Îmbunătățirea Performanței Rețelelor Neuronale Profunde prin Dezvoltarea de Noi Funcții de Activare

Teză destinată obţinerii
titlului ştiinţific de doctor inginer
la
Universitatea Politehnica Timişoara
în domeniul Calculatoare și Tehnologia Informației
de către

Marina Adriana MERCIONI

Conducător ştiinţific:	prof.univ.dr.ing. Ștefan HOLBAN
Referenţi ştiinţifici:	prof.univ.dr. …….. prof.univ.dr.ing. ……. conf.univ.dr.ing. …….

Ziua susţinerii tezei: ……..

Seriile Teze de doctorat ale UPT sunt:

1. Automatică
2. Chimie
3. Energetică
4. Ingineria Chimică
5. Inginerie Civilă
6. Inginerie Electrică
7. Inginerie Electronică și Telecomunicații
8. Inginerie Industrială
9. Inginerie Mecanică
10. Știința Calculatoarelor
11. Știința și Ingineria Materialelor
12. Ingineria sistemelor
13. Inginerie energetică
14. Calculatoare și tehnologia informației
15. Ingineria materialelor
16. Inginerie și Management

Universitatea Politehnica Timişoara a iniţiat seriile de mai sus în scopul diseminării expertizei, cunoştinţelor şi rezultatelor cercetărilor întreprinse în cadrul Şcolii doctorale a universităţii. Seriile conţin, potrivit H.B.Ex.S Nr. 14 / 14.07.2006, tezele de doctorat susţinute în universitate începând cu 1 octombrie 2006.

România, 300159 Timişoara, Bd. Republicii 9,
Tel./fax 0256 403823
e-mail: editura@edipol.upt.ro

Cuvânt înainte

Teza de doctorat a fost elaborată pe parcursul activităţii mele în cadrul Departamentului Calculatoarte și Tehnologia Informației al Universităţii Politehnica Timişoara.

În primul rând, aș dori sa exprim profunde mulțumiri conducătorului meu de doctorat, domnul profesor Ștefan HOLBAN, pentru îndrumarea sa, tot sprijinul, atenția, suportul, timpul, efortul, discuțiile și încurajarea pe care mi le-a oferit pe tot parcursul acestei perioade, care m-au ajutat să iau decizii importante pentru cercetarea mea atât din punct de vedere teoretic cât și din punct de vedere practic. Datorită profesionalismului și a abilităților sale didactice excelente am reușit să dau contur unor idei măreţe care constituie puncte cheie în domeniul Deep Learning, idei pe care le-am acumulat în această teză.

Aș dori, de asemenea, să mulțumesc comisiei de îndrumare constituită din domnii profesori: Călin-Adrian POPA, Dan PESCARU, Cosmin CERNĂZANU-GLĂVAN și Ionel JIAN, pentru că și-au alocat timp din programul lor ocupat să-mi citească și să-mi revizuiască fiecare raport și proiect de cercetare, lucrări care fac parte din teză și de asemenea să-mi furnizeze comentariile și îndrumările lor foarte folositoare.

Aș dori, de asemenea, să îi mulțumesc foarte mult domnului profesor Călin-Adrian POPA, pentru discuțiile foarte utile, pentru tot timpul, efortul și îndrumarea în multiplele prezentări pe care le-am avut în toată această perioadă, care au constituit o sursă de inspirație în conturarea acestei teze.

Aș dori, de asemenea, să îi mulțumesc doamnei profesoare Alina Corina BĂLĂ, pentru încurajarea, susținerea, discuțiile și dragostea pentru mediul academic pe care mi le-a insuflat încă de pe vremea când îi eram studentă la Facultatea de Construcții, din cadrul Universității Politehnica Timișoara, care m-au inspirat pe acest drum.

Aș dori, de asemenea, să aduc mari mulțumiri colegilor mei de lucru: Alexandru VASINCA și Anca CIURTE, care mi-au acordat șansa să intru și să parcurg acest domeniu: Deep Learning, un domeniu vast, cu provocări și un dinamism continuu. Pentru mine a fost o experiență unică, să am șansa să lucrez și să fac cercetare cu privire la Deep Learning și să descopăr în câte domenii are aplicabilitate.

Și pe o notă personală, un cuvânt special de mulțumire se adresează părinților mei, Ana și Dumitru, care m-au sprijinit mereu de-a lungul vieții și au avut încredere deplină în mine și deciziile pe care le-am luat. Mulțumirile mele speciale pentru sora mea geamănă, Ionela, care mereu m-a încurajat și a crezut în mine. Sunt recunoscătoare pentru dragostea și încurajarea întregii familii. Fără încredere și iubire, nu aș fi putut termina această teză.

Sunt deosebit de recunoscătoare prietenului meu Angel TAT, care a fost acolo ori de câte ori am avut nevoie de el. El a găsit întodeauna cuvintele să mă motiveze și să creadă în mine și nu în ultimul rând a aruncat un ochi atent peste redactarea articolelor științifice și a avut răbdarea să asculte prezentările mele de la conferințele științifice inclusiv să mă însoțească la o conferință internațională în Florența, Italia la care am participat cu un articol științific, parte din această teză.

În final aș dori, de asemenea, să le mulțumesc studenților mei, cărora le-am predat un semestru, laboratorul Bazele Inteligenței Artificiale, pentru fructuoasa colaborare și de implicarea de care au dat dovadă și m-au motivat să continui pe acest drum.

Timişoara, decembrie 2020 Marina Adriana MERCIONI

Destinatarii dedicaţiei.

MERCIONI, Marina Adriana

Îmbunătățirea preciziei rețelelor neuronale profunde prin dezvoltarea de noi funcții de activare

Teze de doctorat ale UPT, Seria X, Nr. YY, Editura Politehnica, 200Z, 243 pagini, 176 figuri, 49 tabele.

Cuvinte cheie: *funcție de activare, inteligență artificială, învățare automată, învățare profundă, propagare înapoi, rețele neuronale artificiale*

Rezumat, Rețelele Neuronale Artificiale [297] profunde au fost utilizate în diverse domenii emergente pentru a rezolva probleme complexe din lumea reală cu arhitecturi de Învățare Profundă (*Deep Learning*-DL) [26], fiind într-o permanentă dezvoltare. Pentru a rezolva aceste probleme complexe și a ajunge la performanțe tot mai mari, arhitecturile rețelelor neuronale profunde folosesc funcții de activare, pentru a efectua diferite calcule între straturile ascunse și straturile de ieșire ale rețelei neuronale.

Deci succesul aplicării rețelelor neuronale artificiale, depinde de tipul de arhitectură folosit pentru maparea pe un anumit tip de sarcină, precum și de funcția de activare aplicată rețelei. Atât determinarea arhitecturii cât și a funcției de activare potrivite reprezintă puncte cheie în domeniul Învățării Profunde.

Motivată de importanța funcției de activare care poate avea un impact major și determinant asupra arhitecturii unei rețele neuronale artificiale, propun mai multe funcții de activare care sunt atât cu parametri predefiniți cât și învățabili. Parametri învățabili conferă rețelei o mai mare flexibilitate în calculul de actualizare al ponderilor. Plec de la premisa că de obicei, rețelele neuronale artificiale folosesc funcții de activare neliniare pentru fiecare neuron din rețea.

Metodele și algoritmii pentru implementarea funcțiilor de activare propuse sunt prezentate în cadrul acestei teze. De asemenea, în vederea evaluării cât mai obiective a performanței funcțiilor de activare propuse în rețelele neuronale artificiale profunde, le compar cu o serie largă de funcții de activare extrem de folosite în sarcinele de Învățare Automată folosind arhitecturi state-of-the-art. Seturile de date pe care le-am folosit în cadrul părții experimentale vizează două mari categorii de sarcini: Vederea Artificială (*Computer Vision*-CV) [225] și Procesarea Limbajului Natural (***N****atural* ***L****anguage* ***P****rocessing*-NLP) [298]. Seturile de date din Computer Vision sunt: MNIST [183], Fashion MNIST [279], CIFAR-10, CIFAR-100 [30], Câini și pisici [299] dar și seturi de date care au ca arie de cercetare NLP. Diversitatea seturilor de date, a sarcinilor și arhitecturilor îmi conferă siguranța că am acoperit o parte din principalele obiective ale acestui domeniu atât de vast.

Notaţii, abrevieri, acronime

A
AI - Inteligența artificială (Artificial Intelligence)
ANN - Rețele Neuronale Artificiale (Artificial Neural Networks)
API - Application Programming Interface
APL - Funcție de activare adaptivă pe porțiuni (**A**daptive **P**iecewise **L**inear units)
ARiA2 - Activarea ponderată a Curbei adaptative a lui Richard (Adaptive Richard's Curve weighted Activation)

B
BD - Volum mare de date (*Big Data*)
BP - Algoritmul de propagare înapoi (*Backpropagation*)
BLU – Funcție de activare (Bendable Linear Unit)
BN - Normalizarea lotului (*Batch Normalization*)
BReLU -Funcție de activare (Bipolar Rectified Linear Activation Unit)

C
CEC - "Constant Error Carousel"
CNN - Rețele neuronale de convoluție (Convolutional Neural Networks)
CPU - Unitate centrală de prelucrare (Central Processing Unit)
CReLU – Funcție de activare a unităților liniare rectificate concatenate (Concatenated Rectified Linear Unit)
CU - Unități de bază – (Core unit)
CV - Vederea Artificială (*Computer Vision*)

D
3D – 3 dimensiuni
DA - Mărirea datelor (*Data Augmentation*)
DAS - Conferință internațională (Conference on Development and Application Systems)
DL - Învățare profundă (*Deep Learning*)
Dropout – Tehnică de optimizare a rețelelor neuronale
Dropout fine-tuning - Reglarea fină pentru dropout-ul unităților
dSiLU – Funcția derivată a unității liniare ponderate cu sigmoida

E
EAs - Algoritmii evolutivi (*EvolutionAlgorithms*)
EliSH - Funcția de activare exponențială lineară Squashing (Exponential linear Squashing)
ElliotSig – Funcția de activare ElliotSig (Elliot Sigmoid)
ELU - Unitatea liniară exponențială (Exponential Linear Units)

F
FC - Straturi complet conectate sau dense (*fully connected*)
FER-2013 – Set de date pentru recunoașterea expersiilor faciale (Facial Expression Recognition 2013)

FPGA - Field Programmable Gate Arrays
FReLU - Funcție de activare ReLU flexibilă (flexible rectified linear unit)
FTS - Funcție de activare (Flatten-T Swish)

G
GD – Descendența gradientului (Gradient Descent)
GRU - Gated recurrent unit
GELU - Funcția de activare gausiană (Gaussian Error Linear Unit)
GAP – Eșantionare globală medie (Global Average Pooling)
GLOREP – Conferință internațională (Global and Regional in Environmental Protection)
GPU - Unitatate de procesare grafică (Graphics processing unit)

I
ICCP – Conferință internațională (International Conference on Intelligent Computer Communication and Processing)
ICGSP - Conferință internațională (International Conference on Graphics and Signal Processing)
IJACSA – Jurnal internațional (International Journal of Advanced Computer Science and Aplications)
ILSVRC – Competiție de recunoaștere vizuală pe setul de date de dimensiune mare ImageNet (ImageNet Large Scale Visual Recognition Challenge)
IMC - Indice de masă corporală
IoT - Internet of Things
ISETC – Conferință internațională (International Symposium on Electronics and Telecommunications)
ISRLU - Funcție de activare (inverse square root linear unit)
ISRU – Funcție de activare (inverse square root unit)

L
Logit **-** Intrarea fără normalizare pentru o funcție softmax
LogSig -Funcția de activare Logaritmic-Sigmoid (Logarithmic-Sigmoid)
LReLU -Funcția de activare Leaky ReLU
L-Softmax - Large-margin Softmax
LSUV - Varianța unităților pe straturi secvențiale (Layer-Sequential Unit Variance)
LSTM - Memoria pe termen scurt (*Long Short Term Memory*)

M
MAE – Eroare medie absolută (Mean Absolute Error)
ML - Învățărea Automată (*Machine Learning)*
MLP - Perceptronul multistrat (*Multi-layer perceptron*)
MPP - Procesarea masivă în paralel (Massive Parallel Processing)
MSE - Eroarea medie pătrată (**M**ean **S**quared **E**rror)

N
NAB – Set de date de detecție a anomaliilor în seriile de timp (Numenta Anomaly Benchmark)
Nan - Nu este un număr (not a number)
NiN - Rețea în rețea (network in network)
NLP - Procesarea Limbajului Natural (***N****atural* ***L****anguage* ***P****rocessing*)

O
Oprirea timpurie (*Early stopping*)
One Shot Learning - Idee de învățarea automată
OPLU – Funcție de activare a unității liniare de permutare ortogonală (Orthogonal Permutation Linear Unit Activation Functions)*Overfitting* – Supraspecializarea

P
PELU – Funcția de activare (Parametric Exponential Linear Units)
PReLU - Unitate liniară rectificată parametrică (Parametric Rectifier)
Propagarea datelor de la intrare spre ieșire (*feedforward*)
P-Swish – Funcție de activare (Parametric Swish)

R
ReLU - Unitate liniară rectificată (*Rectified Linear Unit*)
RL - Învățare prin consolidare bazată pe căutare (*reinforcement learning-based search*)
RMSE – Deviația medie rădăcină pătrată (*root mean square error)*
RNN - Rețelele neuronale recurente (Recurent neural networks)
RNS - Rețelele neuronale siameze (*Siamese Networks*)
RadBas – Funcția de activare RadialBasis
ResNet – Arhitectura rețelelor reziduale (Residual Networks)

S
SACI – Conferință internațională (International Symposium on Applied Computational Intelligence and Informatics)
SC – Funcție de activare Soft Clipping
SC Mish – Funcție de activare Soft Clipping Mish
SCL Mish – Funcție de activare Soft Clipping Mish învățabilă (Soft Clipping Mish Learnable)
Sech – Funcție secantă
SGD – Descendența stocastică a gradientului (Stochastic Gradient Descent)
SIMD – Procesor (Single Instruction Multiple-Data)
SNN - Auto-normalizarea rețelelor neuronale
SELU - Unități liniare scalate exponențial (Self-Normalizing Neural Networks)
Swish – Funcție de activare (A self-gated Activation Function)
SiLU - Unitatea liniară ponderată cu sigmoida
SLAF - Funcții de activare auto-învățabile (Self-Learnable Activation Functions)
SLNN - Rețele neuronale echipate cu funcția SLAF (SLAF equipped neural networks)
SNN - Rețea neuronală superficială (Shallow neural network)
Softmax - Funcție exponențială normalizată (*normalized exponential function*)
SReLU – Funcție de activare (S-shaped Rectified Linear Activation Unit)
SQNL - Funcție de activare (Square-Law Non-Linear)
Squashing - Sigmoida
SVM - Mașina vectorului de sprijin (*Support Vector Machine*)

T
Tangenta (tanh)
TanhExp – Funcție de activare (Tanh Exponential)
TanSig – Funcția de activare *tanh*-Sigmoidă

TBSReLU – Funcție de activare (**T**anh-**B**ipolar-**S**igmoid-**Relu**)
TeLU – Funcție de activare *tanh*-elu-ReLU
TL - Învățarea prin Transfer (Transfer Learning)
TReLU – Funcție de activare de prag (Threshold Relu)
TSReLU – Funcție de activare (**T**angent-**S**igmoid-**ReLU**)

V
VAE - Autoencoder variațional (*Variational Autoencoder)*
VGG - Rețele neuronale folosind blocuri (Visual Geometry Group)
VG - Dispariția gradientului – (*vanishing gradient)*
W
WEKA - Waikato Environment for Knowledge Analysis

Lista de tabele

Lista de figuri

Cuprins

1. Introducere

1.1. Motivaţie

Fără funcții de activare, rețeaua neuronală poate să învețe doar sarcini de bază, astfel prin introducerea funcției de activare rețeaua este capabilă să învețe sarcini mai complexe. Înțelegem astfel că funcția de activare este un punct cheie în definirea arhitecturii rețelei neuronale.

Funcția de activare este unul dintre parametrii cei mai importanți pe care trebuie să îi alegem pentru a obține cu succes o performanță mai bună într-o rețea neuronală artificială. Plecând de rolul acesteia, pe care îl joacă în cadrul unei rețele neuronale, focusul nostru s-a îndreptat spre studiul, analiza modului de funcționare, al avantajelor și dezavantajelor funcțiilor de activare pentru a găsi funcția care se mapează cel mai bine pe tipul de sarcină, aducând cele mai bune performanțe.

O proprietate decisivă care determină performanța unei funcții de activare este dată de faptul că este sau nu netedă. [1] Proprietate analizată și de noi în cadrul acestei teze prin comparație cu alte funcții netede precum: funcția de activare tangentă [2], funcția de activare Swish [3] și așa mai departe. Această proprietate conferă funcției să aibă derivate continue, până la un anumit ordin specificat. Acest lucru implică faptul că funcția este diferențiabilă continuu, altfel zis prima derivată există peste tot și este continuă.

De asemenea, datorită faptului că acest domeniu este în continuă dezvoltare, o altă direcție a fost dezvoltarea de noi funcții de activare care să aducă îmbunătățiri la arhitectura rețelelor neuronale artificiale care are ca scop final creșterea performanței acestora și minimizarea funcției de cost.

1.2. Obiectivul cercetării

Această teză are drept obiectiv investigarea utilizării diferitelor funcții de activare care sunt parte integrantă a rețelelor neuronale artificiale în domeniul Învățării Profunde și dezvoltarea de noi funcții de activare care sa aducă îmbunătățiri în timpul antrenării rețelelor. Acest domeniu având mare succes și fiind foarte studiat în ultimii ani datorită disponibilității din punct de vedere hardware, avantaj adus de unitatățile de procesare grafică (*Graphics processing unit – GPU*) precum și de volume tot mai mari de date (*Big Data*). Pentru atingerea acestui obiectiv, am definit următoarele sarcini:

- analiza stadiului curent în domeniul funcțiilor de activare, care s-au dezvoltat de-a lungul timpului, fiind o arie de cercetare continuă, precum și modul de corelare al funcțiilor de activare cu cerințele sarcinii, tipul arhitecturii și tipul de date, în vederea identificării de noi direcții de cercetare în acest domeniu.
- identificarea unor funcții de activare care aduc îmbunătățiri substanțiale și conduc la o convergență rapidă a rețelelor neuronale artificiale. Această sarcină se bazează pe analiza funcțiilor de activare şi selectarea celei mai potrivite funcții pe baza datelor, care constituie de asemenea potenţial de exploatare şi impact.

- implementarea mai multor tipuri de modele şi testarea acestora ţinând cont de diferite metrici ca de exemplu precizie, acurateţe şi funcţie de cost.

1.3. Organizarea tezei

Această teză este organizată în cinci părţi având în total 8 capitole şi mai multe subcapitole (Fig.1.3.1 şi Fig.1.3.2) împărţite astfel:

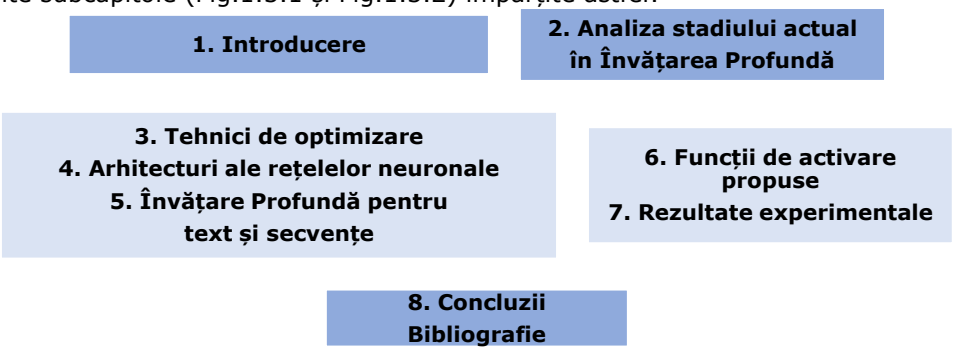


Fig.1.3. 1 Arhitectura lucrării

Prima parte conține capitolul 1 cu scopul de a prezenta motivația și obiectivul cercetării tratate în lucrare. Partea a doua conține analiza stadiului actual în domeniul funcțiilor de activare. Partea a treia, formată din capitolele 3, 4 și 5 corespunde aspectelor teoretice ale lucrării, în care sunt prezentate tehnicile de optimizare folosite în lucrare, diferite arhitecturi ale rețelelor neuronale pe baza cărora am făcut experimente și o scurtă prezentare a Învățării Profunde pentru text și secvențe. Partea a patra este partea principală a lucrării, aceasta conține capitolele 6 și 7 cu funcții de activare propuse și rezultatele experimentale. Ultima parte cuprinde capitolul 8 referitor la concluziile finale privind tratarea problemelor prezentate în lucrare precum și contribuțiile personale. Lucrarea se încheie cu direcții de cercetare viitoare, publicații și bibliografie.

Capitolul 2. Acest capitol se concentrează asupra prezentării state-of-the art și a situației actuale în domeniul funcțiilor de activare, de asemenea prezint una dintre primele arhitecturi, mai exact, arhitectura de tip *Perceptron Multistrat*, un alt topic pe care îl prezint în detaliu sunt funcțiile de activare și în finalul capitolului în mod succint prezint algoritmul de propagare înapoi.

Capitolul 3. Acest capitol prezintă aspecte teoretice privind concepte legate de tehnici de optimizare precum descendența gradientului (*Gradient Descent - GD*), descendența stocastică a gradientului (*Stochastic Gradient Descent - SGD*), evitarea supraspecializării(*overfitting*), oprirea timpurie (*Early stopping*), tehnica de regularizare, tehnica Dropout, care sunt utilizate în evaluarea funcțiilor pe care le propun în cadrul tezei.

Capitolul 4. În cadrul acestui capitol prezint alte aspecte teoretice care cuprind conceptele fundamentale care stau la baza înțelegerii și definirii arhitecturilor rețelelor neuronale profunde, capitol în care prezint în detaliu tipurile de arhitecturi pe care le-am folosit în partea experimentală.

Capitolul 5. În cadrul acestui capitol prezint câteva aspecte teoretice și din domeniul Procesării Limbajului Natural pentru că în cadrul experimentelor îmi voi îndrepta atenția spre o sarcină de tip Analiză Sentiment, folosind rețele neuronale de convoluție (*Convolutional Neural Networks - CNN*).

Capitolul 6. În cadrul acestui capitol care este și cel mai important capitol din teză, descriu în detaliu funcțiile de activare pe care le propun, funcții de activare învățabile și neînvățabile, aplicabile în cadrul rețelelor profunde.

Capitolul 7. În cadrul acestui capitol prezint rezultatele și analiza rețelelor profunde cu funcțiile propuse de activare pe diverse seturi de date și tipuri de sarcini.

Capitolul 8. Acest capitol conține concluziile tezei, posibilitățile de cercetare ulterioare pe marginea acestei teme și publicațiile autorului tezei.

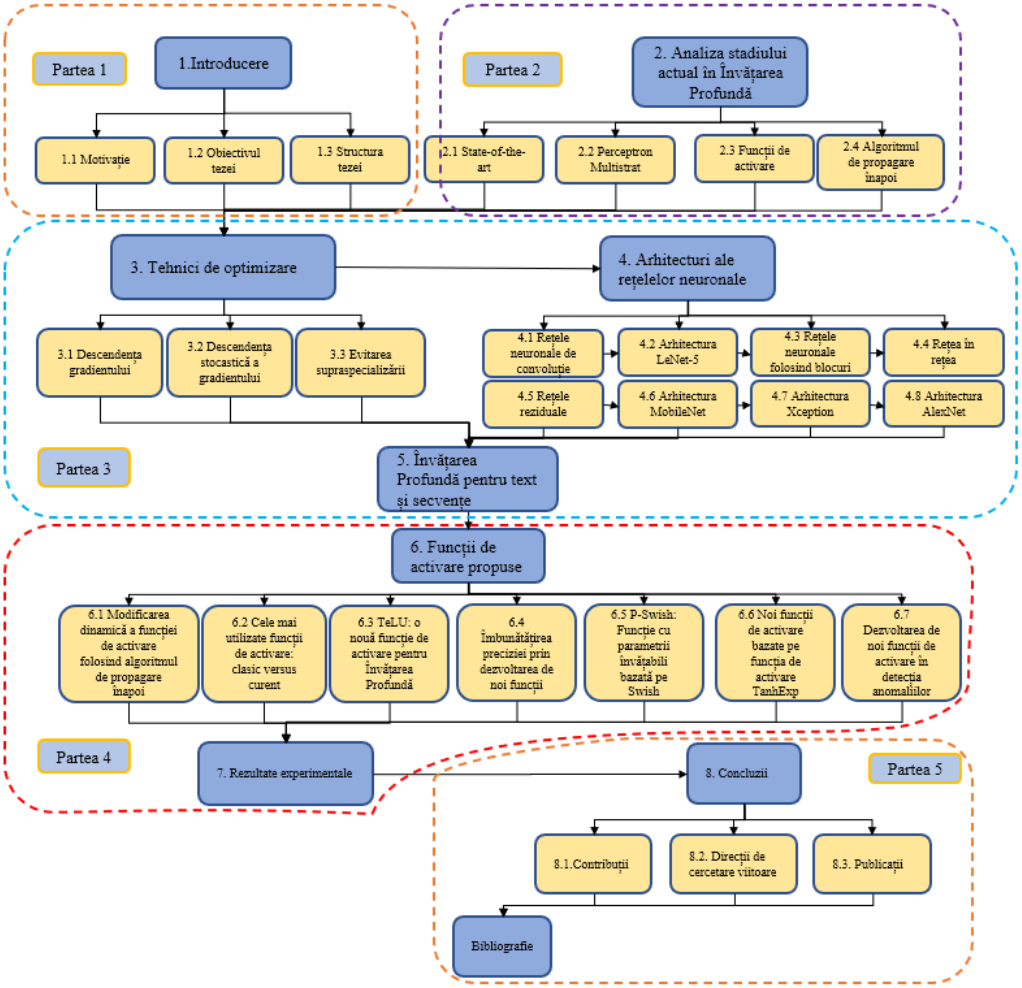


Fig.1.3. 2 Arhitectura lucrării per subcapitole

2. Analiza stadiului actual în Învățarea Profundă

2.1. State of the art

Inteligența artificială (*Artificial Intelligence* - AI) este un domeniu intens studiat în ultimii ani și constituie aspect cheie în dezvoltarea noastră, care oferă calculatoarelor capacitatea să reproducă lumea reală la un grad foarte ridicat, astfel că se obțin rezultate foarte bune. În ultimele două decade s-au înregistrat progrese semnificative în domeniul rețelelor neuronale, apărând domeniul Învățării Profunde (*Deep Learning - DL*), domeniu pe marginea căruia s-au scris un număr mare de articole științifice, care mi-au atras atenția în această direcție. De asemenea se poate remarca faptul că rețelele neuronale se folosesc tot mai des în domenii multivariate și conexe.

