep(frequency, duration)
appendFile.close();
```