Pentru a realiza acest lucru, avem nevoie de o cantitate mare de informații despre tot ce ne înconjoară, informații care trebuie stocate în calculator. Aceste informații le putem modela în așa fel încât să aibă o formă compatibilă cu calculatoarele, pe care acestea o pot folosi pentru a mima la un nivel cât mai ridicat comportamentul uman. Pentru a generaliza acest context mulți cercetători s-au orientat spre algoritmi de învățare pentru a stoca cantitatea de informații într-un timp scurt. S-au făcut multe progrese pentru înțelegerea și optimizarea acestor algoritmi de învățare, dar Inteligența Artificială constituie în continuare o provocare. [4]

Cum am zis mai devreme rețelele neuronale artificiale (*Artificial Neural Networks - ANN*) introduse încă din 1950 sunt un domeniu intens studiat din ultimele două decade. De-a lungul timpului s-au oferit mai multe definiții pentru domeniul Inteligenței Artificiale, prin care acesta se definește ca fiind „efortul de automatizare a sarcinilor intelectuale făcute de om” [5]. Domeniul AI înglobează Învățărea Automată (*Machine Learning* -ML) și Învățarea profundă (*Deep Learning*-DL) ca în Figura 2.1.1.

Rețelele neuronale artificiale sunt sisteme adaptive de procesare a informațiilor neliniare, care combină numeroase unități de procesare cu o serie de caracteristici cum ar fi auto-adaptarea, auto-organizarea și învățarea în timp real. În ciuda dezvoltării cercetării în acest domeniu, s-au întâmpinat probleme precum: selectarea structurii și a parametrilor rețelelor, selectarea valorilor inițiale, convergența algoritmilor de învățare. [6]

S-a constatat că performanța rețelelor neuronale artificiale depinde de numărul de neuroni. Astfel că prea puțini neuroni pot rezulta o aproximare mică, în timp ce un număr prea mare de neuroni pot crea probleme de dimensionare. Începând cu anii '90, algoritmii evolutivi (*EvolutionAlgorithms* - EAs) s-au dezvoltat cu succes în vederea optimizării arhitecturii și a parametrilor rețelei. [7]

În 2006, s-a dezvoltat algoritmul Descendența Gradientului (*Gradient Descent*), dar care nu a dat rezultate bune fără funcții de activare. Funcțiile de activare reprezintă aria de dezvoltare continuă a Inteligenței Artificiale.

Învățarea Profundă este un domeniu care are un impact major în prezent. Practic, este o subdiviziune a domeniului Învățării Automate, care este, de asemenea, un alt topic important. Învățarea Profundă folosește o multitudine de algoritmi pentru a recunoaște obiectele și a înțelege comportamentul uman în diferite domenii [8].

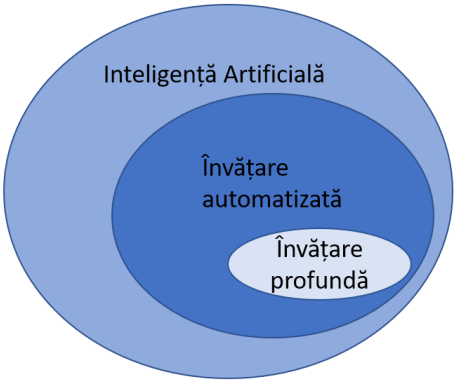


Fig.2.1 1 Inteligență artificială, Învățare automatizată și Învățare profundă -Diagrama Venn [9]

Istoria Învățării Profunde a început încă din 1943, când s-a creat un model bazat pe rețelele neuronale biologice ale creierului uman. S-a folosit o combinație de algoritmi și fundamente matematice, numită „logica pragului" pentru a reproduce cât mai exact procesul gândirii umane. Din acel moment, Învățarea Profundă a evoluat constant, au fost doar două pauze semnificative în dezvoltarea acestui domeniu, menționate în literatura de specialitate ca "iernile urâte" ale Inteligenței Artificiale.

În 1960 se definesc elementele de bază ale unui model continuu de propagare înapoi, derivând algoritmul de propagare înapoi (*Backpropagation*-BP), [10] topic studiat ulterior [11] [12].

În 1965 au continuat studiile prin dezvoltarea de algoritmi în Învățarea Profundă prin folosirea de modele polinomiale ale funcțiilor de activare. [13]

Algoritmul propagării înapoi, care are la bază utilizarea erorilor în vederea formării modelelor de Învățare Profundă, s-a dezoltat semnificativ în 1970 când s-a scris un cod FORTRAN pentru propagarea înapoi. Cu toate acestea, conceptul nu a fost aplicat la rețele neuronale, până în 1985, când s-a arătat că acest algoritm aduce o distribuție valoroasă a reprezentărilor. Ca urmare a acestei idei, în 1989 s-au adus primele experimente practice care validau această teorie. [14]

Primele rețele neuronale de convoluție au fost folosite în 1980, când s-a dezvoltat o rețea neuronală artificială, cunoscută sub numele de *Neocognitron* [15], care avea un design ierarhic, de tip multistrat. Acest design a permis calculatorului să învețe să recunoască tipare vizuale. Rețelele păreau mai degrabă versiuni moderne, dar care totuși au fost antrenate cu o strategie de consolidare cu activări recurente în mai multe straturi, tehnică care a s-a dovedit a fi un punct forte. [16]

Perioada 1985-1990 a constituit un al doilea declin în domeniul Învățării Profunde, declin datorat mai multor teorii „mult prea optimistice în vederea potențialului" pe care îl avea pe atunci Inteligența Artificială. În ciuda acestui declin, în 1995 s-a dezvoltat mașina vectorului de sprijin (*Support Vector Machine* – SVM). Cercetările au continuat astfel că în 1997 apare memoria pe termen scurt (*Long Short Term Memory* - LSTM) pentru rețelele neuronale recurente (*Recurent Neural Networks* - *RNN*).

În ciuda popularității sale, Învățarea Profundă nu este încă bine aplicată în diferite domenii datorită carenței de cunoaștere în acest domeniu care este relativ nou. Dar, pe de altă parte, ne oferă o mulțime de algoritmi pentru a identifica aceste păreri greșite în vederea atingerii unei precizii mai bune. De exemplu, putem detecta

anomalii în situațiile financiare [17] [18] folosind algoritmii de ML, detecția anomaliilor cu date industriale din seriile de timp [19].

Domeniul Vederii Artificiale (*Computer Vision - CV*) apare începând cu anii 1960, prin care cercetătorii au încercat să reproducă modul de viziune umană, pentru a înțelege ce văd calculatoarele prin procesarea de imagini.[20] În anul 1989 s-a aplicat algoritmul de propagare înapoi pe o rețea neuronală de convoluție. În acest fel, a fost construită prima rețea neuronală de convoluție modernă, numită *LeNet-5* [21].

Începând cu 1999, cercetătorii s-au oprit să reconstruiască obiectele prin intermediul modelelor 3D și și-au îndreptat atenția spre recunoașterea caracteristicilor unui obiect. [22] În 2001 a apărut prima soluție capabilă să detecteze fața umană în timp real. [23]

În 2000 ia amploare conceptul de date masive (*Big Data*) care permite analiza, extragerea sistematică de informații sau tratarea seturilor de date care sunt prea mari sau prea complexe pentru metodele software tradiționale de prelucrare a datelor.

În perioada 2006-2012, competițiile anuale au început să aducă rezultate spectaculoase prin aplicarea de diferite metode de recunoaștere a obiectelor din imagine. Cu pași repezi, în 2010 deja putem vedea cât de mult a evoluat tehnologia și algoritmii prin apariția domeniului Învățării Profunde.

Învățarea Profundă este o tehnică de Învățare Automată prin care calculatoarele învață să facă ceea ce pentru oameni le este normal, mai exact, învățarea prin exemplu. Aceasta a câștigat mult interes în ultima perioadă și datorită faptului că a condus la rezultate care până acum erau imposibile.

În Învățarea Profundă, un model învață să efectueze sarcini de clasificare direct din imagini, text sau sunet. Acesta este capabil să învețe fără a fi nevoie de intervenția umană, bazându-se pe date care pot fi nestructurate și fără etichete. Modelele de Învățare Profundă pot obține precizie state-of-the-art, depășind uneori chiar și performanțele umane. Pentru acest tip de învățare este nevoie de un volum mare de date etichetate, pe care modelele sunt antrenate folosind arhitecturi de rețele neuronale care conțin multe straturi. Arhitecturile din Învățarea Profundă oferă o flexibilitate mare ceea le conferă rețelelor proprietatea să poată învăța direct din datele brute și totodată pot ajunge la precizii mari de predicție atunci când sunt furnizate rețelei neuronale mai multe date.

Învățarea Profundă poate fi privită ca o "operație de distilare a informației, se aplică filtre succesive informației obținându-se în final data "purificată" ". [5]

În timp ce Învățarea Profundă a prins contur teoretic pentru prima dată în anii 1980, există două motive principale pentru care recent a devenit utilă din punct de vedere practic și anume: nevoia de cantități mari de date etichetate și o mare putere de calcul. La această problemă vin ca soluție unitățile de procesare grafică (*Graphics Processing Units* - GPU). De asemenea, o atenție deosebită este acordată domeniului Vederii Artificiale care a venit ca o soluție pentru mai multe probleme, cum ar fi detectarea obiectelor [24], recunoașterea feței [25], clasificarea imaginii folosind seturi de date mari - ImageNet [26], care a fost conceput pentru a îmbunătăți performanțele recunoașterii obiectelor, segmentarea părului [27], urmărirea obiectelor în imagini [28], utilizarea rețelelor de convoluție pe dispozitivele mobile [29]. De asemenea, acest domeniu a atins un punct culminant prin apariția arhitecturii AlexNet care a câștigat competiția pe setul de date ImageNet. [30]

Având atât de multe date, avem nevoie și de un model puternic pentru a obține o performanță bună prin alegerea corespunzătoare a funcției de activare. Principalele arhitecturi de învățare profundă utilizate pentru prelucrarea imaginilor sunt Rețeaua Neuronală Convoluțională (*Convolutional Neural Network*-CNN) sau cadre CNN precum VGG [31], Inception [32], rețele reziduale (*Residual Networks - ResNet*) [33].

Cercetarea continuă în Învățarea Profundă prin aplicații precum recunoașterea vocii [34] prin procesarea limbajului natural (NLP) folosind tehnica de memorie de lungă-scurtă durată (*Long Short Term Memory - LSTM*) [35] sau interpretarea textului. [36]

Pentru toate sarcinile menționate mai sus, rețeaua încearcă să înțeleagă și să clasifice mulțimea de informații în informații bune sau rele. Acest lucru joacă un rol important, deoarece nu toate informațiile sunt utile pentru sarcinile noastre. Unele dintre aceste informații reprezintă zgomot care va avea impact la finalul antrenării asupra performanței modelului. Deci, funcția de activare ajută modelul să distileze informația, astfel modelul va fi capabil să păstreze informațiile bune și să suprime informațiile irelevante.

Deși atât de mulți algoritmi au fost dezvoltați de-a lungul timpului, nici funcțiile de activare nu au fost ignorate. Pornind de la rolul funcției de activare, adică de a filtra informațiile, dorim să avem funcții cu proprietăți la fel de specifice, întrucât se bazează pe actualizarea ponderilor din rețeaua neuronală, acest proces fiind cunoscut sub denumirea de învățarea rețelei neuronale. Pornind de la această considerație, s-au dezvoltat mai multe funcții de activare care ne permit să le folosim în diferite contexte și care ajută rețelele neuronale să atingă mai rapid convergența sau le oferă capacitatea de a utiliza mai puține straturi în definirea arhitecturii lor. Cu mai puține straturi în arhitectură, avem mai puțini parametri în rețeaua noastră deci reprezintă o bună modalitate de optimizare a rețelei. Scopul principal al funcției de activare este de a introduce neliniaritate în rezultatul unui neuron. De asemenea, funcția de activare cunoscută și sub denumirea de *funcție de transfer* poate decide dacă un neuron trebuie activat sau nu printr-o sumă ponderată la care se adaugă bias. O rețea neuronală fără funcții de activare este doar un model de regresie liniară. Dar funcția de activare oferă rețelei capacitatea de a învăța sarcini mai complicate.

Un alt aspect important pe care funcția de activare și determinarea inițializării ponderilor a fost dat de spațiul în care va lua valori algoritmul de optimizare. Acest algoritm vine ca o soluție pentru crearea rețelei neuronale care constă într-o problemă de optimizare non-convexă. [37]

Cum am spus mai devreme, funcțiile de activare sunt utilizate de rețelele neuronale artificiale, deci reprezintă un standard în dezvoltarea arhitecturii rețelei neuronale artificiale, în funcție de scopul acestora. [38] [39]

Selectarea unei funcții de activare reprezintă un subiect important, deoarece poate afecta modul în care se schimbă datele de intrare. [40]

2.2. Arhitectura Perceptron Multistrat

Rețelele Neuronale Artificiale sunt definite de 3 elemente: "modelul pentru elementul de procesare (neuronul), arhitectura interconexiunilor dintre neuroni precum și modalitatea de ajustare a interconexiunilor dintre neuroni (algoritmul de învățare)".

Un neuron artificial se reprezintă „într-un mod simplificat, prin intermediul unei funcţii matematice", acesta este corelat cu un neuron dintr-o reţea neuronală biologică. Totalitatea neuronilor biologici conectați constituie creierul uman. Neuronul artificial este unitatea elementară într-o reţea neuronală artificială. Ca și componență, la fel ca și în cazul neuronului biologic, care are dendrite şi axoni, neuronul artificial are o structură arborescentă simplă, cu noduri de intrare şi un nod de ieşire conectat la toate nodurile de intrare. [41]

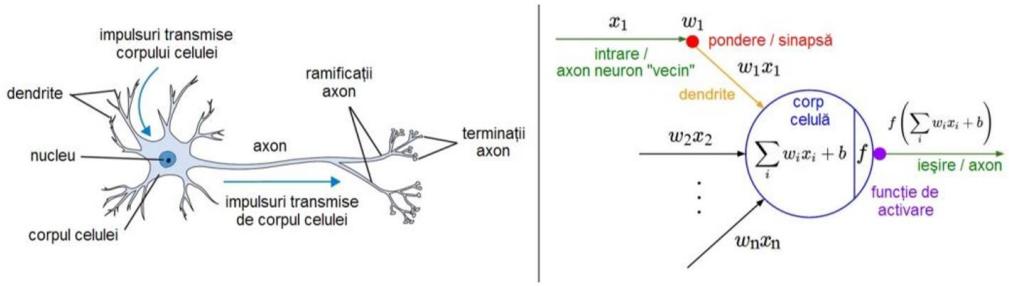


Fig.2.2 1 Reprezentarea neuronului biologic versus versiunea simplificată a neuronlui artificial [42]

Perceptronul este o reţea neuronală artificială simplificată care are în componență un singur neuron artificial. Modelul perceptronului a fost introdus în 1958 [43], bazându-se și pe cercetările anterioare făcute în 1943 [44] vizând un model matematic pentru funcţionarea neuronilor din creierul uman.

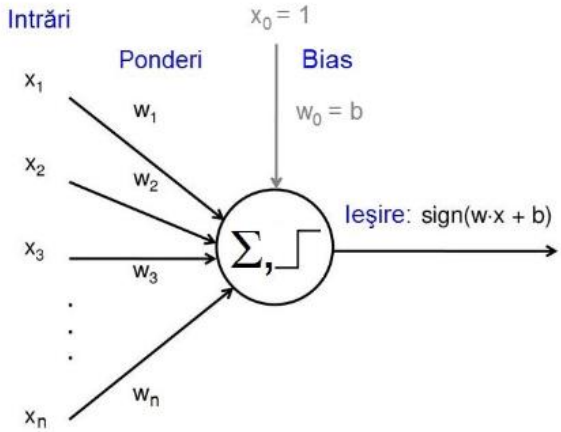


Fig.2.2. 1 Perceptronul [42]

Unde x_i reprezintă intrările, w_i sunt ponderile iar b reprezintă *bias* care este valoarea de prag de activare a perceptronului şi poate fi considerat ca o (*n+1*) intrare, suplimentară, de valoare $x_0 = 1$ şi ponderea $w_0 = b$ constante şi permite translatarea funcţiei de activare la stânga sau la dreapta, după cum este nevoie în timpul antrenării rețelei neuronale. [42]
Procesul de învățare al neuronilor constă în modificarea sau actualizarea ponderilor în timpul antrenamentului cu algoritmul de propagare înapoi. Perceptronul era format dintr-un neuron de prag binar, cunoscut și sub denumirea *de unități de prag binar* și regula de învățare a perceptronului, fiind definit astfel:

$$z = b + \sum_i w_i x_i \tag{2.2.1}$$

$$y = \begin{cases} 1, & dacă\ z \geq 0 \\ 0, & altfel \end{cases} \tag{2.2.2}$$

În ecuația 2.2.1 *z* este ieșirea, b -bias, x_i reprezintă intrările, iar w_i sunt ponderile. Ecuația 2.2.2 definește decizia, făcută de obicei prin neliniaritate, în acest caz este funcția treaptă. [45]

Algoritmii de Învățare Profundă sunt tehnici de învățare pe mai multe nivele, care permit învățarea de funcții complexe. Acest tip de învățare s-a dezvoltat ca urmare a învățării convenționale, mai ales a limitării procesării datelor. [46]

Dacă perceptronul era o rețea simplă, perceptronul multistrat reprezintă o rețea de perceptroni conectați succesiv pe straturi (*Multi-layer perceptron – MLP*). Din punct de vedere constructiv, acest tip de rețea este alcătuită dintr-un strat de intrare, unul sau mai multe aşa numite straturi ascunse şi un strat de ieşire, conectate secvenţial între ele. [46] De obicei stratul de ieșire prezintă rezultatele rețelei care sunt adesea controlate de funcția de activare, în special pentru a efectua clasificări sau predicții, cu probabilități asociate. Poziția unei funcții de activare într-o structură de rețea depinde de rolul său în rețea, astfel că funcția de activare este plasată după straturile ascunse care convertesc mapările liniare învățate în forme neliniare pentru propagare în timp ce se află în stratul de ieșire care efectuează predicții. [9]

Atunci când numărul de straturi ascunse este mai mare de 2, aceste rețele se consideră rețele dense, caz în care care este vorba de o rețea profundă. În situația unui singur strat ascuns avem o rețea neuronală superficială. Straturile cu toți neuronii conectați cu toți neuronii din stratul anterior se numesc *straturi dense* sau *complet conectate*.

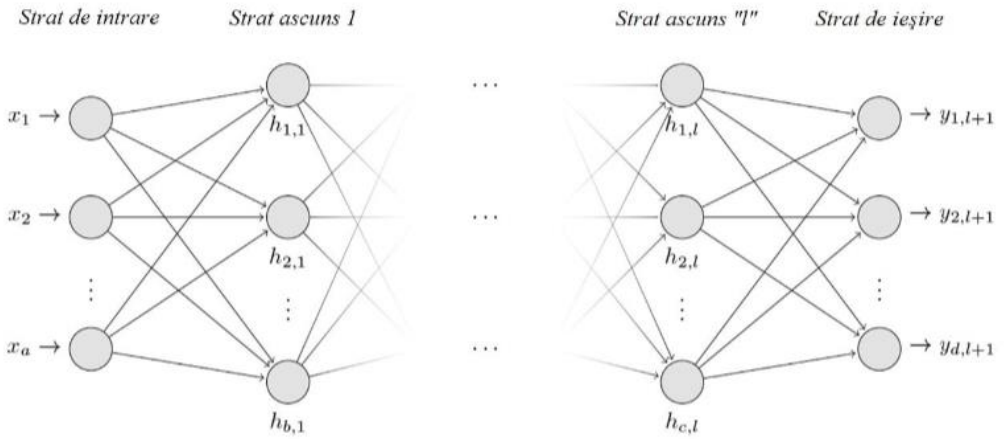


Fig.2.2. 2 Perceptronul multistrat [42]

Ca și arhitectură a interconexiunilor dintre neuroni vom studia arhitectura rețelelor neuronale de tip Perceptron Multistrat (*Multilayer Perceptron-MLP*). Modul de interconexiune a neuronilor între ei determină modul de calcul al ponderilor și funcționalitatea rețelei. Rețelele neuronale de tip perceptron sunt cele mai întâlnite datorită capacității acestora de generalizare prin modul de operare cu date diferite din etapa de antrenament. Acest tip de rețele sunt rețelele în care propagarea datelor se face de la intrare spre ieșire, acest lucru implică faptul că un neuron poate avea ca întrări doar ieșirile neuronilor de pe straturile precedente. Voi începe analiza mea, de la o rețea neuronală cu mai multe straturi, Perceptronul Multistrat cu propagare înainte. Acest lucru înseamnă unul sau mai multe straturi între intrare și ieșire. Acest tip de straturi (straturi ascunse) conțin neuroni care nu sunt conectați în mod direct cu întrările sau ieșirile rețelei. Acest tip de rețele a fost foarte folosit datorită capacității lor de rezolvare a sarcinilor de clasificare prin aproximarea universală. [47]

Rețelele neuronale artificiale se bazează pe unități computaționale care seamănă cu proprietățile de bază ale procesării informațiilor neuronilor biologici într-o manieră abstractă și simplificată. Neuronul este tratat ca o cutie neagră, astfel că extinderea spațială și dinamica temporală prezente în neuronii biologici sunt de cele mai multe ori neglijate. Chiar dacă neuronii artificiali sunt simplificați, ei pot prezenta o varietate de relații intrare-ieșire pe baza funcțiilor de transfer pe care le aplicăm. [48]

S-a arătat de asemenea, că modularitatea este o caracteristică cheie de construcție a creierului uman care a inspirat mult studiul în cercetarea Rețelelor Neuronale Artificiale. Aspectele structurale, de învățare și funcționale ale rețelelor modulare au fost studiate pe larg și multe sisteme cu grade diferite de complexitate au fost propuse. Avantajele pentru structurile non-modulare includ un timp redus de antrenament, o scalabilitate mai bună cu creșterea complexității problemelor și înțelegerea mai ușoară a sarcinilor îndeplinite de diferite părți ale sistemului global. [49]

Un model de tip Perceptron Multistrat poate fi utilizat pentru a efectua sarcini de clasificare. S-a arătat că cel mult un strat ascuns este necesar pentru aproximarea funcțiilor. [50] [51] În orice caz, testele experimentale au arătat că o rețea cu două straturi ascunse se va antrena mai repede decât o rețea cu un strat ascuns în cazul

unor probleme de clasificare. Numărul de straturi utilizate depinde de tipul problemei, la fel și numărul de noduri din fiecare strat ascuns. [52] Pentru actualizarea ponderilor au apărut mai multe reguli, dar cea mai populară rămâne însă propagarea înapoi. [53] [54] [55] Tehnica se bazează pe calculul gradientului descendent în spațiul de ponderi pe o suprafață de eroare creată prin suma erorii pătrate la fiecare nod de ieșire pe întregul set de antrenare. O altă regulă pentru ajustarea ponderilor este algoritmul de filtrare numit *Kalman*. [56] S-a observat că perceptronul cu mai multe straturi, atunci când lucrează în modul de asociere automată, este potrivit pentru a efectua compresia de date sau reducerea dimensionalității. [57]

O rețea multistrat evaluează compozițiile funcțiilor calculate pe noduri individuale. [58]

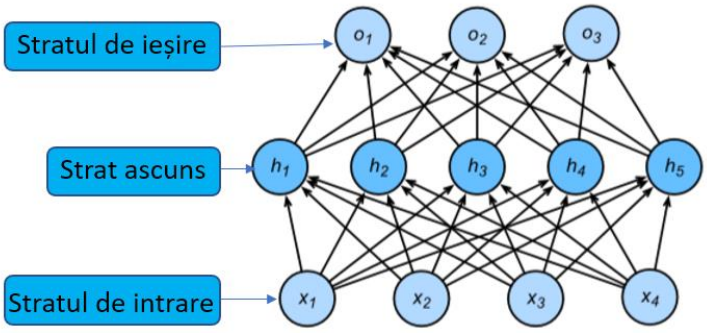


Fig.2.2. 3 Rețea neuronală de tip propagare înainte

Liniaritatea în rețelele Perceptron Multistrat și nu numai, implică o prezumție mai slabă a monotoniei: că orice creștere a caracteristicii noastre trebuie să provoace întotdeauna o creștere a valorii de ieșire a modelului, dacă ponderea corespunzătoare este pozitivă, și invers în caz contrar. Exemplu clar în acest sens îl constituie predicția dacă un împrumut bancar va fi returnat sau nu. [59] Însă această proprietate poate fi ușor contracarată, de exemplu, vrem să prezicem probabilitatea ca moartea să fie cauzată de procentul de grăsime. Pentru indivizii cu un indice de masă corporală (IMC) mai mare de 40, se incadrează în sfera obozitate morbidă care crește riscul decesului. Totuși indivizilor cu un IMC mai mic de 40, scade acest risc. Pentru a calcula acest risc, putem calcula distanța între această valoare de prag și valoarea curentă a individului.

S-au arătat o serie de motive pentru care utilizarea funcțiilor care nu sunt monotone ca funcții de activare pot duce la o îmbunătățire semnificativă a performanței unei rețele neuronale. S-a arătat că o rețea cu mai multe straturi care utilizează funcții sinusoidale împreună cu o alegere adecvată a inițializării parametrilor învață mult mai repede decât una care conține funcții sigmoide. [60]

În Figura 2.2.3 avem o rețea Perceptron Multistrat cu un strat ascuns, model care are 4 intrări, 3 ieșiri și stratul ascuns care conține 5 neuroni (unități). Fiind complet conectate straturile, intrările influențează fiecare neuron din stratul ascuns, care la rândul lui influențează ieșirile.Totodată, modelele de tip Perceptron Multistrat sunt cunoscute și ca *aproximatori universali* [61], care implică faptul că rețelele neuronale pot avea o mare varietate de funcții interesante atunci când au ponderi corespunzătoare. Pentru a beneficia de potențialul acestor tipuri de modele, trebuie să adăugăm un element cheie și anume, ***o funcție de activare*** neliniară care să fie

aplicată fiecărui neuron din straturile ascunse. În prezent, cea mai populară și folosită funcție pentru a introduce non-liniaritate în rețea este dată de funcția de activare ReLU.

2.3. Funcții de activare

Funcția de activare este responsabilă pentru decizia dacă un neuron ar trebui sau nu să fie activat prin calculul unei sume ponderate căreia i se adaugă bias. Funcția de activare introduce neliniariatate în rețea, permițând astfel învățarea de sarcini complexe. Pentru că funcția de activare constituie un element fundamental în Învățarea Profundă, în cadrul acestui subcapitol voi prezenta o mare parte dintre aceste funcții.

Funcția de activare manipulează datele prezentate prin procesarea unor gradienți, de obicei descendența gradientului (*Gradient Descent*-GD) și apoi produce o ieșire pentru rețeaua neuronală, care conține parametri din date. Aceste funcții de activare sunt deseori menționate ca o funcție de transfer în literatura de specialitate. [9]

Funcțiile de activare includ convertirea semnalelor de intrare liniare și a modelelor în semnale de ieșire neliniară, care ajută învățarea polinoamelor de ordin superior pentru rețele profunde. O proprietate a funcțiilor de activare neliniare este diferențierea. [2]

Alegerea funcțiilor de transfer poate influența puternic complexitatea și performanța rețelelor neuronale. Deși funcțiile de transfer de tip sigmoidă sunt cele mai frecvente, nu există un motiv pentru care modelele bazate pe astfel de funcții ar trebui să ofere întotdeauna limite optime de decizie. [62]

Interesul pentru analiza Rețelelor Neuronale Artificiale prin folosirea unei funcții de activare a crescut vertiginos în ultimii ani, devenind astfel un domeniu de cercetare foarte studiat, drept dovadă articolele existente în această direcție.
Motiv pentru care în această teză doresc să subliniez importanța funcției de activare în analiza mai multe seturi de date pentru diferite sarcini, pentru a vedea în ce mod impactează funcția de activare procesul de antrenare.

S-a evaluat impactul alegerii funcției de activare, în ideea că aceasta ar putea afecta saturația și procedura de inițializare, deoarece pre-antrenarea nesupervizată este o formă particulară de inițializare și poate avea un impact determinant. S-a observat că rețelele neuronale cu sigmoidă dar inițializate din pre-antrenarea nesupervizată nu suferă de saturație. [63]

Această secțiune evidențiază diferitele tipuri de funcții de activare și evoluția acestora pe parcursul anilor. Cercetarea și aplicabilitatea funcțiilor de activare în arhitecturi profunde, utilizate în diferite aplicații au fost un domeniu de cercetare continuu și intens până în prezent, dar asiduu datorită minusurilor pe care le avea fiecare funcție în parte. În funcțiile de activare se stabilesc limite pentru ieșirile neuronilor. Rețelele neuronale pot folosi mai multe funcții de activare. Cele mai frecvente funcții de activare clasice și curente sunt tratate în această teză. Selectarea unei funcții de activare reprezintă un aspect important din moment ce poate afecta modul în care sunt modificate datele de intrare.

Rezultatele cercetării de ultimă oră sunt subliniate în continuare, totuși, în această teză funcțiile de activare nu sunt prezentate cronologic.

Funcția de activare **sigmoidă** este uneori întâlnită ca funcția logistică sau funcția de *squashing* în literatura de specialitate. [64] Din punct de vedere al evoluției, această funcție s-a dezvoltat din funcția Heaveside. Sigmoida este o funcție neliniară utilizată în majoritatea rețelelor neuronale de tip propagare înainte. Este o funcție reală, diferențiabilă, definită pentru valorile de intrare reale, cu derivate pozitive peste tot și este netedă. [65] Se recomandă atunci când ieșirea funcției de activare ia o valoare între 0 și 1. Este utilă de exemplu, în cazul clasificării folosind rețele neuronale. Ea a fost utilizată la scară largă în Învățarea Automată, mai ales pentru regresia logistică și alte câteva implementări de rețele neuronale tradiționale. Oricum, această funcție nu este singura alegere ca funcție de activare, de asemenea având și dezavantaje. De obicei, această funcție apare pe straturile de ieșire ale arhitecturilor Învățării Profunde și este utilizată pentru predicția bazată pe probabilități a ieșirii și a fost aplicată cu succes în probleme de clasificare binară, modelarea sarcinilor de regresie logistică, dar și alte tipuri de arhitecturi ale rețelelor neuronale. Principalele avantaje ale funcțiilor sigmoide sunt simplitatea de aplicare și utilizarea în rețelele de adâncime mică. [66] Mai mult de atât, se sugerează ca funcția de activare sigmoidă să fie evitată atunci când inițializăm rețeaua neuronală cu ponderi mici aleatorii. [63] Cu toate acestea, funcția sigmoidă suferă dezavantaje majore care includ gradienți abrupți în timpul propagării înapoi de la straturile ascunse mai profunde către straturile de intrare, probleme de saturație a gradienților, convergența lentă și ieșirea centrată non-zero care implică astfel actualizările gradientului de propagare în diferite direcții. Au fost propuse alte forme de funcții de activare, de exemplu funcția tangentă hiperbolică (*hyperbolic tangent - tanh*), care remediază unele dintre aceste dezavantaje suferite de funcția de activare sigmoidă. [9]
Funcția sigmoidă este dată de relația:

$$f(x) = \frac{1}{1+e^{-x}} \tag{2.3.1}$$

Pornind de la această formulă, putem calcula funcția derivată astfel:

$$f'(x) = f(x)(1 - f(x)) \tag{2.3.2}$$

În ciuda faptului că derivata se calculează ușor, pentru construirea de modele acest lucru înseamnă că funcția sigmoidă, de fapt, forțează modelul să "piardă" din date.

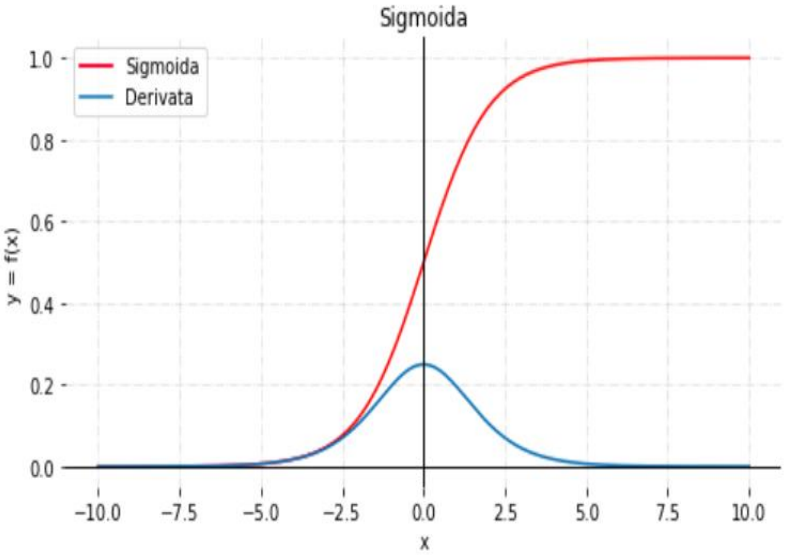


Fig.2.3 1 Funcția de activare sigmoidă

Sigmoida este des apelată pentru neliniaritatea ei. De asemenea, multe funcții de activare sunt neliniare, sau combinația dintre liniar și neliniar. Dezavantajul folosirii acestei funcții este legat de folosirea ei în algoritmul de propagare înapoi. Deoarece în cadrul acestui algoritm calculăm gradienți pentru fiecare pondere, mai exact facem mici actualizări pentru fiecare pondere, tocmai în vederea optimizării valorilor activărilor ieșirilor prin întreaga rețea, care va conduce implicit la un rezultat bun de pe stratul de ieșire.

De-a lungul algoritmului de propagare înapoi trebuie să calculăm o metrică care ne indică cât de mult ponderile impactează funcția de cost prin găsirea derivatelor parțiale ale acestei funcții prin respectarea fiecărei ponderi.

Atunci când avem o intrare mare, pozitivă sau negativă, pentru valoarea intrării din funcția sigmoidă se va returna o valoare egală cu 0.

Atunci când ponderile au valori mari, vom avea cazul în care ne confruntăm cu o rețea care nu ajustează ponderile, acest fapt reprezintă o problemă. Acest lucru conduce la ineficiența algoritmului rețelei. Calculul derivatelor parțiale cu respectarea ponderii, pe care le punem într-un vector cu gredienți pentru a actualiza rețeaua. Dar în cazul în care aceste valori ale gradienților sunt apropriate de 0, nu mai putem face această actualizare. Aici intervine cunoscuta problemă de dispariție a gradientului (*vanishing gradient*), caz în care ponderile și bias-ul primesc valori mici. În mod analog, dar la polul opus se află problema exploziei gradienților (*exploding gradient*) dacă folosim arhitecturi de rețele recurente precum LSTM [67] sau GRU (***G**ated **R**ecurrent **U**nit*) [68]. GRU este o variantă de LSTM introdusă de K. Cho. GRU reține proprietățile dispariției gradientului a LSTM-ului, dar fiind mai simplă și mai rapidă decât LSTM.

Unul din avantajele funcției sigmoide constă în faptul că este ușor de înțeles și de aplicat, ușor de antrenat pe un set de date de dimensiune mică. [69]

Dezavantajele funcției sigmoide

- Gradientul pentru intrările care sunt departe de origine este aproape zero, astfel încât învățarea gradienților este lentă pentru neuronii saturați.
- Când este utilizată ca activare finală într-un clasificator, suma tuturor claselor nu este egală 1, cum ar trebui să fie în mod normal.
- Necesită calcul exponențial.

Funcția **tangentă** (*tanh*) sau tangenta hiperbolică, introdusă de Yann LeCun et al., este un alt tip de funcție de activare utilizată în Învățarea Profundă. Tanh este destul de similară funcția sigmoidă, fiind o funcție sigmoidă scalată, dar intervalul său este de la (-1, 1). Aceasta înseamnă că pentru orice neuron, nod sau activare pe care îl introducem, acesta va fi scalat la o valoare cuprinsă între -1 și 1. Forma *tanh* este, de asemenea, sigmoidală (în formă de S). Această funcție asigură o mapare puternică între intrările negative și zero, care vor fi în jur de zero în grafic. Alte proprietăți sunt funcțiile diferențiabile și monotone. Un punct forte al funcției *tanh* este că gradientul său este mai puternic, ceea ce înseamnă că derivatele sunt mai abrupte.
Tanh a fost deseori mult mai eficientă în cadrul rețelei neuronale în interiorul straturilor ascunse. Motivul este că atunci când folosim funcția sigmoidă, un set de intrări negative produce o ieșire aproape de 0. Asta înseamnă că este apoi dificil în schimbarea ponderilor asociate cu acel nod ascuns, în esență, devine un nod "mort". La acest dezavantaj vine ca soluție funcția *tanh*. Funcția tanh este o funcție centrată în zero, netedă [46] și este dată de relația:

$$f(x) = \frac{1-e^{-2\cdot x}}{1+e^{-2\cdot x}} \tag{2.3.3}$$

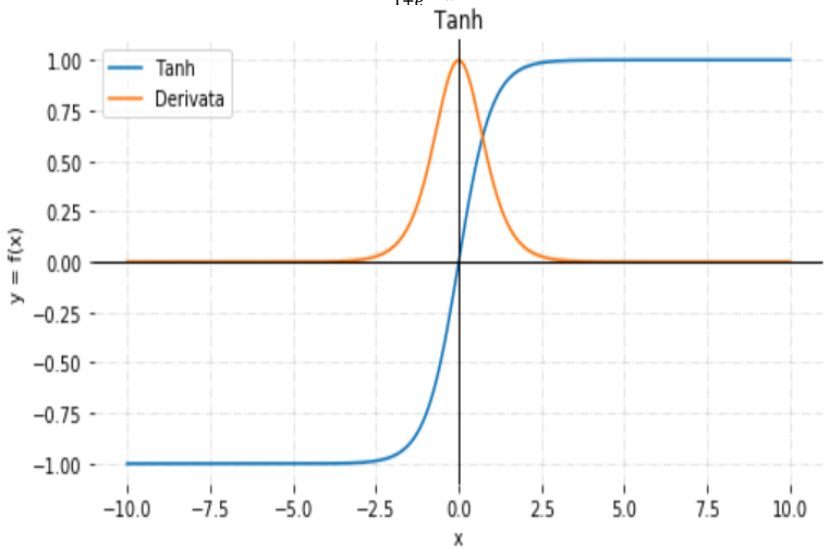


Fig.2.3 2 Funcția de activare *tanh*

Derivata funcției tangente este dată de următoarea relație:

$$\frac{d}{dx}\tanh(x) = 1 - \ tanh^2(x) \tag{2.3.4}$$

Pe măsură ce intrarea se apropie de 0, derivata funcției *tanh* se apropie de maximum 1. Și cum am văzut cu funcția sigmoidă, pe măsură ce intrarea se îndepărtează de 0 în oricare direcție, derivata funcției *tanh* se apropie de 0.

Funcția *tanh* a fost aleasă în detrimentul funcției sigmoide datorită performanțelor mai bune pe care le oferă în antrenarea rețelelor neuronale cu mai multe straturi. [70] [66] Totuși, funcția *tanh* nu a putut rezolva problema dispariției gradientului problemă care apare și în cazul funcțiilor sigmoide și de asemenea s-a remarcat că este greu de antrenat pe seturi de date mici. [69]

Principalul avantaj oferit de această funcție este acela că aceasta produce ieșirea centrată în zero ajutând în procesul de propagare înapoi. O proprietate remarcabilă a funcției *tanh* este aceea că poate atinge doar un gradient egal cu 1, doar când valoarea intrării este 0. Aceasta face ca funcția *tanh* să producă niște neuroni *"morți"* în timpul calculului. Neuronul mort corespunde condiției în care ponderea de activare, este rar utilizată ca urmare a gradientului care devine zero. Această limitare a funcției *tanh* a stimulat continuarea cercetării funcțiilor de activare pentru a rezolva această problemă, moment în care a apărut funcția de activare a unității liniare rectificate (Rectified Linear Unit - ReLU). [9] Funcțiile *tanh* au fost utilizate în majoritatea rețelelor neuronale recurente pentru procesarea limbajului natural (NLP) [71] și sarcinile de recunoaștere a vocii [72].

Funcția de activare unitate liniară rectificată ReLU (***Re****ctified* ***L****inear* ***U****nit*) a fost propusă de Nair și Hinton în 2010 și, încă de atunci, a fost cea mai utilizată pe scară largă pentru sarcinile de Învățare Profundă datorită perfomanțelor sale. [73] ReLU este cea mai rapidă funcție de activare, care s-a dovedit a fi cea mai reușită și mai folosită funcție. [3] Drept urmare, a devenit funcția implicită de activare pentru multe tipuri de rețele neuronale, deoarece un model care o folosește este mai ușor de antrenat și deseori obține performanțe mai bune. Oferă performanțe mai bune de generalizare în Învățarea Profundă spre deosebire de funcțiile de activare sigmoidă și *tanh*. [74] [75] ReLU se reprezintă aproape liniar și, prin urmare, păstrează proprietățile modelelor liniare care i-au permis o optimizare mai facilă, cu scăderea gradientului. [2]

Pentru a utiliza descendența stocastică a gradientului cu propagarea înapoi a erorilor în antrenarea rețelelor neuronale profunde, este necesară o funcție de activare care să arate și să funcționeze ca o funcție liniară, dar este, de fapt, o funcție neliniară care să permită învățarea relațiilor complexe a datelor. Funcția trebuie să acorde mai multă atenție la suma activării intrării care să evite saturația. Acest profil corespunde funcției ReLU. Un nod sau unitate care implementează această funcție de activare se numește o unitate de activare liniară rectificată. Adesea, rețelele care folosesc această funcție pentru straturile ascunse sunt denumite *rețele rectificate*.

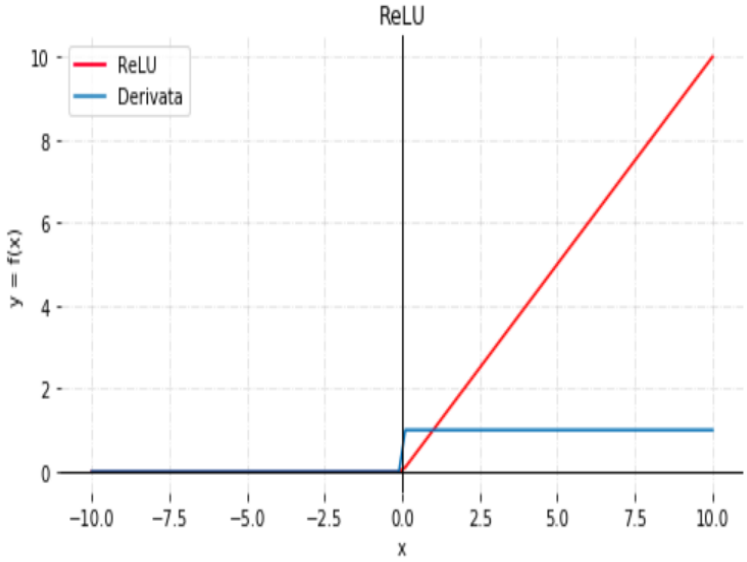


Fig.2.3 3 Funcția de activare ReLU și derivata sa

Funcția de activare ReLU efectuează o operație de prag pentru fiecare element de intrare unde sunt setate valori mai mici de zero, astfel că ReLU este dată de relația:

$$f(x) = \max(0, x) = \begin{cases} x_i, & dacă\ x_i \geq 0 \\ 0, & dacă\ x_i < 0 \end{cases} \tag{2.3.5}$$

Această funcție rectifică valorile intrărilor mai mici de zero, forțându-le la zero și eliminând gradientul de dispariție, problema observată în tipurile de funcții de activare anterioare. [9] Derivata funcției ReLU este ușor de calculat. Derivata funcției este panta. Panta pentru valori negative este 0 respectiv 1 pentru valori pozitive.

$$f'(\mathrm{x}) = \begin{cases} 1 \text{ if } \mathrm{x} > 0 \\ 0 \text{ if } \mathrm{x} \leq 0 \end{cases} \tag{2.3.6}$$

O altă proprietate a funcției ReLU este că distribuie valorile între zero și maxim. Cu toate acestea, ReLU are a limitare, referitoare la intrările negative, care devin 0, dar însă pe care o depășește cu ușurință în comparație cu funcția sigmoidă. [40]

Funcția ReLU are o serie de avantaje remarcabile și acestea sunt:

- *Simplitatea computațională*
  Mai puțină complexitate de timp și spațiu, în comparație cu sigmoida, nu evoluează exponențial, care e mai costisitor.
- *Spațialitate reprezentativă*
  Un beneficiu este dat de capacitatea acesteia de a da o valoare adevărată 0. Asta înseamnă că intrările negative pot returna *Adevărat* 0 permițând activării straturilor ascunse în rețeaua neuronală să conțină unul sau mai multe valori *Adevărat* 0. Asta se numește o *reprezentare spațială* și este foarte dorită, doarece accelerează învățarea și simplifică modelul.
- *Comportament liniar*: ReLU arată și acționează ca o funcție de activare liniară.

- Poate fi folosită la antrenarea rețelelor neuronale profunde.
- Evită problema dispariției gradientului.
- Nu se saturează în partea pozitivă.
- În practică, converge mult mai rapid decât sigmoida sau *tanh*.

Funcția ReLU are și *dezavantaje*, acestea fiind:

- Ieșirea nu are media egală cu zero.
- O situație nefavorabilă este dată de valorile negative, atunci când $x < 0$, gradientul este 0.
- ReLU nu evită problema exploziei gradienților.

Non-liniaritatea funcției ReLU

O funcție este neliniară dacă panta nu este constantă. Deci, funcția ReLU este neliniară în jurul valorii 0, dar panta este întotdeauna 0 pentru valori negative sau 1 pentru valori pozitive. Acesta este un tip de non-liniaritate foarte limitat.
Dar este de remarcat că în cazul rețelelor profunde, acestea ne permit să creăm multe tipuri diferite de neliniarități prin modul cum putem combina nodurile ReLU.

În primul rând, majoritatea modelelor includ un termen numit *bias* pentru fiecare nod. Termenul *bias* este doar un număr constant care este determinat în timpul formării modelului. Bias-ul ne permite să ne deplasăm acolo unde panta se schimbă.

Cu toate acestea, modelele reale au multe noduri. Fiecare nod, chiar și într-un singur strat, poate avea o valoare diferită pentru bias-ul său, astfel încât fiecare nod poate schimba panta la valori diferite pentru intrarea noastră.
Când adăugăm funcțiile rezultate înapoi, obținem o funcție combinată care schimbă pantele în multe locuri. Aceste modele au flexibilitatea de a produce funcții neliniare și de a ține cont de interacțiuni bune care va conduce la predicții mai bune. Pe măsură ce adăugăm mai multe noduri în fiecare strat sau mai multe convoluții dacă folosim un model de convoluție, modelul capătă o capacitate și mai mare de reprezentare a acestor interacțiuni și non-liniarități.

Chiar dacă domeniul Învățării Profunde ia amploare dezvoltându-se numeroase aplicații, dar Învățarea Profundă încă nu este foarte bine înțeleasă din punct de vedere teoretic. Lucru pe care încearcă să îl clarifice Tomaso Poggio et al. S-a arătat că rețelele profunde au garanția teoretică, pe care rețele superficiale nu o au, că pot evita blestemul dimensionalității pentru o clasă importantă de probleme, corespunzătoare funcțiilor compoziționale, adică funcții ale funcțiilor. Un subset interesant al unor astfel de funcții compoziționale îl reprezintă *funcțiile compoziționale ierarhice locale*. Printre rețelele profunde care pot aproxima fără blestemul dimensionalității menționăm rețelele de convoluție. S-a concluzionat că practic succesul Învățării Profunde este asigurat de proprietățile teoretice precum: arhitectura profundă de rețea de convoluție, supra-parametrizarea ei, utilizarea stocastică a scăderii gradientului, funcția de cost exponențială, omogenitatea funcției RELU. [76]

S-a arătat că funcțiile neliniare, cum ar fi rețelele neuronale pot fi aproximate local de planurile afine [77]. Studiile recente se folosesc de *input-Jacobians*, care descriu normala la aceste planuri. Srinivas et al. în lucrarea lor, intruduc *full-Jacobians*, care includ această normală împreună cu un termen suplimentar de interceptare numit *bias-Jacobians*, împreună descriind complet planele locale. Pentru rețele neuronale cu ReLU, *bias-Jacobians* corespunde sumelor de gradienți ale ieșirilor prin activarea stratului intermediar. Experimental, rezultatele arată o distilare

îmbunătățită pe seturi de date mici, generalizare îmbunătățită pentru antrenarea rețelei neuronale și hărți mai clare.

Yoshua Bengio et al., au arătat că în ciuda capacității lor covârșitoare de supraspecializare, arhitecturile de Învățare Profundă tind să generalizeze relativ bine pentru date nevăzute, permițându-le să fie folosite în practică. O ipoteză permanentă care câștigă renume susținută de Hochreiter et al. (1997), Keskar et al. (2017), este dată de planeitatea minimelor din funcția de cost dată de gradientul stocastic bazat pe metodele care au ca rezultat o generalizare bună. Mai mult de atât, au arătat că dacă permiteau reparametrizarea unei funcții, geometria parametrilor săi se poate modifica radical, însă fără a-i afecta proprietățile de generalizare. Ei s-au concentrat pe analiza erorii de estimare. Au folosit diferite abordări pentru a răspunde la întrebarea de ce descendența stocastică a gradientului conduce la o generalizare bună. Au clarificat, de asemenea, conceptul de *minim plat*. Dacă ne imaginăm eroarea ca pe o curbă unidimensională, minimul este plat dacă există o regiune având în jurul ei aproximativ aceeași eroare, altfel minimul este ascuțit. Atunci când ne raportăm la mai multe dimensiuni spațiale, definirea netezimii devine mai complicată. În Hochreiter et al. (1997) este definit ca fiind dimensiunea regiunii conectate în jurul valorii minime în care funcția de cost a antrenării este relativ aceiași. [78]

Datorită lipsei unei cantități mari de date de antrenare, acesta a constituit întotdeauna factorul constrângător în rezolvarea multor probleme în Învățarea Automată, transformând *One Shot Learning* într-una dintre cele mai intrigante idei de Învăţare Automată. S-au studiat arhitecturi de Învățare Profundă bazate pe metrici pentru o singură învățare precum *rețelele neuronale siameze (RNS)* [79] și s-a prezentat o metodă de îmbunătățire a preciziei lor folosind rețele *Kafnets* care sunt activări non-parametrice bazate pe funcții kernel pentru rețelele neuronale [80] prin învățarea încorporărilor corecte cu un număr relativ redus de epoci. Folosind funcțiile de activare a kernelului, Jadon et al., au obținut rezultate foarte bune care le depășesc pe cele ale funcției ReLU. [81]

Învățarea Profundă a jucat un rol decisiv în avansarea Învățării Automate, dar necesită, de asemenea, seturi de date mari. Diferite tehnici, cum ar fi regularizarea, reduc supraspecializarea în seturile de date mici, dar nu rezolvă inerenta problemă care vine cu mai puține date de antrenare. În plus, dimensiunea mare a seturilor de date conduce la învățare lentă, necesitând multe actualizări de ponderi folosind metoda SGD. Acest lucru se datorează mai ales datorită parametrilor modelului, în care datele de antrenare se învață lent având un număr mare de parametri de actualizat. S-au folosit în experimente seturi de date precum: *MNIST* [82] *AT&T Database of Faces* [83] respectiv *Omniglot* [84]. Setul de date MNIST este un set de date colectiv format din 0-9 cifre scrise de mână de diverși oameni. Este format dintr-un set de antrenare cuprinzând 60.000 de exemple și un set de test de 10.000 de exemple. Setul de date al fețelor AT&T constă în zece imagini diferite ale fiecăruia dintre cei 40 de oameni diferiți. Pentru anumite fețe, imaginile sunt făcute în alte momente ale zilei, în alte condiții de luminozitate precum și diferite expresii ale fețelor. Omniglot este unul din cele mai populare seturi de date utilizat de *One Shot Learning*. Este conceput special pentru a compara și a contrasta abilitățile de învățare ale oamenilor versus mașinilor. Acest set de date se compune din caractere scrise de mână din 50 de limbi cu un total de 1623 de caractere. Sunt doar 20 probe pentru fiecare caracter, fiecare fiind desenat de un individ distinct. Setul de date se împarte în 2 seturi: set de antrenare și set de test. Setul de antrenare conține 30 de alfabete (964 de caractere) și este utilizat pentru a antrena modelul, în timp ce restul de 20 de alfabete sunt folosite doar pentru test. [81]

Tian [85] a analizat rețeaua profundă cu ReLU, alegând relația elev-profesor în care rețeaua de studenți supra-parametrizată învață de la ieșirea unei rețele fixe de profesori de aceeași adâncime, folosind agoritmul SGD.
Analiza lui a arătat că supra-parametrizarea joacă două roluri:

(1) Este o condiție necesară pentru ca alinierea să se întâmple în punctele critice.

(2) În antrenarea dinamică, ajută nodurile studenților să acopere mai multe noduri ale profesorilor cu mai puține iterații, ambele reuducând problema supraspecializării. [85]

He et al., au definit o **unitate liniară rectificată parametrică** (***P**arametric **Re**ctifier - PReLU*) care este o generalizare a lui ReLU și totodată generalizează funcția de activare LReLU (**L**eaky ReLU). PReLU îmbunătățește reglarea modelului, cu costuri de calcul suplimentare aproape zero și riscuri de supraspecializare reduse. [86]
În cazul acestei funcții putem învăța parametrul de pantă α folosind propagarea înapoi având o creștere nesemnificativă a funcției de cost în timpul antrenamentului.

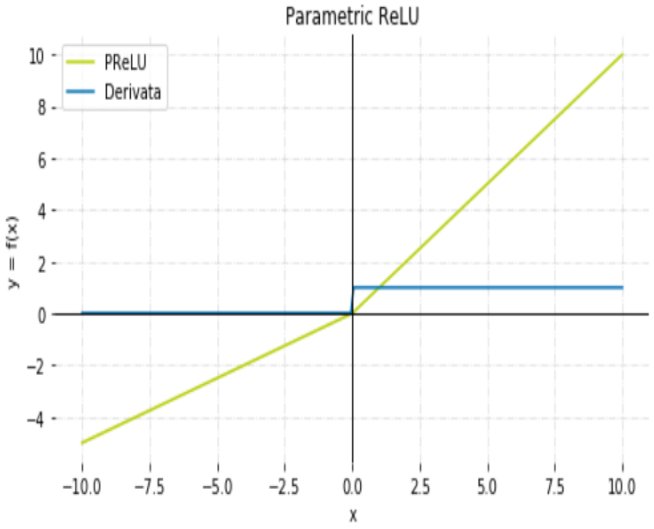


Fig.2.3 4 Funcția de activare PReLU $\alpha = 5$

Relația funcției de activare PReLU este dată de expresia:

$$f(y_i) = \begin{cases} y_i, & dacă\ y_i > 0 \\ \alpha_i y_i, & dacă\ y_i \leq 0 \end{cases} \tag{2.3.7}$$

Unde y_i reprezintă orice intrare de pe canalul *i* și α_i este panta negativă care este un parametru învățabil.
Putem avea următoarele situații:

a) Dacă $\alpha_i = 0$, PReLU devine ReLU.
b) Dacă $\alpha_i > 0$, funcția de activare devine Leaky ReLU funcție pe care o tratăm în următorul subcapitol.
c) Dacă α_i este un parametru învățabil, atunci avem funcția de activare PReLU.

Pentru a atenua potențialele probleme cauzate de gradientul egal cu 0 pentru valorile negative în cazul funcției ReLU, s-a introdus o nouă formă a funcției ReLU numită ***Leaky ReLU*** (LReLU) [86]. Redresorul *leaky* permite obținerea unui gradient diferit de 0, mic, atunci când unitatea este saturată și nu este activă.
Ecuația matematică a funcției LReLU este dată de relația:

$$f(x) = \max(0.01 \cdot x, x) \tag{2.3.8}$$

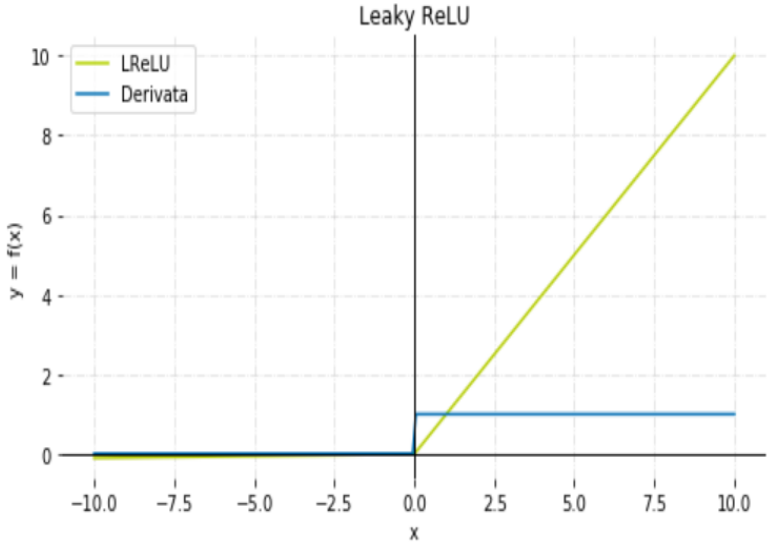


Fig.2.3 5 Funcția de activare LReLU

Principalele *caracteristici semnificative* ale funcției LReLU sunt:

- Foarte eficientă din punct de vedere computațional.
- În practică, converge mult mai rapid decât sigmoida/*tanh* (aproximativ 6 ori).
- Nu se saturează.

Are toate proprietățile ReLU, plus că nu va avea niciodată problema lui ReLU "mort". Pornind de la această variantă de funcție putem considera diferiți factori de înmulțire pentru redresorul *leaky* pentru a forma diferite variații ale Leaky ReLU.
Din reprezentarea sa grafică se poate vedea că LReLU sacrifică o sparsitate de zero pentru gradient, care are un potențial mai robust în timpul optimizării.

Clevert et al., în 2015 introduc o nouă funcție derivată din ReLU numită ***unitatea liniară exponențială*** (**E**xponential **L**inear **U**nits - ELU) care accelerează învățarea în rețelele profunde și conduce la precizii de clasificare superioare. ELU reduce problema dispariției gradientului prin identitatea valorilor pozitive.

Cu toate acestea, ELU are caracteristici de învățare îmbunătățite în comparație cu alte funcții de activare. Spre deosebire de ReLU, ELU are valori negative care îi permite să împingă activările medii ale unităților mai aproape de zero, cum ar fi normalizarea, dar cu o complexitate de calcul mai mică. Din punct de vedere experimental, ELU nu doar aduce o învățare mai rapidă ci și o mai bună generalizare decât ReLU și LReLU în cazul rețelelor cu mai mult de 5 straturi. [87]
Ecuația pentru funcția de activare ELU cu $\alpha > 0$ este dată de relația:

$$f(x) = \begin{cases} x, & \text{dacă } x > 0 \\ \alpha(\exp(x) - 1), & \text{dacă } x \leq 0 \end{cases} \quad (2.3.9)$$

Derivata funcției ELU este dată de relația:

$$f'(x) = \begin{cases} 1, & \text{dacă } x > 0 \\ f(x) + \alpha, & \text{dacă } x \leq 0 \end{cases} \quad (2.3.10)$$

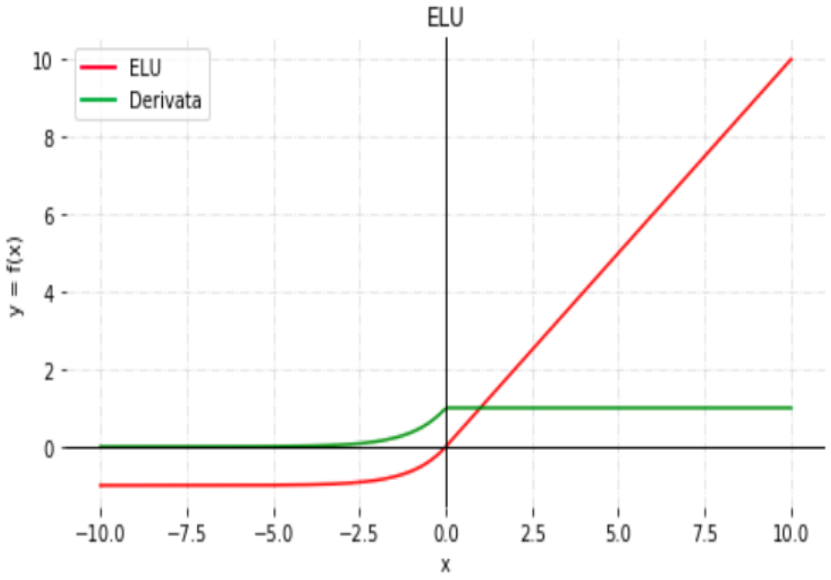


Fig.2.3 6 Funcția de activare ELU și derivate sa

Hiper-parametrul α controlează valoarea la care ELU se saturează pentru intrările negative ale rețelei neuronale.

S-a verificat la dezvoltarea acestei funcții dacă ELU păstrează activările medii mai aproape de zero decât alte funcții.

Spre deosebire de LReLU și PReLU, ELU are un platou de saturație evident în partea sa negativă, care îi permite să învețe o reprezentare mai robustă și mai stabilă. Printre câteva ***avantaje*** ale funcției ELU putem remarca:

- Toate beneficiile ReLU.
- Nu se saturează.
- Ieșirea medie aproape de zero.

Dezavantaj:

- Ca și funcțiile sigmoidă, *tanh* implică calcul exponențial.

Günter Klambauer et al., introduc auto-normalizarea rețelelor neuronale (SNN) pentru a permite reprezentări abstracte la nivel înalt. În timp ce normalizarea necesită normalizare explicită, activarea neuronilor de tip SNN converg automat către media zero și variația unității. Funcția de activare pentru rețelele SNN este dată de funcția numită *„**unități liniare scalate exponențial**”* (***Self**-Normalizing Neural Networks - SELU*), care introduc proprietăți de auto-normalizare. Proprietatea de convergență a acestor tipuri de rețele permite:

- antrenarea rețelelor profunde cu multe straturi,
- utilizează o regularizare puternică,
- face învățarea extrem de robustă. [88]

Funcția de activare SELU are următoarea relație:

$$\text{selu}(x) = \lambda \begin{cases} x & \text{dacă } x \geq 0 \\ \alpha e^{x} - \alpha & \text{dacă } x \leq 0 \end{cases} \tag{2.3.11}$$

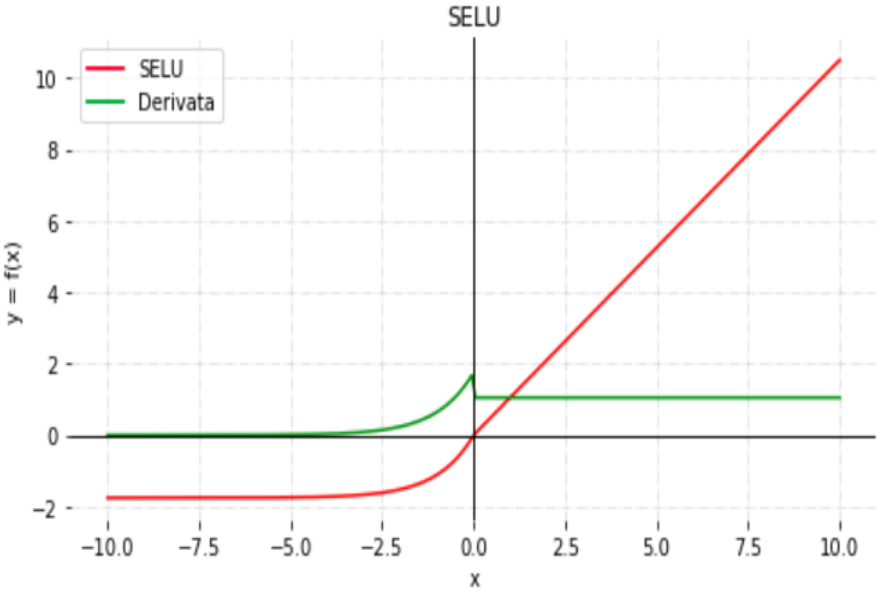


Fig.2.3 7 Funcția de activare SELU

Se poate observa că funcția de activare SELU este un fel de ELU, dar cu o mică răsucire, unde α și λ sunt doi parametri fixi, ceea ce înseamnă că nu ne propagăm înapoi prin intermediul lor și nu sunt hiperparametri pentru a lua decizii. Rețelele neuronale de tip SNN se inițializează cu ponderi de medie zero și folosesc deviația standard a rădăcinii pătrate de *1/(dimensiunea intrării).*

Dan Hendrycks et al., propun în 2016 o nouă funcție de activare numită funcția de activare **gausiană** (***G**aussian **E**rror **L**inear **U**nit* – GELU [89]) care este o funcție de activare performantă pentru rețeaua neuronală. Non-liniaritatea GELU se obține prin transformarea unui regulator stocastic, care aplică aleatoriu identitatea sau harta zero la intrarea unui neuron. Acest regulator stocastic este comparabil cu non-liniaritățile aduse prin tehnica *dropout*, dar elimină nevoia unei non-liniarități tradiționale. Conexiunea dintre GELU și regulatorul stocastic sugerează o nouă abordare probabilistică a non-liniarităților. Această funcție este doar o combinație între două funcții, și anume, funcția de activare *tanh* și numere aproximate. Ceea ce este mai interesant, este forma graficului pentru unitatea liniară de eroare gaussiană dat de următoarea figură:
Funcția de activare GELU are următoarea relație:

$$\mathrm{GELU(x)} = \mathrm{xP(X \leq x)} = x\emptyset(x) = x \cdot \frac{1}{2}[1 + \mathrm{erf}\,(\frac{x}{\sqrt{2}})] \quad (2.3.12)$$

Putem aproxima GELU cu:

$$0.5x(1 + \tanh\,[\sqrt{\frac{2}{\pi}}(x + 0.044715x^3]) \quad (2.3.13)$$

, sau

$$x\sigma(1.702x) \quad (2.3.14)$$

Se aplică relația (2.3.13) dacă viteza de propagare înainte este mai mare atunci merită să avem o valoare exactă.

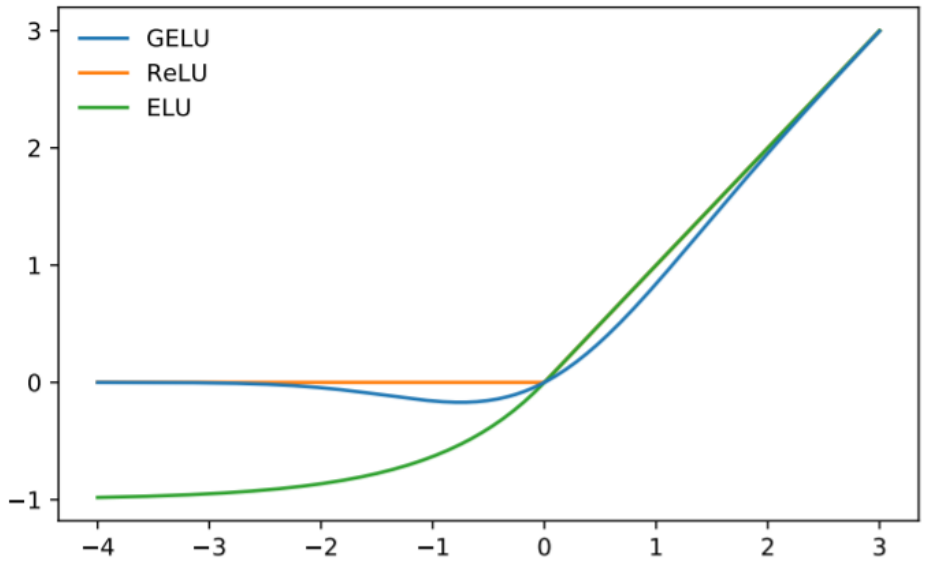


Fig.2.3 8 Funcția de activare GELU (μ = 0, σ = 1) versus ReLU și ELU (α = 1) [89]

Dacă funcția GELU care era o combinație de două funcții, în acest subcapitol introduc **funcția Swish** (A self-gated activation function), care vom vedea că și această funcție la rândul ei este o combinație de două funcții.

Funcția de activare Swish este una din primele funcții de activare compuse, propusă ca și o combinație dintre funcția de activare sigmoidă și funcția de activare ReLU, obținându-se astfel o funcție de activare hibridă.

Cu toate că au fost propuse diverse alternative ale funcției de activare ReLU, niciuna nu a reușit să o înlocuiască din cauza câștigurilor inconsistente. Funcția de activare SWISH [3] tinde să dea rezultate mai bune decât ReLU pe modele mai dense. De exemplu, doar prin înlocuirea lui ReLU cu Swish s-a obținut top-1 acurateță de clasificare pe ImageNet [90][91].

Studiile s-au concentrat pe găsirea unei funcții de activare scalare, care ia ca intrare un scalar și ca ieșire tot un scalar, deoarece funcțiile de activare scalare pot fi utilizate pentru a înlocui funcția ReLU fără a schimba însă arhitectura rețelei. S-a proiectat un spațiu simplu de căutare inspirat din spațiul de căutare al *optimizatorului*

lui Bello [92] care compune funcții unare și binare pentru a construi funcția de activare. [3]

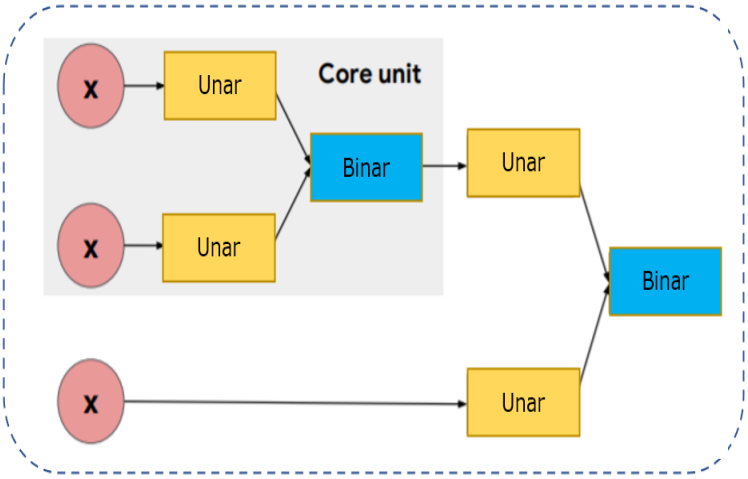


Fig.2.3 9 Structura funcției de activare [3]

Funcția de activare Swish are următoarea relație:

$$f(x) = x \cdot sigmoid(\beta x) = \frac{x}{1+e^{-\beta x}} \tag{2.3.15}$$

Unde β poate fi un parametru predefinit sau un parametru învățabil în timpul antrenării rețelei.

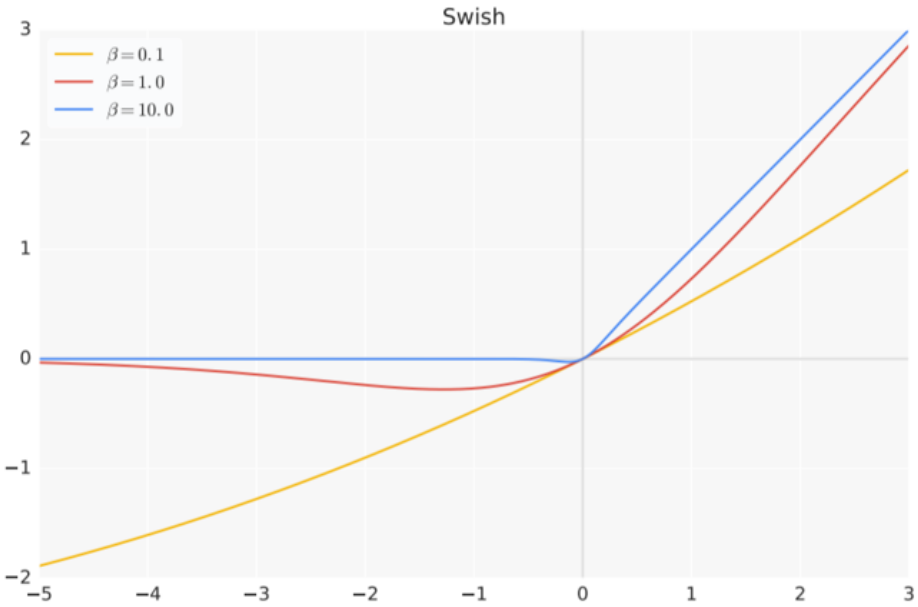


Fig.2.3 10 Funcția de activare Swish [3]

Principalele avantaje ale funcției Swish constă în simplitatea, precizia îmbunătățită și faptul că nu are probleme de dispariție a gradientului, oferind o propagare bună a informației în timpul antrenamentului. Mai mult decât atât, s-a observat că cele mai bune funcții de activare pot fi reprezentate de 1 sau 2 unități de bază (*core unit*).

O nouă funcție de activare derivată din funcția Swish a apărut în 2018, funcție numită ***E-Swish*** [93] care este foarte asemănătoare cu Swish având relația:

$$f(x) = \beta x \cdot sigmoid(x) = \frac{\beta x}{1+e^{-x}} \tag{2.3.16}$$

Când parametru β = 1, E-swish devine funcția Swish. La fel ca și funcțiile de activare ReLU și Swish, E-swish este nelimitată în partea pozitivă și delimitată în partea negativă. E-swish este netedă la fel ca funcția Swish. S-a observat că pentru valori mari ale parametrului β performanța funcției E-swish a scăzut. Valorile parametrului β care oferă îmbunătățiri atât pentru ReLU cât și pentru Swish se situează în intervalul $1 \leq \beta \leq 2$, s-a speculat că acest lucru se datorează exploziei gradienților când $\beta \leq 2$ și a problemelor de dispariție a gradienților când $\beta \leq 1$.
Totodată s-a descoperit că 1.125 și 1.25 sunt valorile pentru care această funcție a funcționat mai bine pe o arhitectură de tip SimpleNet. [93]

Funcția de activare exponențială liniară Squashing (**E**xponential **l**inear **S**quashing) cunoscută și sub numele de funcția de activare **EliSH** [94] este una dintre cele mai recente funcții de activare, propusă de Basirat și Roth în 2018. ELiSH are proprietăți comune cu funcția Swish.

Funcția ELiSH este alcătuită din ELU și funcția sigmoidă și este dată de relația:

$$f(x) = \begin{cases} \frac{x}{1+e^{-x}}, & dacă\ x \geq 0 \\ \frac{e^{x}-1}{1+e^{-x}}, & dacă\ x < 0 \end{cases} \tag{2.3.17}$$

Proprietățile funcției ELiSH variază atât în părțile negative, cât și în cele pozitive definite de limite. Partea sigmoidă din funcția ELiSH îmbunătățește fluxul de informații, în timp ce părțile liniare elimină problemele care apar datorită dispariției gradienților.

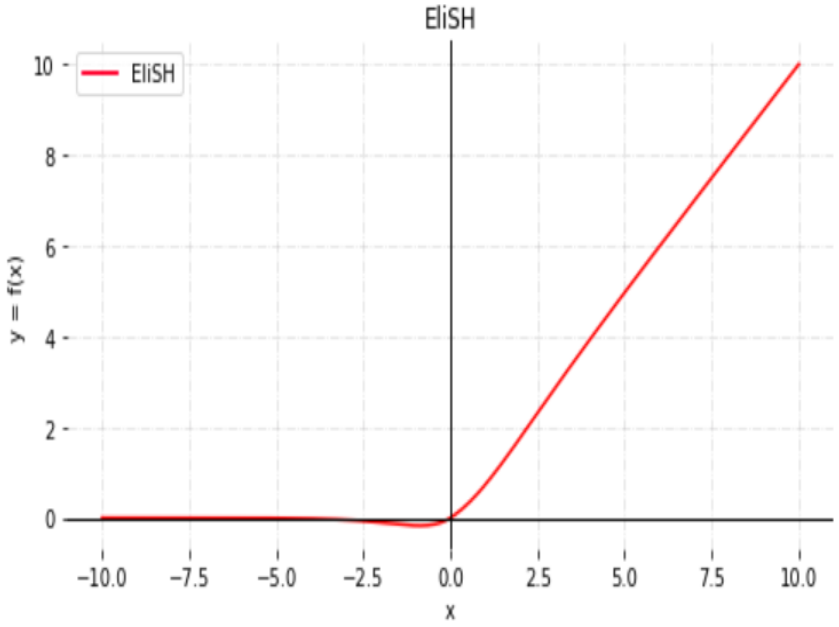


Fig.2.3 11 Funcția de activare EliSH

În mod similar, s-a introdus funcția de activare **HardELiSH** ca o înmulțire a HardSigmoid și ELU în partea negativă și HardSigmoid și ReLU în partea pozitivă [94]

$$f(x) = \begin{cases} x \cdot max\left(0, \min\left(1, \frac{x+1}{2}\right)\right), & dacă\ x \geq 0 \\ (e^x - 1) \cdot max\left(0, \min\left(1, \frac{x+1}{2}\right)\right), & dacă\ x < 0 \end{cases} \quad (2.3.18)$$

Mai mult de atât, în propunerea acestor două funcții s-a folosit conceptul de funcții de compoziție.

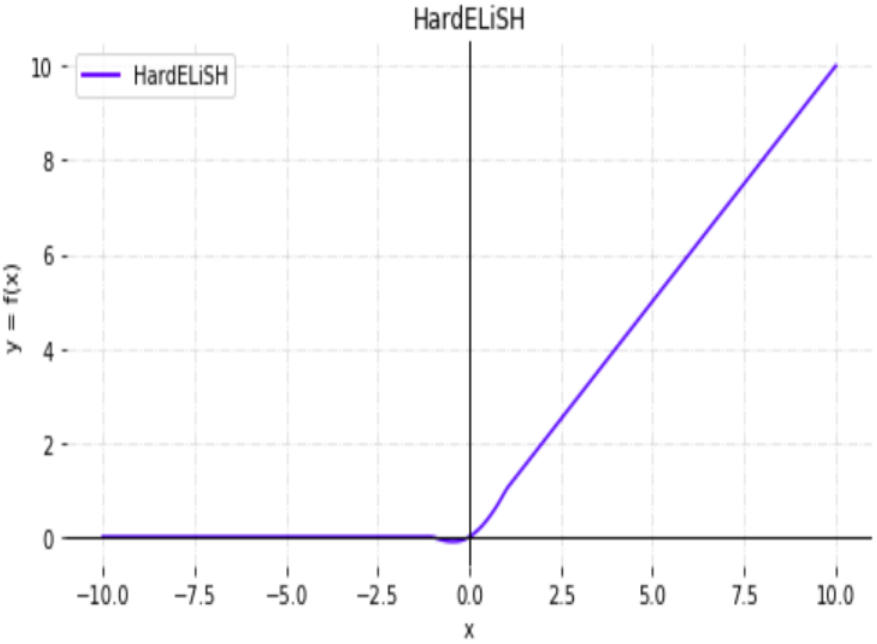


Fig.2.3 12 Funcția de activare HardELiSH

În urma studiului celor două funcții de activare s-a observat din rezultatele experimentale că sunt două tipuri de funcții de activare care iau în considerare doar partea pozitivă și anume:

a) contracție atunci când nu avem o expansiune și
b) dilatație care corespunde unei mapări expandate care poate fi considerată ca un aproximator universal.

În general, o expansiune extinde modificările valorilor de intrare, în timp ce o contracție este mai puțin sensibilă la schimbările valorilor intrărilor. [95]

În [96] o nouă funcție de activare este propusă, numită **Flatten-T Swish** (FTS) care aduce în plus beneficiul părții negative.

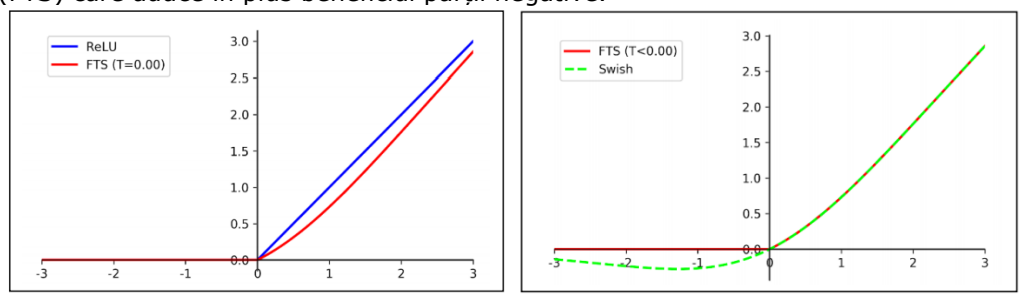


Fig.2.3 13 Funcția de activare FTS versus funcțiile ReLU și Swish [96]

FTS are proprietăți similare cu ReLU și Swish, ce apare nou este un prag dat de parametru *T*, introdus în scopul îmbunătățirii preciziei de clasificare într-o rețea de

tip propagare înainte și s-a constat că această funcție de activare aduce îmbunătățire majoră privind viteza antrenării rețelei, fiind de 2 ori mai rapidă decât funcția ReLU.

$$f(x) = \begin{cases} \frac{x}{1+e^{-x}}, & dacă\ x \geq 0 \\ 0, & dacă\ x < 0 \end{cases} \quad (2.3.19)$$

Ecuația funcției de activare prin adăguarea parametrului *T* devine:

$$f(x) = \begin{cases} \frac{x}{1+e^{-x}} + T, & dacă\ x \geq 0 \\ T, & dacă\ x < 0 \end{cases} \quad (2.3.20)$$

Derivata funcției FTS are relația:

$$FTS'(x) = \begin{cases} \sigma(x)\big(1 - f(x)\big) + f(x), & dacă\ x \geq 0 \\ 0, & dacă\ x < 0 \end{cases} \quad (2.3.21)$$

S-a observat că pentru valoarea de prag egală cu T=-0.20 s-au obținut cele mai bune rezultate.

Funcția de activare **Softplus** este o versiune a funcției ReLU care are proprietăți de înclinare și gradient diferit de 0, prin aceastea îmbunătățind stabilitatea și performanța rețelei neuronale profunde construite cu această funcție. Se consideră a fi o funcție învățabilă, crescătoare în două din argumentele sale și convexă pe unul dintre ele. [97]

Ecuația funcției de activare Softplus se definește astfel:

$$f(x) = \log(1 + e^x) \quad (2.3.22)$$

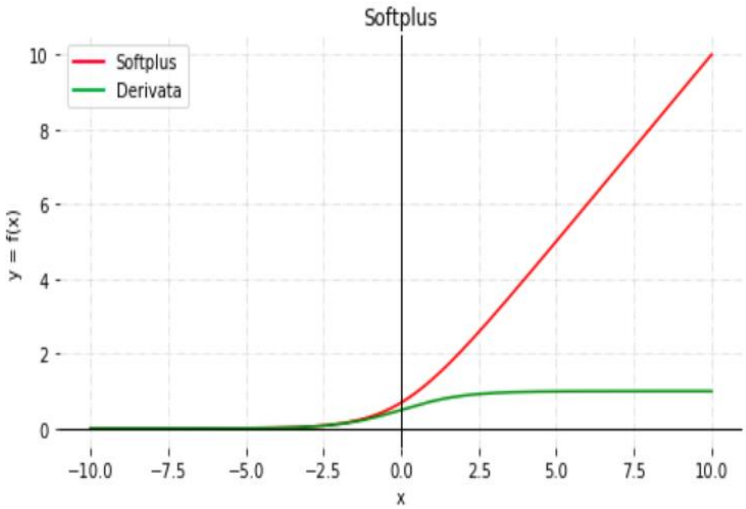


Fig.2.3 14 Funcția de activare Softplus

Comparativ cu ReLU, Softplus are o serie de avantaje. În primul rând, este netedă pe domeniul de definiție. Această proprietate îi conferă un comportament mai stabil indiferent dacă este estimată din direcțiile pozitive sau negative, în timp ce ReLU

are un gradient discontinuu în punctul 0. Un alt avantaj este că funcția Softplus are un gradient diferit de 0 în timp ce intrarea unității este negativă. Spre deosebire de ReLU care nu propagă gradienți pentru x <0, funcția Softplus poate propaga gradienți pe toate intrările reale. De remarcat faptul că derivata funcției Softplus este chiar funcția sigmoidă. Asta înseamnă că gradientul funcției Softplus ia valoarea 1 când intrarea crește, aspect care reduce efectele negative ale problemei datorate dispariției gradienților. Mai mult de atât, gradienții mari beneficiază de o reglarea fină atunci când se aplică dropout unităților. [98]

În 2019 se propune o nouă funcție de activare numită **Mish** [99], care nu este o funcție monotonă dar este o funcție netedă. Funcția Mish este o combinație între trei funcții de activare și anume: funcția ReLU, funcția *tanh* și funcția Softplus.
Ecuația funcției de activare Mish se definește astfel:

$$f(x) = \mathrm{x} \cdot \tanh\left(\mathrm{softplus(x)}\right) \tag{2.3.23}$$

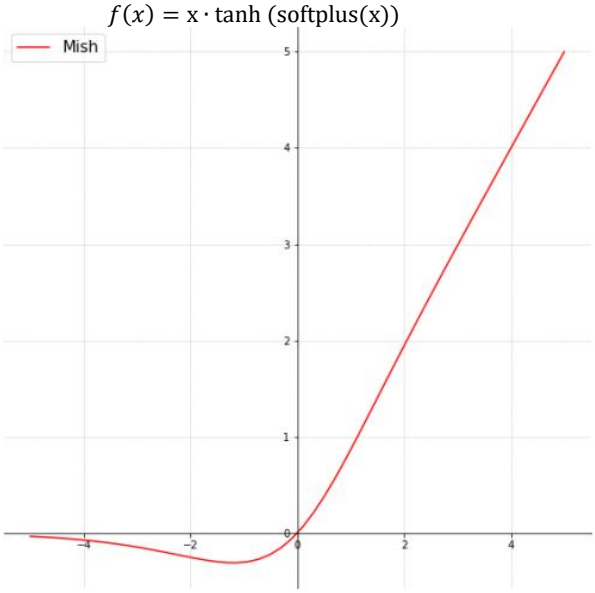


Fig.2.3 15 Funcția de activare Mish

Proprietățile funcției Mish, care o ajută să obțină performanțe superioare funcțiilor ReLU și Swish sunt:

- nemărginită în partea pozitivă,
- marginită în partea negativă,
- netedă,
- nu este monotonă.

La fel ca funcțiile ReLU și Swish, Mish este mărginită în partea negativă și nemărginită pe intervalul $[\approx -0.31, +\infty]$.
Derivata funcției Mish este dată de relația:

$$f'(x) = \frac{e^x w}{\delta^2} \tag{2.3.24}$$

Unde,

$$w = 4(x+1) + 4e^{2x} + e^{3x} + e^x(4x+6) \tag{2.3.25}$$

Și

$$\delta = 2e^{x} + e^{2x} + 2 \tag{2.3.26}$$

Minimul funcției Mish este la $x \approx -1.1924$ cu o magnitudine de ≈ -0.30884.

O nouă funcție de activare [100] **ARiA2**, fiind numită așa datorită faptului că este o versiune cu doi parametri a Curbei specializate a lui Richard și cunscută și sub denumirea de Activarea ponderată a Curbei adaptative a lui Richard (**A**daptive **Ri**chard's Curve weighted **A**ctivation – ARiA2). Nu este o funcție monotonă, asigurând un control mare asupra părții convexe care nu este monotonă prin intermediul variației hiper-parametrilor.

Ecuația funcției de activare ARiA se definește astfel:

$$\sigma(\alpha, \beta, x) = (1 + e^{-\beta x})^{-\alpha} \tag{2.3.27}$$

Unde α este un hiper-parametru care are un dublu efect: reduce curbura în al treilea cadran și totodată crește curbura în primul cadran în timp ce scade valoarea de activare (implicit $\alpha = 1$), iar β este rata de creștere exponențială (implicit $\beta = 0.5$).

Ca observație, funcția Swish ca în relația (2.3.15.2) este un caz particular al funcției ARiA și ARiA2 ca relația (2.3.15.3) derivă din funcția ARiA. ARiA devine Swish atunci când are forma:

$$ARiA = f(x,\ 1,\ 0,\ 1,\ 1,\ \beta,\ 1) \tag{2.3.28}$$

Iar ARiA2 are următoarea formă:

$$ARiA = f(x,\ 1,\ 0,\ 1,\ 1,\ 0,\ \beta,\ \tfrac{1}{\alpha}) \tag{2.3.29}$$

Derivata funcției ARiA2 este dată de relația:

$$\text{ARiA2}\ '(x) = \left(1 + e^{-\beta x}\right)^{-\alpha} + e^{-\beta x}\left(1 + e^{-\beta x}\right)^{-1-\alpha} x\alpha\beta \log\,[e] \tag{2.3.30}$$

În urma studierii acestei funcții s-a observat că pentru valori mici ale lui *x* (pozitive și negative), ARiA2 (și Swish) prezintă parte convexă în partea de sus, această curbură este complet absentă în cazul funcției de activare ReLU. Aceasta scade valoarea activării atunci când avem intrări mici negative, spre deosebire de ReLU în care astfel de intrări nu contribuie la valoarea activării. Pentru ponderi pozitive mici, din cauza non-liniarității lui Swish și ARiA2, valorile activării sunt mai mici decât pentru ReLU. Când o astfel de funcție este folosită în antrenarea arhitecturilor rețelelor neuronale profunde, stratul de normalizare a lotului (*Batch Normalization-BN*) scalează intrările până la următorul strat unde probabil că ponderile sunt definite în regiunea semnificativă. Derivata funcției de activare ARiA2 demonstrează că pentru valori negative mari, spre deosebire de ReLU, există un gradient mic pentru ARiA2 care este valabil și în cazul funcției de activare Swish, în timp ce pentru valori mari pozitive curba funcției ARiA2 devine similară cu RELU. Derivatele funcțiilor ARiA2 și Swish sunt continue, proprietate care le poate oferi un mic avantaj de calcul în comparație cu funcția de activare ReLU a cărui derivată este definită la valoarea 0 și atunci trebuie definită pe porțiuni.

Deci controlul convexității atât în partea pozitivă mică cât și cea negativă mică joacă un rol important pentru ca funcția de activare să ofere mai multă precizie și stabilitate în cadrul unui model.

În 2017 sunt propuse două funcții de activare pentru aproximarea funcției pentru rețeaua neuronală într-un domeniu nou Învățare prin consolidare (Reinforcement Learning - RL) [101] și anume: **unitatea liniară ponderată cu**

sigmoidă (SiLU) și funcția sa derivată (**dSiLU**) [102]. Funcția de activare SiLU este calculată ca produsul dintre funcția de activare sigmoidă și intrarea sa. RL este o arie din domeniul Învățării Automate care se concentrează pe modul cum acționează agenții într-un mediu în vederea obținerii unei recompense cât mai mari.

Activarea SiLU pare o versiune continuă și „sub-solicitată" a unității rectificate liniare (ReLU). Activarea dSiLU pare mai abruptă și este o versiunea „suprasolicitată" a funcției de activare sigmoidă. Pentru a testa performanța acestor funcții, autorii se folosesc de SZ-Tetris [103] stocastic, care este o versiune simplificată dar dificilă a lui Tetris. Funcția de activare SiLU este dată de relația:

$$a_k(s) = z_k\sigma(z_k) \tag{2.3.31}$$

Unde $\sigma(z_k)$ este dat de relația:

$$\sigma(z_k) = \frac{1}{1+e^{-x}} \tag{2.3.32}$$

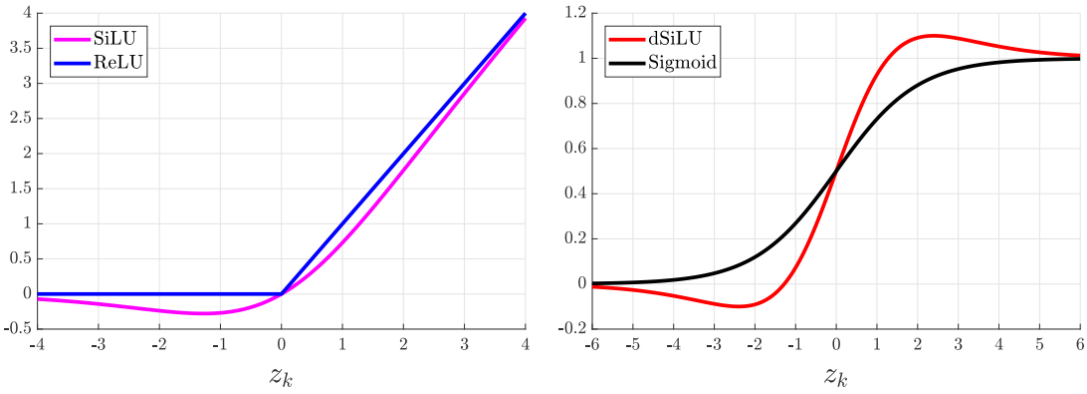


Fig.2.3 16 Funcțiile de activare SiLU și dSiLU

Din Fig.2.3.16 se poate observa că atunci z_k are valori mari, SiLU este aproximativ egală cu funcția ReLU, de asemenea activarea este aproximativ egală cu 0 pentru valori negative mari ale lui z_k, și aproximativ egal cu z_k pentru valori pozitive mari ale lui z_k. Spre deosebire de ReLU și alte funcții de activare utilizate frecvent, cum ar fi sigmoida și funcția *tanh*, activarea SiLU nu crește monoton. Totodată SiLU are un minim global care are o valoarea ≈ -0.28 pentru $z_k \approx -1.28$. O proprietate importantă a acestei funcții este dată de stabilitatea proprie.

Minimul global, unde derivata funcției devine zero, funcționează ca un „platou neted" pentru ponderi care servește ca un regulator implicit care inhibă învățarea ponderilor care au valori mari. Cea de-a doua funcție pe care o propun aceștia, numită dSiLU nu este altceva decât derivata funcției de activare SiLU și este dată de relația:

$$a_k(s) = \sigma(z_k)(1 + z_k(1 - \sigma(z_k))) \tag{2.3.33}$$

dSiLU are o valoare maximă ≈ 1.1 și o valoare minimă ≈ -0.1 pentru $z_k \approx \pm 2.4$.

În 2012 se implementează mai multe funcții de activare și anume RadBas, LogSig și TanSig [104] aplicabile în domeniul FPGA (**F**ield **P**rogrammable **G**ate **A**rrays

– [105]). FPGA reprezintă implementarea hardware a rețelelor neuronale artificiale. Funcțiile de activare sunt RadialBasis (RadBas), Logaritmic-Sigmoid (**Log**arithmic-**Sig**moid - LogSig) și **Tan**gent-**Sig**moid (TanSig). Proprietatea comună a acestor funcții este necesitatea calculului exponențial.
Funcția de activare RadBas este dată de relația:

$$f(x) = e^{-(x^2)} \tag{2.3.34}$$

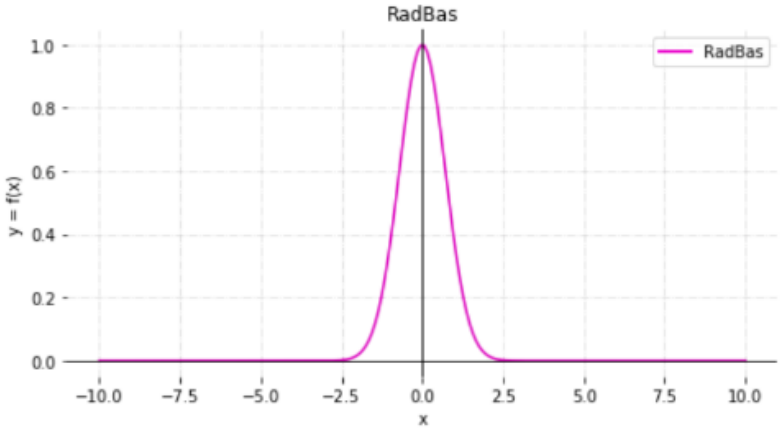


Fig.2.3 17 Funcția de activare RadBas

Funcția de activare LogSig (Fig.2.3.18) este dată de relația:

$$f(x) = \frac{1}{(1+e^{-x})} \tag{2.3.35}$$

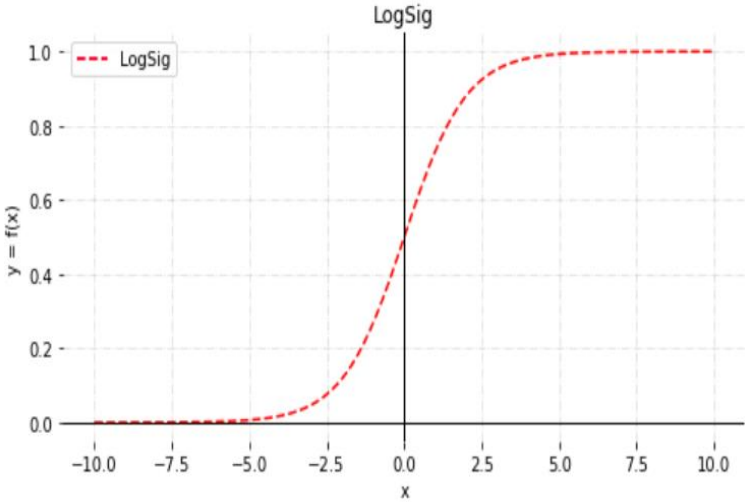


Fig.2.3 18 Funcția de activare LogSig

Funcția de activare TanSig (Fig.2.3.19) este dată de relația:

$$f(x) = \frac{2}{(1+e^{-2x})} - 1 \tag{2.3.36}$$

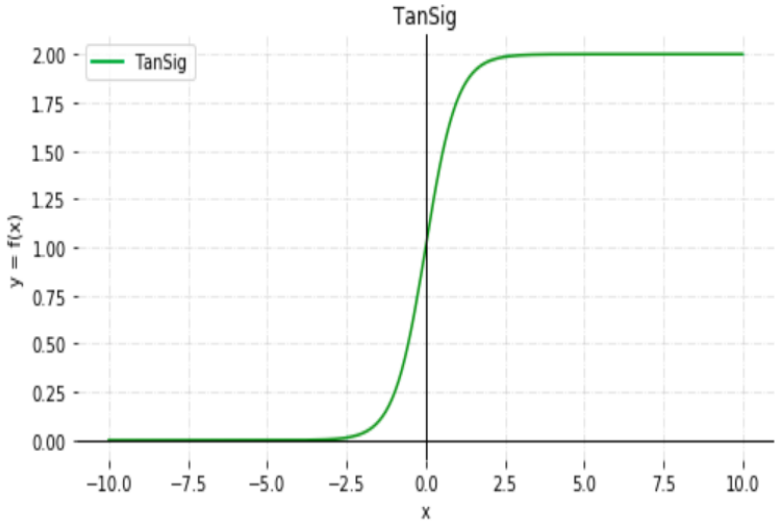


Fig.2.3 19 Funcția de activare TanSig

S-a constatat că funcțiile de activare RadBas și TanSig au aceiași perioadă de timp spre deosebire de LogSig care are perioade timp mai mari datorită adăugirilor în cascadă după extensia 2-bit.

În 1993 apare o funcție de activare numită **ElliotSig** (Elliot Sigmoid) [106] pentru a fi folosită în simularea digitală a rețelelor neuronale artificiale, în ideea că această funcție necesită un calcul mult mai redus decât funcțiile care au în componență funcții exponențiale. ElliotSig are o ecuație simplă divergentă care generalizează ecuația funcției logistice (sigmoida).

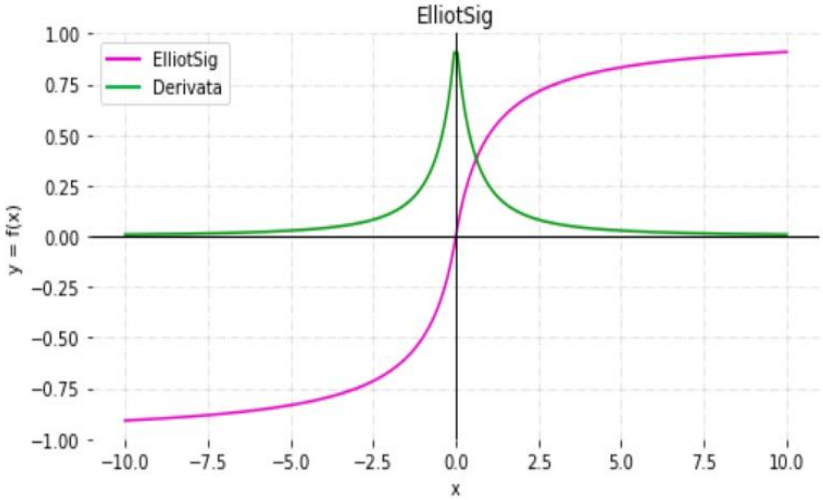


Fig.2.3 20 Funcția de activare ElliotSig

Funcția de activare ElliotSig este similară cu funcțiile sigmoidă sau tanh pentru valori mici, dar are nevoie de mai mult timp pentru a converge pentru valori mari, cu alte cuvinte nu merge la 1 sau 0 la fel de repede precum sigmoida și tanh, dar asta nu este o problemă în cazul în care se folosește pentru o sarcină de clasificare.
Funcția de activare ElliotSig este dată de relația:

$$f(x) = \frac{1}{(1+|x|)} \tag{2.3.37}$$

Funcția de activare ElliotSig este o funcție derivabilă, derivata sa este dată de relația:

$$f'(x) = \frac{1}{(1+|x|)^2} \tag{2.3.38}$$

Principala caracteristică a acestei funcții de activare este dată de viteza de convergență, fiind chiar de două ori mai rapidă decât funcția exponențială.

Se propune o nouă funcție de activare neliniară numită **SQNL**(**Sq**uare-Law **N**on-**L**inear) [107], această funcție de activare folosește operatorul pătrat pentru a introduce non-liniaritatea necesară spre deosebire de funcția TanSig care folosește operatorul exponențial. Această funcție se caracterizează printr-un număr mai mic de operații de calcul și succesul acestei funcții este dat de o convergență mai rapidă atunci când este utilizată împreună cu arhitecturi ale rețelei neuronale artificiale de tip multistrat.
Funcția de activare SQNL este dată de relația:

$$f(x) = \begin{cases} 1 & dacă \quad x > 2 \\ x - \frac{x^2}{4} & dacă \quad 0 \le x \le 2 \\ x + \frac{x^2}{4} & dacă \quad -2 \le x < 0 \\ -1 & dacă \quad x < -2 \end{cases} \tag{2.3.39}$$

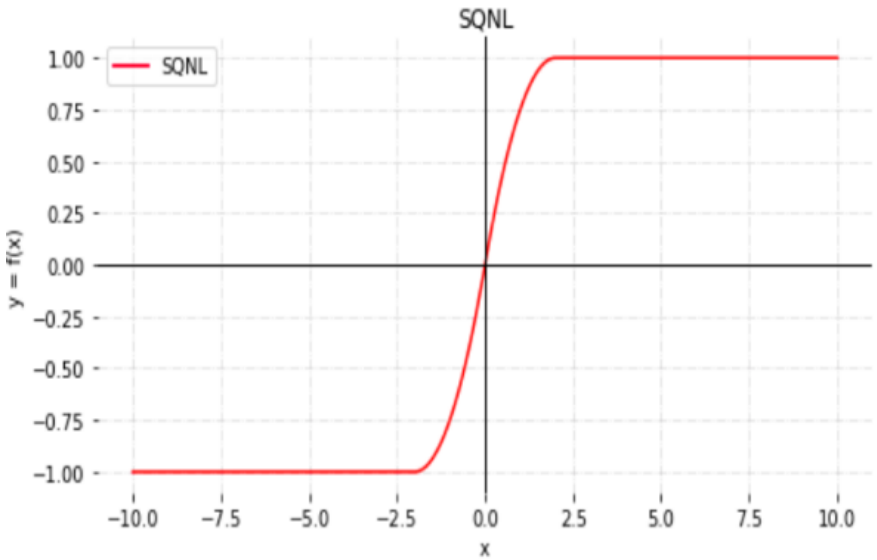


Fig.2.3 21 Funcția de activare SQNL

Proprietățile acestei funcții de activare sunt:

- Non-liniaritate simplă datorită operatorului pătrat care apare în ecuația sa.
- Simetrică și continuă, fiind o funcție de activare simetrică în jurul valorii 0 și este continuă în intervalul $[-\infty, +\infty]$.
- Are o derivată liniară.

Din punct de vedere al performanței înglobează două aspecte importante: viteza de convergență și capacitatea de a evita supraspecializarea.

Funcția de activare **ISRLU** (**I**nverse **S**quare **R**oot **L**inear **U**nit) [108] în 2017 a fost introdusă pentru a crește viteza pentru antrenarea rețelelor neuronale profunde. Această funcție de activare la fel ca și funcția SQNL are în componență operatorul pătrat, aducând performanțe de viteză comparabile cu ELU. De asemenea, această funcție are multe caracteristici similare cu funcția ELU. Ambele funcții de activare au valori negative, ceea ce le permite să aducă activarea medie a unității mai aproape de zero și să aproprie valorile gradienților către gradientul natural al unităților, totodată asigură o stare de îndepărtare a zgomotului, diminuând riscul de supraspecializare. De asemenea, avantajul semnificativ al funcției de activare ISRLU este dat de eficiența obținută pe procesoarele tradiționale în cazul implementărilor hardaware pe codul hardware/software pentru rețele neuronale de convoluție sau rețelele neuronale recurente. Pe lângă această funcție este menționată și funcția de activare ISRU (**I**nverse **S**quare **R**oot **U**nit) care poate fi folosită în cazul rețelelor neuronale recurente. Deși este mai puțin costisitoare, ISRU are o formă similară cu funcțiile de activare *tanh* și sigmoida.

Funcția de activare ISRLU este dată de relația:

$$f(x) = \begin{cases} x & dacă\ x \geq 0 \\ x\left(\frac{1}{\sqrt{1+\alpha x^2}}\right) & dacă\ x < 0 \end{cases} \tag{2.3.40}$$

Derivata funcției de activare ISRLU este dată de relația:

$$f'(x) = \begin{cases} 1 & dacă\ x \geq 0 \\ \left(\frac{1}{\sqrt{1+\alpha x^2}}\right)^3 & dacă\ x < 0 \end{cases} \tag{2.3.41}$$

Hiper-parametrul α controlează valoarea la care funcția de activare ISRLU se saturează pentru valorile negative. Prima și a doua derivată a acestei funcții de activare este netedă și continuă.

Avantajul major al funcției ISRLU constă în complexitatea sa de calcul redusă în comparație cu ELU, deoarece rădăcinile pătrate inverse sunt mai rapid de calculat decât cele exponențiale.

Funcția de activare ISRU este dată de relația:

$$f(x) = x\left(\frac{1}{\sqrt{1+\alpha x^2}}\right) \tag{2.3.42}$$

Derivata funcției de activare ISRU este dată de relația:

$$f'(x) = \left(\frac{1}{\sqrt{1+\alpha x^2}}\right)^3 \tag{2.3.43}$$

Funcția de activare Soft Clipping(SC) [109] introdusă în 2018, a fost dezvoltată pornind de la ideea că ELU nu poate produce valori exponențiale mari. Această funcție de activare are relația:

$$SC_p(x) = \frac{1}{p}\log\left(\frac{1+e^{px}}{1+e^{p(x-1)}}\right) \quad (2.3.44)$$

Această funcție de tăiere ușoară (*soft clipping*) este aproximativ liniară pentru $x \in (0, 1)$ și devine asimptotă foarte repede în afara acestui interval. Această funcție de activare are parametrul p care determină cât de aproape de liniaritate este regiunea centrală și cât de brusc se transformă regiunea liniară către valori asimptotice.

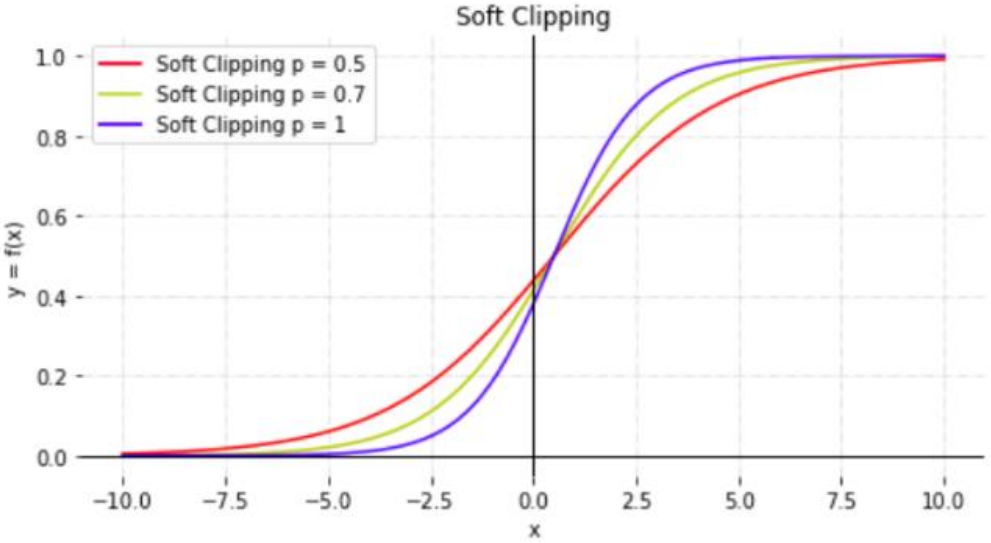


Fig.2.3 22 Funcția de activare Soft clipping pentru $p = 0.5, 0.7, 1$

În urma rezultatelor experimentale s-a constat că funcțiile ELU și SC au fost mai rapide în procesul de antrenare decât *tanh* si sigmoida.

În 2015 apare funcția de activare **SReLU** (**S**-shaped **Re**ctified **L**inear Activation **U**nit) [110] care este o unitate de activare liniară rectificată în formă de S dezvoltată pentru a învăța atât funcții convexe cât și non-convexe, imitând multiplele forme de funcții date de cele două legi fundamentale, și anume legea Webner-Fechner și legea Stevens, aplicate în psihofizică și neuroștiință. Această funcție de activare are patru parametri învățabili, învățarea se face folosind algoritmul de propagare înapoi.

În timpul antrenării, pentru inițializarea SReLU în diferite straturi, se propune o metodă de „înghețare" pentru a degenera SReLU într-o unitate liniară predefinită, rectificată, în mai multe epoci de formare inițiale și apoi să învețe în mod adaptiv valorile inițiale bune. SReLU poate fi utilizată în rețelele neuronale profunde cu parametri suplimentari dar totuși nu aduc costuri de calcul suplimentare.

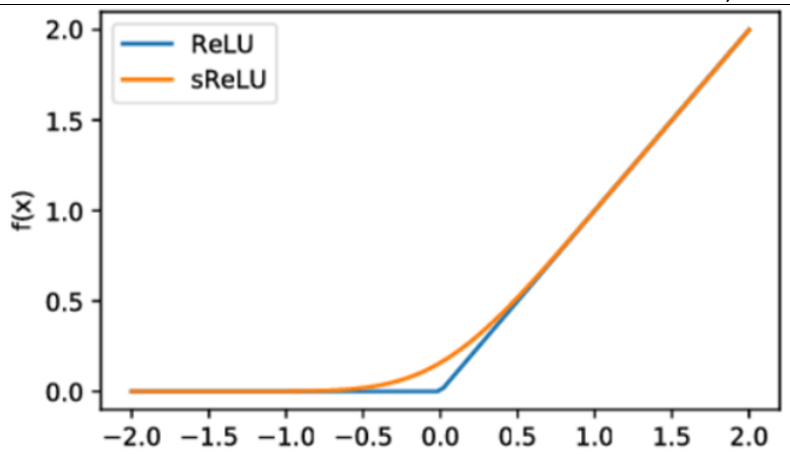


Fig.2.3 23 Funcția de activare SReLU

În esență SReLU este o combinație de funcții liniare și este de forma:

$$f(x) = \begin{cases} t_i^r + a_i^r\,(x_i - t_i^r) & dacă\ x_i \geq t_i^r \\ x_i & dacă\ t_i^r > x_i > t_i^l \\ t_i^l + a_i^l\,(x_i - t_i^l) & dacă\ x_i \leq t_i^l \end{cases} \quad (2.3.45)$$

Cei patru parametri învățabili de care spuneam mai devreme sunt $\{t_i^r, a_i^r, t_i^l, a_i^l\}$.
Avantajele folosirii acestei funcții de activare sunt:

- SReLU poate învăța atât funcții convexe cât și non-convexe, fără a impune restricții asupra parametrilor săi învățabili, astfel rețeaua neuronală profundă cu SReLU are o capacitate de învățare a caracteristicilor mai puternică.
- Având în vedere că SReLU este compusă din funcții liniare, mai degrabă decât funcții saturate, are astfel aceleași avantaje ca ale funcțiilor de activare nesaturate: nu suferă de problema exploziei gradienților respectiv dispariției gradienților și are o viteză de calcul mare în timpul antrenării rețelelor neuronale profunde cu propagarea înapoi.

În 2018 apare funcția de activare **BReLU** (**B**ipolar **Re**ctified **L**inear Activation **U**nit) [111] care derivă din funcția de activare ReLU, aceasta permite mutarea activării medii spre valoarea zero. BReLU împreună cu o inițializare corectă a ponderilor poate substitui necesitatea straturilor de normalizare, inițializarea s-a făcut cu algoritmul variației unităților pe straturi secvențiale (Layer-Sequential Unit Variance LSUV [112]). Această funcție de activare a fost analizată și testată folosind rețelele neuronale recurente tip "vanilla" având până la 144 de straturi și se arată că funcțiile de activare bipolare ajută la învățarea în astfel de condiții. Rețelele neuronale recurente tip "vanilla" sunt o îmbunătățire peste LSTM (Long Short-Term Memory) care analizează problema dispariției gradienților prin CEC ("Constant Error Carousel").

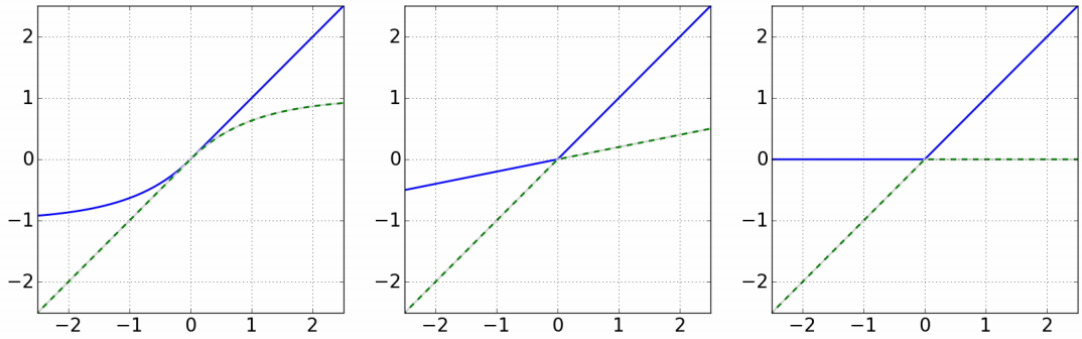


Fig.2.3 24 Funcțiile de activare Bipolar ELU, Bipolar Leaky ReLU, Bipolar ReLU [111]

Funcția de activare BReLU (Bipolar ReLU) este dată de relația:

$$f_B(x_i) = \begin{cases} f(x_i) & dacă\ i\ mod\ 2 = 0 \\ -f(-x_i) & dacă\ i\ mod\ 2 \neq 0 \end{cases} \quad (2.3.46)$$

BReLU nu face altceva decât să aducă media activării către valoarea 0, care are ca efect o stabilitate a activărilor.

În 2016 apare funcția de activare **CReLU** (**C**oncatenated **Re**ctified **L**inear **U**nit) [113] care este o funcție care păstrează atât informațiile pozitive, cât și pe cele negative în timpul antrenării în timp ce aplică non-liniaritatea nesaturată. CReLU permite o caracterizare matematică a straturilor de convoluție în vederea proprietății de reconstrucție, care este un indicator important pentru a vedea cât de expresive și generalizabile sunt caracteristicile rețelelor de convoluție. Prin folosirea acestei funcții, pur și simplu înlocuind ReLu cu CReLU, s-a obținut o reducere de parametri fără să impacteze performanța rețelei. ReLU concatenat are două ieșiri, o ieșire ReLU și o ieșire ReLU negativă, concatenate împreună. Cu alte cuvinte, pentru x pozitiv produce *[x, 0]*, iar pentru x negativ produce *[0, x]*. Deoarece are două ieșiri, CReLU dublează dimensiunea de ieșire.

Funcția de activare CReLU este dată de relația:

$$f(x) = (\max(0, x), \max(0, -x)) \quad (2.3.47)$$

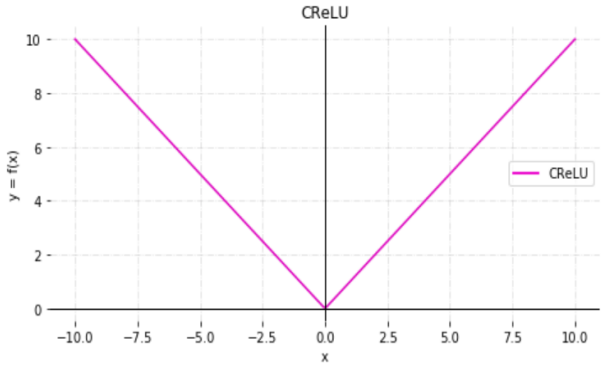


Fig.2.3 25 Funcția de activare CReLU

S-a observat că această funcție de activare are ca rol gruparea perechilor. Rețeaua cu funcția de activare CReLU nu se concentrează pur și simplu pe informațiile modulului, ci impune manipulări diferite în faze opuse. În [114] s-a combinat funcția de activare CReLU cu o inițializare a ponderilor în oglindă.
Funcția de activare CReLU poate fi scrisă ca în relația următoare:

$$x \to \begin{pmatrix} \rho(x) \\ p(-x) \end{pmatrix} \tag{2.3.48}$$

Inițializarea ponderilor cu funcția de activare CReLU:

$$(W - W) \cdot \begin{pmatrix} \rho(x) \\ p(-x) \end{pmatrix} = W\rho(x) - W\rho(-x) = Wx \tag{2.3.49}$$

Ieșirea va înceta să fie liniară imediat după ce se actualizează ponderile care determină divergența celor două blocuri ca în ecuația (2.3.49).

În 2013 apare funcția de activare **Maxout** [115] care este o funcție în care non-linearitatea este aplicată ca produs între ponderile unei rețele neuronale și date. Funcția Maxout generalizează ReLU în care neuronul moștenește proprietățile ReLU și unde nu există neuroni morți sau saturație în calculul rețelei.
Funcția Maxout este dată de relația:

$$f(x) = \max\left(w_1^T x + b_1, w_2^T x + b_2\right) \tag{2.3.50}$$

Unde *w* = ponderi, *b* = deplasări, *T* = transpunere. Principalul dezavantaj al funcției Maxout este că este costisitoare din punct de vedere computațional deoarece dublează numărul de parametri utilizați de toți neuronii, crescând astfel numărul de parametri care trebuie calculați de rețea.

Goodfellow et al., în 2013 definesc un model nou numit ***Dropout*** pentru design-ul unui model bazat pe aproximarea medie. De asemenea definesc și un model simplu numit ***Maxout***, numit așa pentru că ieșirea este valoarea maximă a setului de intrare și pentru că se alătură modelului dropout, ambele au fost create pentru a facilita optimizarea cauzată de dropout și a îmbunătăți viteza dropout-ului asupra tehnicii modelului de aproximare medie. Au folosit maxout și dropout să demonstreze performanța state of the art pe 4 seturi de date mari: MNIST [82], CIFAR-10, CIFAR-100 [30] și SVHN [116] .

Modelul dropout este considerat de Hinton în [117] mai ușor de implementat decât modelul Bayesian în care ponderile fiecărui model sunt calculate de probabilitatea posterioară dată de setul de antrenare. Atunci când se doresc clase de modele mai complexe precum rețelele neuronale de tip propagare înainte, metodele Bayesiane folosesc metoda Markov chain Monte Carlo pentru a proba modele din distribuția posterioară. [118]

Rețelele Maxout nu învață doar relația dintre unitățile ascunse, ci și activarea funcției fiecărei unități ascunse. În figura următoare se poate observa modul în care funcționează acest lucru:

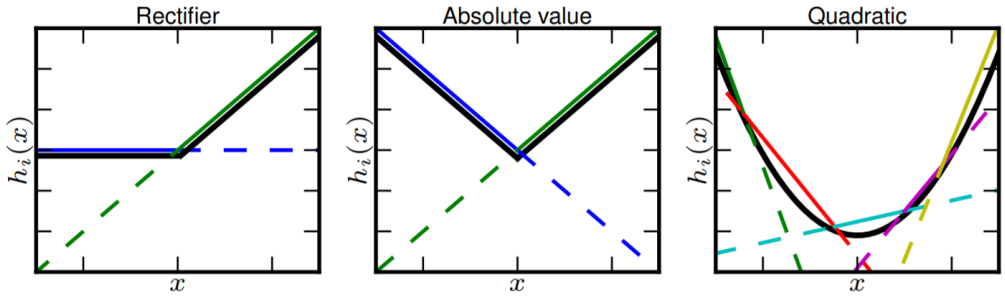


Fig.2.3 26 Funcția de activare Maxout

Se poate observa din Fig.2.3.26 că funcția de activare Maxout poate fi implementată pentru rectificator liniar, rectificator al valorii absolute respectiv prin aproximarea activării cvadratice a funcției [115].
Atunci când se dă un input $x \in R^d$, *x* poate fi *v* sau poate fi starea unui strat ascuns, atunci funcția pentru un strat ascuns Maxout devine:

$$\mathrm{h_i(x)} = \max \mathrm{z_{ij}} \quad \text{unde } \mathrm{j} \in [1, \mathrm{k}] \tag{2.3.51}$$

$$\mathrm{z_{ij}} = \mathrm{x^T W_{\dots ij}} + \mathrm{b_{ij}} \tag{2.3.52}$$

Unde, $W \in R^{dxmxk}$ *şi* $b \in R^{mxk}$ sunt parametri învățabili.

Avantajele funcției de activare Maxout sunt:

- Nu are forma generală a produsului scalar ceea ce conduce la non-liniaritate.
- Generalizează funcțiile de activare ReLU și Leaky ReLU.
- Liniară pe intervale, nu se saturează.

Principalul ***dezavantaj*** care se remarcă în dublarea numărului de parametri per neuroni.

În 2016 apare funcția de activare **OPLU** (**O**rthogonal **P**ermutation **L**inear **U**nit Activation Functions) [119] care este o funcție adaptivă care implementează mapări ortogonale neliniare a elementelor pe baza permutărilor. Principalele sale avantaje sunt necesitatea redusă de momorie și eficiența din punct de vedere computațional.

Funcția de activare OPLU asigură păstrarea normalizată a gradienților înapoi, care reprezintă un potențial bun pentru formarea rețelelor neuronale profunde, foarte profunde, chiar și a celor recurente. Funcțiile de activare tradiționale anulează proprietatea de ortogonalitate în cazul propagării înapoi, drept urmare se anihilează păstrarea gradienților normalizați, lucru însă asigurat de OPLU. Funcția de activare produce un vector de valori postsinaptice *z* de aceeași dimensiune pentru un vector de valori presinaptice *a* ca în următoarea relație:

$$\begin{pmatrix} z_i \\ z_j \end{pmatrix} = \begin{pmatrix} \max(a_i, a_j) \\ \min(a_i, a_j) \end{pmatrix} \tag{2.3.53}$$

Se fac permutări de perechi ale valorilor presinaptice după următoarea regulă:

$$\begin{cases} (z_i \;\; z_j)^T = (a_i \;\; a_j)^T & dacă \quad a_i \geq a_j \\ (z_i \;\; z_j)^T = (a_j \;\; a_i)^T & altfel \end{cases} \tag{2.3.54}$$

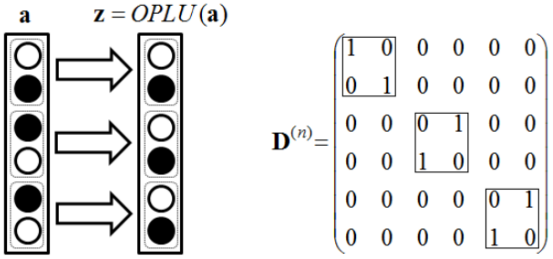


Fig.2.3 27 Permutări făcute cu funcția de activare OPLU [119]

Funcția de activare OPLU, din punct de vedere al performanței, dă rezultate similare cu funcțiile de activare *tanh* și ReLU. De remarcat faptul că un izomorfism ar putea fi stabilit între funcția de activare OPLU și funcția de activare Maxout [115].

Funcția de activare APL (**A**daptive **P**iecewise **L**inear units) [120] în 2015 s-a dezvoltat pornind de la faptul că rețelele neuronale artificiale au de obicei o funcție fixă, neliniară de activare la fiecare neuron. Au conceput o formă nouă de funcție liniară de activare liniară, care este învățată independent pentru fiecare neuron, folosind scăderea gradientului.

Cu această ***funcție de activare adaptativă***, au îmbunătățit arhitecturile de rețele neuronale profunde compuse din unități liniare rectificate statice. O modalitate de explorare constă în învățarea funcției de activare în timpul antrenării. Eforturile anterioare pentru a face acest lucru s-au concentrat în mare măsură pe algoritmi genetici și evolutivi [121], care încearcă selecția unei funcții de activare pentru fiecare neuron dintrun set predefinit. Recent, Turner et al., au combinat această strategie cu un singur parametru scalat care este învățat în timpul antrenării. [122]

$$h_i(x) = \max(0,x) + \sum_{s=1}^{S} \alpha_i^S \max(0, -x + b_i^S) \qquad (2.3.55)$$

Unde *S* este hiper-parametru, α_i^S, b_i^S , pentru $i \in 1..S,$ sunt învățate folosind scăderea gradientului în timpul antrenării.
α_i^S, controlează pantele segmentelor liniare, b_i^S determină locațiile curbelor.

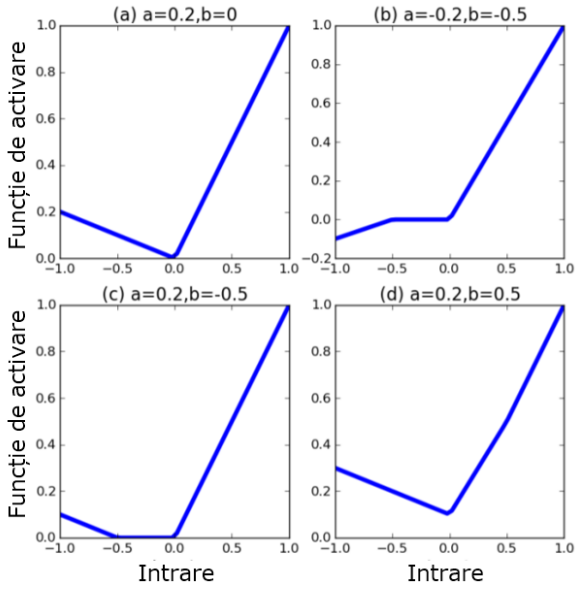


Fig.2.3 28 Funcția de activare APL [120]

În Fig.2.3.28 se pot observa funcții de activare obținute prin modificarea parametrilor. Se remarcă că cifra *b* indică faptul că funcția de activare poate fi și ea non-convexă. [122]

Agostinelli et al., au observat că atât unitățile Maxout, cât și rețeaua nu pot învăța orice funcție de activare liniară pe când unitățile APL o pot face, dar necesită mult mai mulți parametri pentru a face acest lucru. Această diferență permite APL să aibă un număr mic de parametri necesari pentru reglaj, antrenarea rețelei de convoluție prin aplicarea diferitelor non-liniarități în fiecare punct al fiecărei hărți de caracteristici, ceea ce ar fi complet nepractic pentru abordările din rețelele Maxout sau Rețea în Rețea (*NiN -network in network*). [123]

Funcția de activare **Softmax** apărută în 2016, întâlnită și ca **Softargmax** [124] sau funcție exponențială normalizată (*normalized exponential function*) [125] este o funcție de tip funcție squashing. Funcțiile squashing limitează ieșirea funcției în intervalul *[0, 1]*. Asta permite ca ieșirea să fie interpretată direct ca o probabilitate. Cu alte cuvinte, funcțiile Softmax sunt sigmoide cu mai multe clase, ceea ce înseamnă că sunt utilizate pentru a determina probabilitatea mai multor clase simultan. Deoarece rezultatele unei funcții Softmax pot fi interpretate ca o probabilitate, adică suma să fie 1, un strat Softmax este de obicei stratul final utilizat în funcțiile Rețelei Neuronale Artificiale. Este important de luat în considerare că un strat Softmax trebuie să aibă același număr de noduri ca ieșirea.

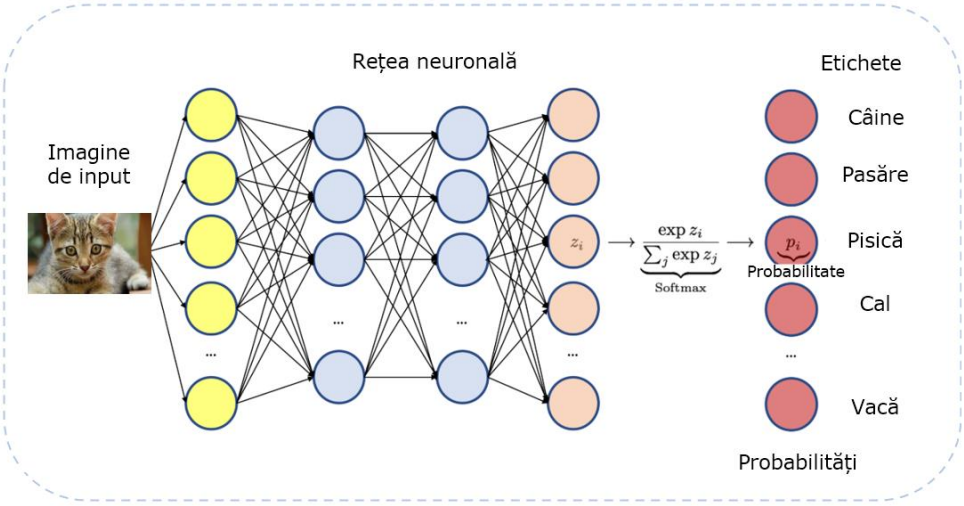


Fig.2.3 29 Expresia funcției de activare Softmax

Funcția Softmax este folosită pe scară largă de multe rețele CNN datorită simplității și interpretării probabiliste. Împreună cu entropia încrucișată, formează una dintre componentele cele mai utilizate în arhitecturile CNN. [26] Pe scurt, este o aproximare netedă a funcției max, care arată ca funcția ReLU. Partea lină și netedă este cheia acestei funcții, ceea ce o face să fie diferențiabilă.
Relația de calcul a funcției Softmax este:

$$f(x) = \text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{2.3.56}$$

Intrarea fără normalizare pentru o funcție Softmax se numește ***logit***. Această intrare este de obicei ieșirea unei rețele neuronale. Funcția Softmax are logit-uri și probabilitățile ieșirilor care însumate dau 1.

În 2017 apare funcția de activare **Large-margin Softmax** [126] care este o generalizare a funcției de activare Softmax numită ***Large-margin softmax loss*** (L-Softmax) care nu face altceva decât să încurajeze în mod explicit compactitatea intra-clasă și separabilitatea inter-clase între caracteristicile învățate.

Mai mult decât atât, L-Softmax nu doar că poate atinge scopul dorit, ci și poate evita supraspecializarea. De asemenea, se arătă că funcția de cost pentru L-Softmax poate fi optimizată prin folosirea scăderii stocastice a gradientului.

În ultimii ani, Softmax și SGD au devenit o componentă frecvent utilizată și, respectiv, strategia de formare implicită în cadrul rețelelor de convoluție. Cu toate acestea, atunci când se optimizează CNN cu SGD, comportamentul de saturație din spatele funcției Softmax ne oferă întotdeauna o iluzie de antrenare bună și atunci saturația este omisă. Binghui Chen et al., prin lucrarea lor în 2017, subliniază mai întâi faptul că acest comportament de saturație precoce al funcției Softmax va împiedica explorarea SGD-ului, care este uneori un motiv pentru ca modelul să conveargă la un minim local greșit, astfel propunând o nouă funcție de activare derivată din Softmax numită ***Noisy Softmax***, care atenuează această problemă de saturație timpurie prin injectare de zgomot care însă va fi recuperat prin Softmax în timpul fiecărei iterații. [127]

Apariția funcției de activare Noisy Softmax constituie state-of-the-art pe mai multe seturi de date precum: MNIST [82], CIFAR-10 [30], CIFAR-100 [30], LFW [128], FGLFW [129] și YTF [130].

Funcția de activare **SparseMax** [131] apărută în 2016, este o versiune a funcției Softmax, dar care are capacitatea să dea ca ieșiri probabilități aleatorii. Împreună cu această funcție de activare se propune și o funcție de cost, netedă și convexă, care este analoaga Sparsemax a funcției de cost logistice. Aceste propuneri au fost testate pe o sarcină de clasificare multi-clase pentru inferența limbajului natural obținându-se rezultate similare cu ale funcției Softmax, dar soluția fiind mai robustă. Relația la Sparsemax este dată de relația:

$$sparsemax(z) \coloneqq \underset{p\epsilon\Delta^{K-1}}{argmin}||p-z||^2 \tag{2.3.57}$$

Unde,

$$\Delta^{K-1} \coloneqq \{p\epsilon R^K | 1^T p = 1,\ p \geq 0\} \tag{2.3.58}$$

va fi $K-1$ simplex dimensional, funcțiile de interes se mapează în domeniul R^K la distribuțiile de probabilitate în Δ^{K-1}.

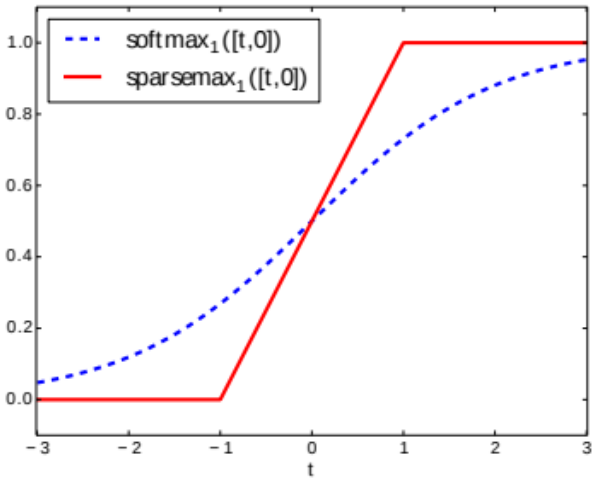


Fig.2.3 30 Funcția Sparsemax versus funcția Softmax [131]

În Fig.2.3.30 se poate vedea că Sparsemax este o funcție liniară, dar asimptotică fiind foarte asemănătoare cu funcția de activare Softmax. Sparsemax întoarce proiecția euclidiană a vectorului de intrare z pe probabilitatea simplex. Această proiecție intersectează limita simplexului, caz în care $sparsemax(z)$ devine aleatorie. Sparsemax are toate proprietățile funcției Softmax și are în plus proprietatea de producere aleatorie de distribuții.

Funcția de activare **Dropmax** [132] apărută în 2018, este o versiune stocastică a clasificatorului Softmax care la fiecare iterație scade clasele non-țintă în funcție de probabilitatea decisă pentru fiecare instanță. Mai exact, se suprapun măști binare ale variabilelor peste probabilitățile de ieșire ale clasei, care sunt învățate în mod adaptiv la intrare prin varianța inferenței. Această regularizare stocastică are ca efect construirea unui ansamblu clasificator în afara multor clasificatori exponențiali

cu limite de decizie diferite. Mai mult decât atât, învățarea ratelor dropout pentru clasele care nu sunt țintă pentru fiecare instanță permite clasificatorului să se concentreze mai mult asupra clasificării claselor țintă. La fiecare etapă are loc o scădere stocastică a gradientului în antrenarea rețelei, aceasta permite clasificatorului Dropmax să fie învățat să rezolve o problemă distinctă în identificarea problemei de clasificare de tip multi-clase, focusându-se pe proprietățile clasei țintă. În cele din urmă, când antrenarea s-a încheiat, se obțin un ansamblu de clasificatori exponențiali cu limite de decizie diferite.

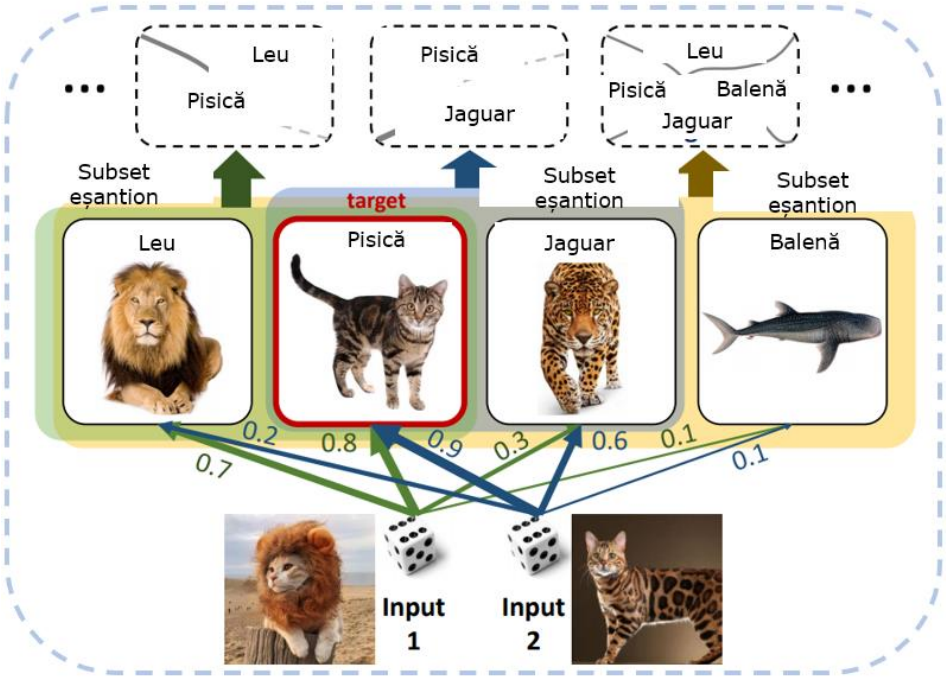


Fig.2.3 31 Funcția de activare Dropmax

În Fig.2.3.31 se poate observa că pentru o anumită instanță clasele sunt eșantionate la întâmplare cu probabilitățile învățate adaptativ pentru fiecare instanță. Apoi subsetul eșantionat participă la clasificare. De asemenea, în urma rezultatelor experimentale s-a constatat că are o viteză de convergență mare fără să necesite costuri suplimentare.

Funcția de activare **Softsign** [133] [63] este un alt tip de funcție de activare care este utilizată în calculul rețelelor neuronale. Funcția Softsign a fost introdusă în 2009, dar această funcție de activare a fost introdusă inițial în 1993 sub denumirea de ElliotSig [106], Softsign fiind un alt tip de funcție de activare neliniară utilizată în învățărea rețelelor neuronale profunde. Funcția Softsign este un pătrat polinomial, dat relația:

$$f(x) = \frac{x}{(|x|+1)} \tag{2.3.59}$$

Unde $|x|$ reprezintă valoarea absolută a intrării.

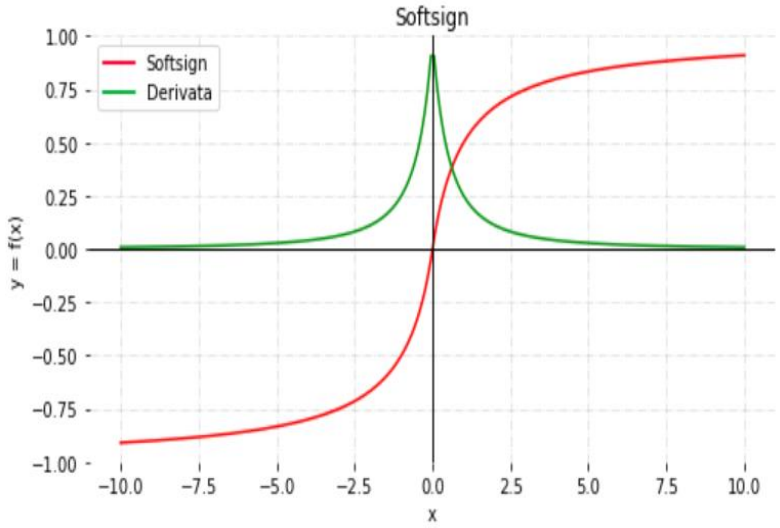


Fig.2.3 32 Funcția de activare Softsign și derivata sa

Principala diferență dintre funcția Softsign și funcția *tanh* constă în forma convergenței funcției, în cazul funcției Softsign fiind polinomială spre deosebire de funcția *tanh* care converge exponențial. Derivata funcției Softsign este dată de relația:

$$f'(x) = \frac{1}{(|x|+1)^2} \tag{2.3.60}$$

Funcția de activare Softsign este mărginită în intervalul finit (-1,1). Atunci când intervalul funcției de activare este finit, metodele de formare pe bază de gradient tind să fie mai stabile, deoarece prezența tiparelor ale modelului afectează semnificativ doar ponderile limitate. Atunci când intervalul este infinit, antrenarea este, în general, mai eficientă, deoarece prezența tiparelor ale modelului afectează semnificativ majoritatea ponderilor. În ultimul caz, sunt de obicei necesare rate de învățare mai mici. Funcția de activare Softsign este o funcție monotonă adică suprafața de eroare asociată cu un model cu un singur strat este garantată a fi convexă. [134] Softsign este o funcție derivabilă. Această proprietate este foarte dorită, pentru că de exemplu ReLU nu este diferențiabilă continuu și are unele probleme cu optimizarea bazată pe gradient, fiind derivabilă se permite optimizarea fără probleme folosind metoda gradientului. [135] De asemenea, are proprietatea de aproximare a identității în apropierea originii [136], proprietate care asigură rețelei neuronale să învețe eficient atunci când ponderile sale sunt inițializate cu valori aleatorii mici. Când însă funcția de activare nu are această proprietate, trebuie să se facă atent inițializarea ponderilor.

Funcțiile de activare **KAFs** [137] apărute în 2017, sunt o familie de funcții de activare flexibile care se bazează pe o expansiune a nucleului fiecărui neuron. Au plecat de la multe proprietăți bazate pe modele kernel, autorii propun variante multiple pentru proiectarea și inițializarea acestor funcții de activare a kernel-ului pe care le numesc *KAFs*, inclusiv o schemă multidimensională care permite combinarea informațiilor neliniare de pe diferite părți din rețea. Rețelele neuronale care folosesc aceste funcții de activare se numesc *KAFNETS*. Aceste funcții de activare sunt netede

pe întreg domeniul, liniare în parametri lor și pot fi regularizate folosind orice metodă, de exemplu, poate fi folosită penalitatea l_1 pentru a forța comportamentul aleatoriu.

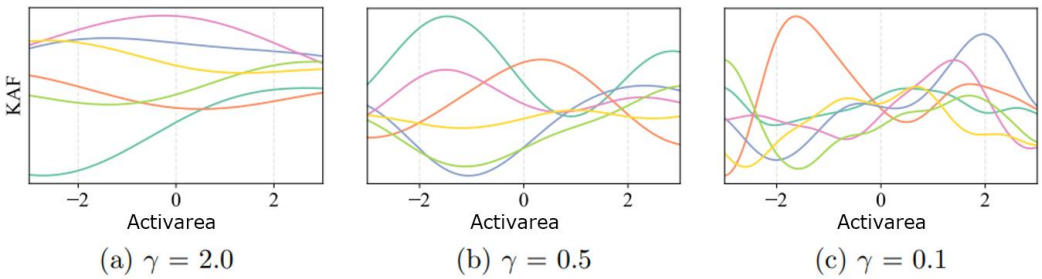
Aceste funcții de activare sunt non-parametrice la fel ca funcțiile Maxout și APL. Spre deosebire de abordările parametrice, funcțiile non-parametrice de activare permit modelarea unei clase mai mari de forme dar cu minusul unui număr mai mare de parametri adaptavi. Funcțiile de activare KAFs sunt definite astfel:

$$g(x) = \sum_{i=1}^{D} \alpha_i k(s, d_i) \tag{2.3.61}$$

Unde $\{\alpha_i\}_{i=1}^{D}$sunt coeficienții mixti, iar $\{d_i\}_{i=1}^{D}$ reprezintă dicționarul elementelor și $k(\cdot,\ \cdot): \mathcal{R} \to \mathcal{R}$ este o funcție kernel 1D (unidimensională).

$$k(s,\ d_i) = \exp\{-\gamma(s - d_i)^2\} \tag{2.3.62}$$

Unde $\gamma \epsilon \boldsymbol{\mathcal{R}}$ reprezintă lățimea de bandă a kernel-ului Gaussian.



(a) $\gamma = 2.0$ (b) $\gamma = 0.5$ (c) $\gamma = 0.1$

Fig.2.3 33 Funcțiile de activare KAFs cu lățimi de bandă diferite [137]

Cele două funcții de activare propuse KAF și 2D-KAF sunt funcții netede, pot folosi regularizare, de remarcat că au ca hiper-parametru dimensiunea dicționarului *D* și ponderi învățabile.

Goyal et al., propun în 2019, un nou tip de funcții de activare numite „***Funcții de activare auto-învățabile***" (**S**elf-**L**earnable **A**ctivation **F**unctions - SLAF), care sunt învățate în timpul antrenării și sunt capabile să se aproprie de majoritatea funcțiilor de activare existente. SLAF se calculează ca o sumă ponderată a unei baze predefinite de elemente care pot servi pentru o aproximare bună a funcției de activare optime. Autorii propun și diverse rutine de antrenare care pot fi utilizate pentru a obține performanțe cu rețele neuronale artficiale care au SLAF pe care le numesc *SLNN* (*SLAF equipped neural networks*). [138]

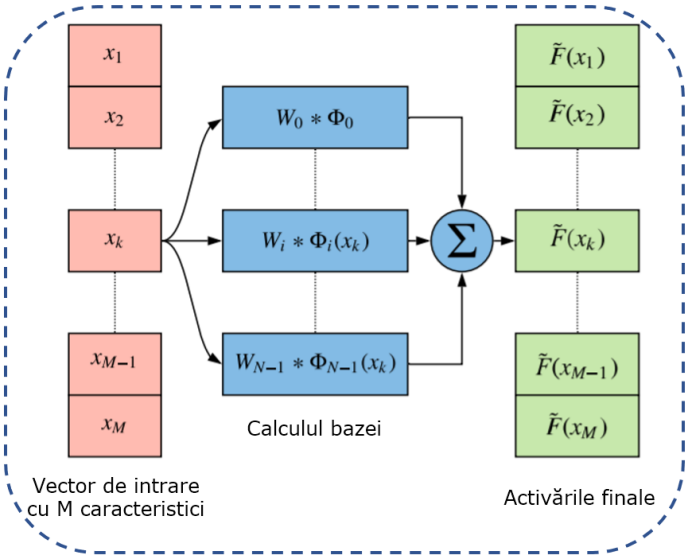


Fig.2.3 34 Funcțiile de activare SLAF [138]

Fig.2.3.34 prezintă un strat ascuns activat cu SLAF folosind M elemente de bază, unde $W_i s$ reprezintă parametri învățabili. Pentru a se demonstra eficiența SLNN-urilor și pentru a măsura performanța acesteia cu algoritmul BP, s-au efectuat experimente pe sarcini de regresie, clasificări și învățarea polinoamelor rare. În toate experimentele s-au aplicat regularizări L_2 pe coeficienții de activare. S-a aplicat regresie pe setul de date *Boston Housing* [139], clasificare pe *Two Spiral (un set de date care* conține un total de 194 de puncte*)* [140]. S-au înlocuit toate funcțiile de activare ReLU cu funcția de activare SLAF și acest model s-a numit SLAF cu rețele neuronale (*SLAF equipped neural networks - SLNN)*.

De asemenea, datorită faptului că funcțiile SLAF conțin polinoame iar polinoamele au termeni de grad mai mare, ceea ce face ca intrarea să crească la o rată foarte mare. Acest lucru are drept rezultat activarea explozivă a datelor de test, efect care este minimizat prin aplicarea regularizării L_2 aplicată ponderilor rețelei.

Funcția de activare **Sinusoid** [141] apărută în 2014, a fost folosită pentru antrenarea rețelelor neuronale profunde pentru date de tip serii de timp. Pentru inițializarea ponderilor s-au folosit transformări Fourier și s-a observat o îmbunătățire a generalizării când s-a folosit regularizare la antrenament. Abordarea lor folosește un mix de funcții de activare și antrenează rețeaua neuronală cu SGD. Aceste funcții de activare sunt:

- Sinusoid dat de relația:

$$f(x) = \sin(x) \tag{2.3.63}$$

- Funcția de activare Softplus dată de relația:

$$f(x) = log_e(1 + e^x) \tag{2.3.64}$$

- Funcția de activare identitate dată de relația:

$$f(x) = x \tag{2.3.65}$$

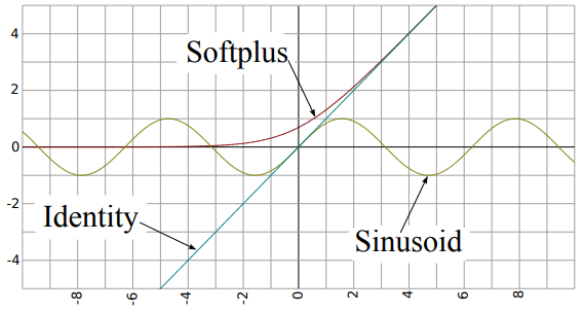


Fig.2.3 35 Funcțiile de activare Sinusoid, Softplus și Identitate

Pentru a ajunge la o generalizare mai bună, metoda propusă folosește mai multe unități cu activări simple care funcționează în rețea și apoi se antrenează modelul astfel încât să permită ponderilor să se orienteze spre aceste activări până când se observă un comportament bun al antrenării raportat la date. O metodă dinamică simplă de reglare a parametrilor se folosește pentru antrenarea eficientă a modelului. De asemenea, s-a observat că reglarea dinamică poate fi o soluție eficientă a problemelor de instabilitate care apar în mod inerent când folosim funcțiile de activare sinusoidale.

Funcția de activare **PELU** (**P**arametric **E**xponential **L**inear **U**nits) apărută în 2017, este o funcție de activare care generealizează ELU prin faptul că permite parametrului α să fie învățabil în mod dinamic în timpul antrenamentului. Această funcție de activare are de fapt doi parametrii, și anume α care este asimptota negativă, și β care asigură ajustarea pantei a ieșirilor pentru valorile pozitive. Atât α cât și β sunt parametri învățabili. [142]

Funcția de activare PELU este dată de relația:

$$f(x) = \begin{cases} \gamma x & dacă\ x \geq 0 \\ \alpha\left(e^{\frac{x}{\beta}}\right) & dacă\ x < 0 \end{cases} \tag{2.3.66}$$

Unde α, β, γ >0, iar $\gamma = \frac{\alpha}{\beta}$. Funcția de activare ELU poate fi obținută prin $\alpha = \beta = \gamma = 1$. Parametrul γ controlează panta în cadranul pozitiv (γ mai mare, cu atât panta devine mai abruptă), parametrul β afectează scala de descompunere exponențială (β mai mare, cu cât este mai mică coborârea), în timp ce α acționează asupra punctului de saturație din cadranul negativ (cu cât α este mai mare, cu atât saturația este mai mică). PELU are doar doi parametri învățabili α și β , parametru γ obținându-se pe baza celor doi parametri. Limitarea parametrilor în cadranul pozitiv obligă funcția de activare să fie o funcție monotonă, astfel încât reducerea valorilor ponderilor în timpul antrenamentului conduce la o scădere a contribuției neuronilor. Folosind această parametrizare, rețeaua își poate controla comportamentul neliniar pe parcursul antrenării. PELU a fost testată pe sarcini din domeniul Vederii Artificiale și aceasta dă rezultate mai bune ca funcția de activare ELU și totodată dă o curbă mai stabilă a erorii de-a lungul întregului proces de antrenare, spre deosebire de funcția ELU ne-parametrizată. Totodată autorii au analizat impactul efectului utilizării normalizării lotului (BN) înainte de funcția de activare PELU și arată că BN crește rata de eroare în cazul arhitecturii ResNet.

Funcția de activare **BLU** (**B**endable **L**inear **U**nit) [143] apărută în 2019, este o funcție de activare parametrică. Această funcție de activare are doi parametri: α și β, unde $0 \leq \alpha$ și $\beta \leq 1$.

Funcția de activare BLU este dată de relația:

$$f(\alpha,\beta,x) = \beta\left(\sqrt{x^2+\alpha^2+\epsilon}-\alpha\right)+x \tag{2.3.67}$$

Unde α controlează claritatea funcției, printr-o interpolare a funcțiilor de activare LReLU [86] și Softplus. [40]

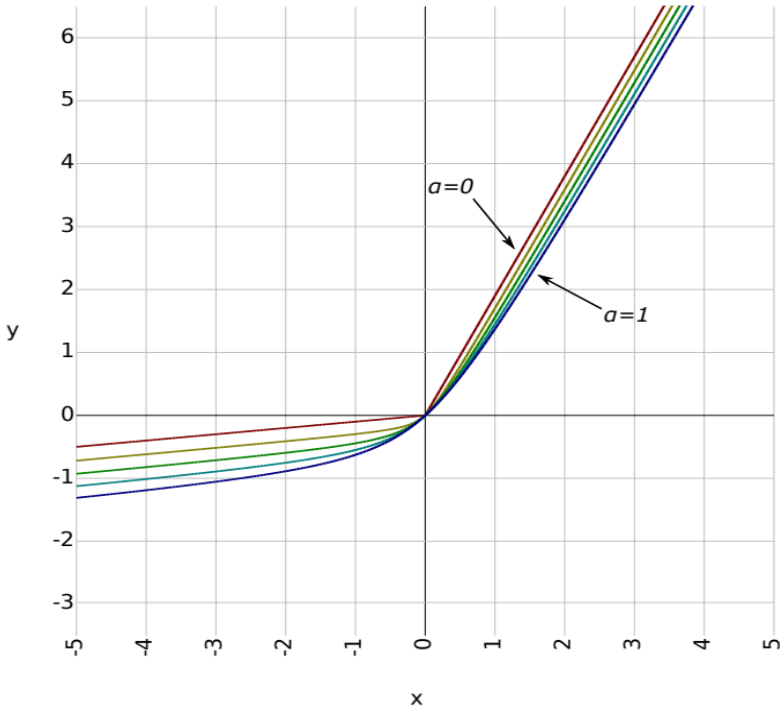


Fig.2.3 36 Funcția de activare BLU ($\alpha = \beta = 0.9$) [143]

Când $\alpha = 0$ funcția de activare devine ascuțită ca LReLU, iar când $0 < \alpha \leq 1$ BLU este netedă ca funcția Softplus. Prin adăgirea valorii mici notate cu ϵ se evită discontinuitatea din derivată pentru cazul când $x = \alpha = 0$. β controlează forma funcției, prin interpolarea dintre funcția identitate și funcția ReLU. Când $\beta = 0$ funcția de activare devine funcția identitate, iar când $0 < \beta \leq 1$ funcția produce îndoire neliniară.

Derivatele funcției de activare BLU sunt date de relațiile:

$$\frac{\partial y}{\partial x} = \frac{\beta x}{\sqrt{x^2+\alpha^2+\epsilon}}+1 \tag{2.3.68}$$

$$\frac{\partial y}{\partial \alpha} = \beta\left(\frac{\alpha}{\sqrt{x^2+\alpha^2+\epsilon}}-1\right) \tag{2.3.69}$$

$$\frac{\partial y}{\partial \beta} = \sqrt{x^2+\alpha^2+\epsilon}-\alpha \tag{2.3.70}$$

Unul din avantajele majore ale acestei funcții de activare este că are în componență funcția identitate, care are proprietățiile:

- nu are problema exploziei gradienților,
- straturile care nu sunt necesare pentru reprezentarea datelor în antrenarea rețelei neuronale profunde pot fi „omise” prin folosirea funcției identitate.

Funcția de activare **SL-ReLU** apărută în 2020 [144], este o funcție de activare pe porțiuni liniare validată pe seturi de date pentru o sarcină de recunoaștere a expresiei faciale: JAFFE [145] și FER2013 [146] . JAFFE este o bază de date publicată în 1998 și relativ mică, include 213 de imagini produse de 10 femei japoneze și fiecare persoană are șapte imagini care exprimă emoțiile: dezgust, furie, frică, fericire, tristețe, surprins și o stare neutră.

Facial Expression Recognition 2013 (FER-2013) conține 35,887 de imagini diferite, imagini la fețe de dimensiune 48 × 48 × 1 pixeli gri, unde 48 dă lațimea și înalțimea, iar 1 este canalul care fiind imagine gri, această valoare va 1.

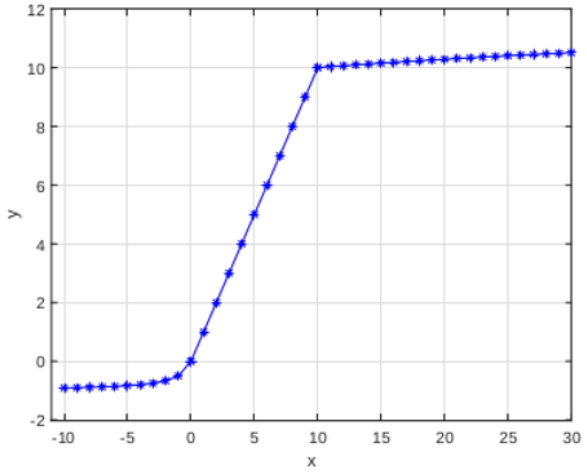


Fig.2.3 37 Funcția de activare SL-ReLU [144]

Funcția de activare SL-ReLU este dată de relația:

$$f(x)=\begin{cases}\frac{x}{1+|x|} & dacă\ x\le 0\\ \max(0,x) & dacă\ k\ge x>0\\ \log(a\cdot x+1)+|\log(a\cdot k+1)-k| & dacă\ x>k\end{cases} \tag{2.3.71}$$

Se poate observa din relația (2.3.71) că această funcție de activare este compusă din mai multe funcții: funcția Softsign, ReLU și funcția log, astfel că păstrează unele dintre caracteristicile funcției ReLU, dar folosește și proprietăți ale funcției Softsign. Această funcției evită problema supraspecializării în timpul antrenării și de asemenea are capacitatea să reducă problema oscilațiilor.

Funcțiile de activare **DP ReLU** și **Dual Line** [147] sunt propuse în 2019, în vederea explorării conceptului de învățare dinamică a pantei și deplasarea medie a parametrilor pentru modificarea pantelor funcției de activare în vederea îmbunătățirii regiunii totale de răspuns. Funcția de activare a liniei duale (Dual Line) poate fi privită ca o combinație a funcței de activare DP ReLU cu parametrul de schimbare a

capacității medii care este învățabil. Performanța mai bună a funcției Dual Line în comparație cu DP ReLU indică clar impactul parametrului învățabil în faza de antrenare a rețelei neuronale artificiale.

Funcția de activare DP ReLU este dată de relația:

$$f(x) = \begin{cases} \alpha \cdot x, & dacă\ x < 0 \\ \beta \cdot x, & dacă\ x > 0 \end{cases} \tag{2.3.72}$$

Din ecuația (2.3.72) se poate vedea că diferența dintre Parametric ReLU (PReLU) și DP ReLU este utilizarea parametrului învățabil care controlează panta pe axa pozitivă. Parametrul pantei α pe axa negativă este inițializat cu valoarea 0,01 devine Leaky ReLU. Parametrul pantei β pe axa pozitivă este inițializat cu valoarea 1.

Funcția de activare Dual Line este dată de relația:

$$f(x) = \begin{cases} \alpha \cdot x + m, & dacă\ x < 0 \\ \beta \cdot x + m, & dacă\ x > 0 \end{cases} \tag{2.3.73}$$

Din ecuația (2.3.73) se poate vedea că funcția de activare Dual Line este o extensie a funcției de activare DP ReLU, la care se adaugă un parametru m care definește deplasarea medie. Parametrul mediu este inițializat cu o valoare de -0,22 prin adăugarea deplasării medii (-0.25) și a parametrului de prag (0.03) folosiți în Threshold Relu (TReLU) [148]. Funcția de activare ReLU care efectuează o operație de prag pentru fiecare element de intrare în care valorile mai mici de zero sunt setate zero, în cazul funcției TReLU putem considera acest 0 un parametru T care poate controla această valoare de prag.

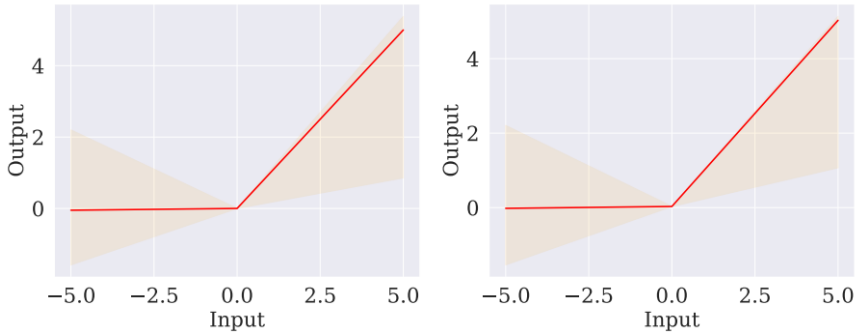


Fig.2.3 38 Funcția de activare DP ReLU (stânga) și funcția de activare Dual Line (dreapta) pe arhitectura WideResNet pe setul de date CIFAR-10 [149]

Câteva proprietăți semnificative ale acestor funcții de activare sunt:

- Ambii parametri ai pantei sunt independenți de alți parametri și acționează direct pe intrare fără nicio restricție.
- Regiunea de răspuns e mai mare decât fără parametri învățabili.
- Deplasarea medie în Dual Line ajută rețeaua să pună activarea medie aproape de 0.
- Forma acestor funcții nu afectează complexitatea timpului de calcul al rețelei, singurul minus îl constituie parametri învățabili în plus.

Funcția de activare **Hard *tanh*** (*Hard Hyperbolic*) apărută în 2016, este o variantă a funcției de activare *tanh* folosită în Învățarea Profundă, dar mai eficientă

din punct de vedere computațional. Hard *tanh* este în intervalul (-1,1) și are ecuația dată de relația:

$$f(x) = \begin{cases} -1, & dacă\ x < -1 \\ x, & dacă\ -1 < x \leq 1 \\ 1, & dacă\ x > 1 \end{cases} \quad (2.3.74)$$

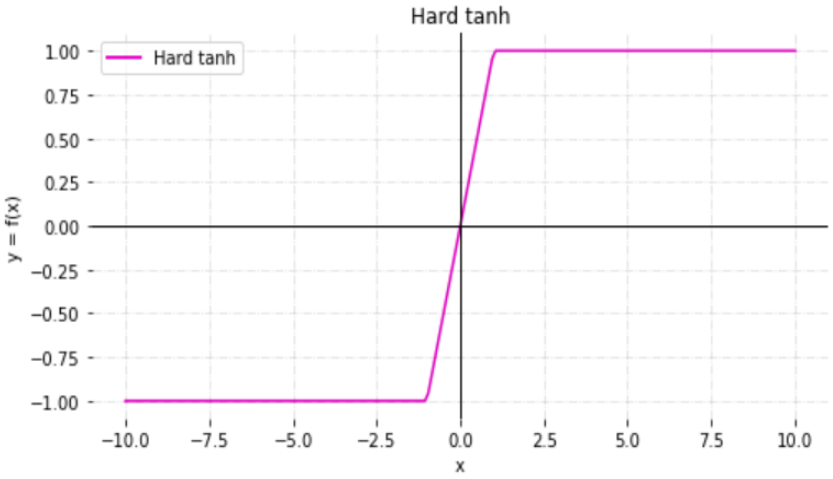


Fig.2.3 39 Funcția de activare Hard *tanh*

Această funcție de activare a dat rezultate foarte bune pe sarcini de Procesare a Limbajului Natural [72]. Funcția de activare **Hard sigmoid** [150] după cum îi spune numele derivă din funcția sigmoidă mai puțin costisitoare din punct de vedere al calculului atât în aplicații software, cât și în implementări hardware specializate.
Hard sigmoid are ecuația dată de relația:

$$f(x) = max\left(0,\ \min\left(1,\ \frac{(x+1)}{2}\right)\right) \quad (2.3.75)$$

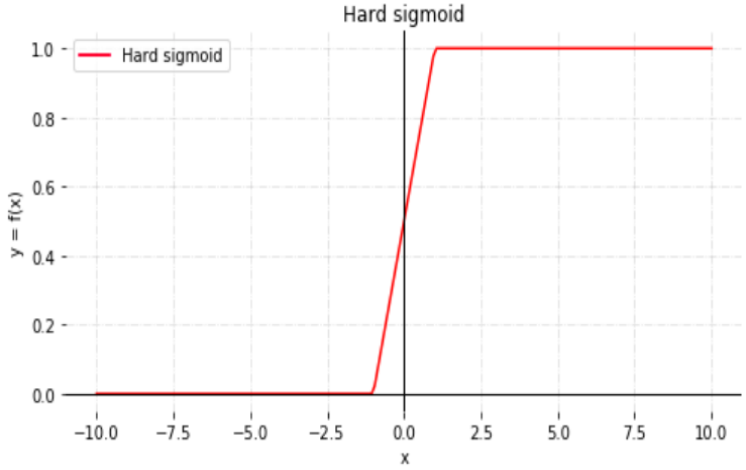


Fig.2.3 40 Funcția de activare Hard sigmoid

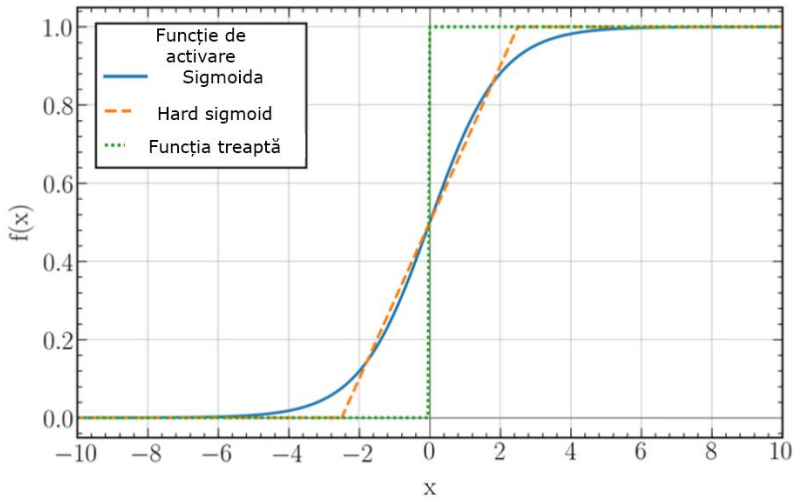


Fig.2.3 41 Funcția Hard sigmoid versus funcțiile Sigmoidă și treaptă [151]

Funcția de activare **FReLU** (***F**lexible **Re**ctified **L**inear **U**nit*) [152] apărută în 2017, este un redesign a funcției ReLU prin adăugarea unui parametru învățabil. FReLU tinde să conveargă la o valoare negativă, care îmbunătățește expresivitatea și astfel și performanța rețelei neuronale.

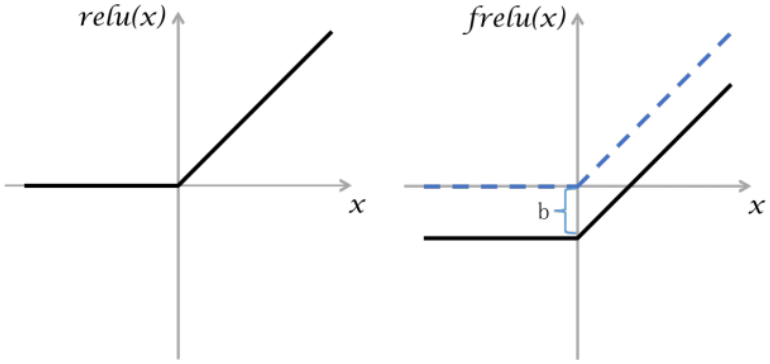


Fig.2.3 42 Funcția ReLU versus funcția FReLU [152]

Funcția de activare FReLU este dată de relația:

$$frelu(x) = relu(x + a) + b \tag{2.3.76}$$

Unde a și b sunt doi parametri învățabili.
Avantajele funcției FReLU sunt:

- convergență rapidă și performanțe superioare.
- calcul computațional redus datorită lipsei de operații exponențiale.
- compatibilitatea cu normalizarea lotului.

Funcția de activare numită **Snake** [153] apărută în 2020, se poate mări cu un factor *a* pentru a controla frecvența părții periodice. Astfel se propune funcția de activare Snake cu frecvența a astfel:

$$Snake_a := x + \frac{1}{a} sin^2(ax) = x - \frac{1}{2}\cos(2x) + \frac{1}{2} \tag{2.3.77}$$

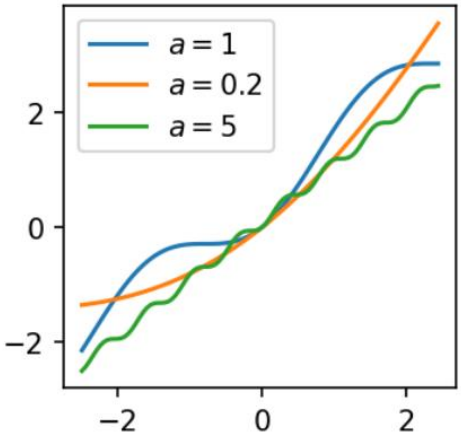


Fig.2.3 43 Funcția de activare Snake pentru diferite valori ale parametrului a

Pentru $a = 1$ ecuația (2.3.77) devine:

$$Snake_{a=1} := x + sin^2(x) \tag{2.3.78}$$

S-a remarcat că această funcție este mai ușor de optimizat decât funcția sin și valoarea maximă calculată folosind distribuția normală standard este ≈ 0.56045. Validitatea acestei funcții este dată de rezolvarea problemei periodicității, problemă abordată prea puțin în literatura de specialiate, se face folosind „teorema extrapolării”.

2.4. Algoritmul de propagare înapoi

Algoritmul de propagare înapoi (*Backpropagation-BP*) este cel mai frecvent algoritm utilizat pentru a antrena rețelele neuronale, indiferent de natura setului de date utilizat. Arhitectura rețelei neuronale este determinată de încercări repetate, scopul este obținerea celei mai bune clasificări posibile a setului de date utilizat în acest context. [154] [155] BP este un algoritm de învățare supervizat și scopul său este de a minimiza eroarea. BP compară valoarea de ieșire calculată cu valoarea reală și încearcă să schimbe ponderile prin eroarea calculată și, în mod analog, până când dimensiunea erorii obținute devine mai mică decât eroarea obținută în prima iterație. Componența rețelei de de propagare înapoi este detaliată în figura următoare:

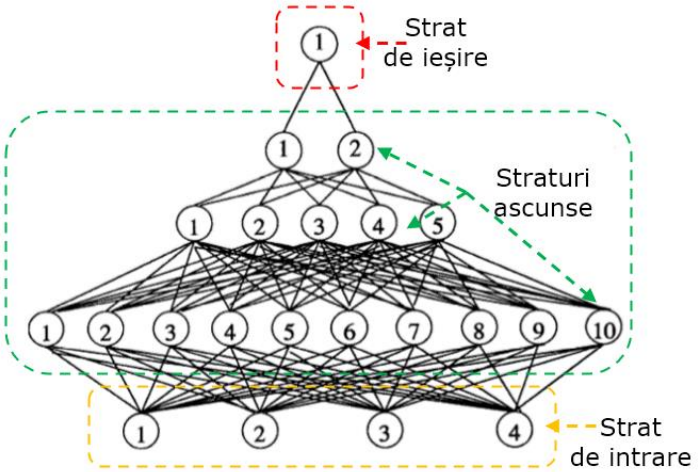


Fig.2.4. 1 Rețea neuronală artificială cu 3 straturi ascunse [155]

Ca strategie de învățare, algoritmul de propagare înapoi s-a dovedit a fi eficient prin faptul că asigură o clasificare a cărei precizie este în general satisfăcătoare. [156] [157]

Principalul dezavantaj al acestei tehnici de învățare este faptul că presupune încercări repetate pentru a stabili arhitectura rețelei, numărul de straturi ascunse și numărul de neuroni în fiecare strat ascuns, în contextul în care antrenare necesită o mulțime de resurse precum memoria și timp de rulare, ca în figura următoare:

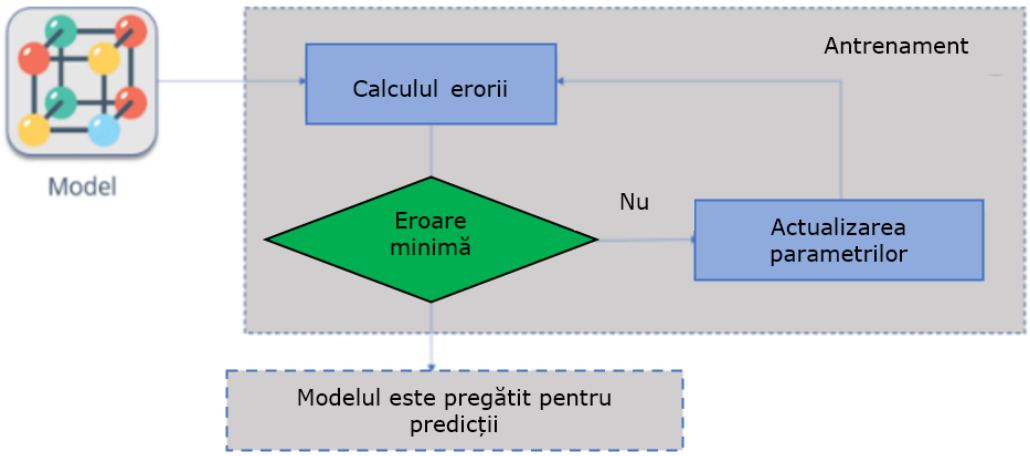


Fig.2.4. 2 Algoritmul de propagare înapoi

Algoritmul clasic de propagare înapoi este împărțit în două faze: prima fază constă în propagarea informațiilor utile din stratul de intrare spre stratul de ieșire, apoi răspândit în direcția inversă a erorilor, iar a doua fază constă în actualizarea ponderilor. Algoritmul BP s-a concentrat pe teoreme care au garantat „aproximarea universală”, o proprietate a rețelelor neuronale. [65]

2.5. Concluzii

Diversitatea funcțiilor de activare și extinderea continuă a acestora în vederea îmbunătățirii performanței în cadrul unei rețele neuronale profunde m-a determinat să acord acestei arii atât de studiate o atenție deosebită în vederea conceperii, prin această teză, a unor noi funcții superioare din punct de vedere al performanței pe care o aduc în cadrul rețelei. Astfel, am ales diferite tipuri de arhitecturi de rețelele neuronale atât tradiționale precum CNN dar și arhitecturi de ultimă generație (AlexNet [26], MobileNet [158], Xception [159], *ResNet* [33], rețele neuronale folosind blocuri (*Visual Geometry Group - VGG)* [31]). De asemenea, pentru aceste funcții de activare am extins studiul la mai multe seturi de date pentru a vedea cum se comportă funcțiile propuse.

Primul pas în definirea unor noi funcții de activare a fost identificarea funcțiilor deja existente care să dețină proprietăți importante și să ne ofere flexibilitatea să le modelăm, studiu realizat în detaliu în cadrul acestui capitol, din care am observat că funcția ReLU este o soluție robustă pentru o arie mare de sarcini și seturi de date, dar are și câteva minusuri precum ReLU “mort”, dezvoltându-se alte funcții care să acopere acest minus: PReLU, LReLU. De asemenea s-au văzut două funcții de activare care conduc la rezultate foarte bune și anume Swish și Mish.

S-a văzut că funcția tradițională sigmoidă este recomandat să se evite în cadrul rețelelor atunci când inițializarea ponderilor se face cu valori mici. [63] Funcția sigmoidă suferă și de alte dezavantaje majore care includ gradienți abrupți în timpul propagării înapoi de la straturile ascunse mai profunde către straturile de intrare,

probleme de saturație a gradienților, convergența lentă și ieșirea centrată non-zero care implică astfel actualizările gradientului de propagare în diferite direcții. De la această propunere s-a văzut că au plecat alte propuneri, de exemplu, funcția tangentă hiperbolică (*tanh*), care remediază unele dintre aceste dezavantaje suferite de funcția de activare sigmoidă. [9] Din graficul funcției tanh s-a putut vedea că este o funcție centrată în 0 și ieșirea poate lua valori între -1 și 1, în cazul sigmoidei am văzut că ieșirea ia valori doar între 0 și 1. Sigmoida s-a observat că atunci când este utilizată ca activare finală într-un clasificator, suma tuturor claselor nu este egală 1, cum ar fi firesc. Atât sigmoida cât și tanh necesită calcul exponențial. Datorită restricțiilor mari pe care aceste funcții le-au prezentat, s-a observat o ascensiune continuă în această direcție, dezvoltându-se funcții diverse cu avantaje și dezavantaje.

ReLU rămâne o funcție folosită pe scară largă datorită modului facil de folosire și a capacității ridicate de generalizare, această funcție se reprezintă aproape liniar și, prin urmare, păstrează proprietățile modelelor liniare care i-au permis o optimizare mai facilă, cu scăderea gradientului. [2] Fiind o funcție de prag și faptul că ignoră partea negativă, au apărut funcțiile PReLU și LReLU care nu fac altceva decât să ia în considerare și valori negative, corespunzătoare cadranului III, venind astfel ca o soluție pentru ReLU " mort" datorită lipsei acestei flexibilități privind valorile negative. Dar și aceste abordări aduc în plus un parametru predefinit leaky în cazul funcției de activare LReLU sau un parametru învățabil în cazul funcție de activare PReLU.

Ulterior a apărut funcția ELU care este o bună alternativă pentru ReLU, deoarece accelerează învățarea în rețelele profunde și conduce la precizii de clasificare superioare, dar și această funcție are un parametru în plus și mai mult de atât am văzut că presupune la fel ca sigmoida și tanh, calcul exponențial. Ca și funcțiile ReLU, LReLU și PReLU, ELU reduce problema dispariției gradientului prin identitatea valorilor pozitive. Deci poate fi o variantă de funcție atunci când ne confruntăm cu o astfel de problemă în cadrul rețelei.

Atunci când ne dorim să avem o învățare extrem de robustă se recomandă SELU, funcție care introduce proprietăți de auto-normalizare și permite antrenarea rețelelor profunde cu multe straturi folosind o regularizare puternică.

GELU este și ea o funcție performantă care datorită regulatorului stocastic pe care îl folosește pentru a introduce non-liniariate poate fi folosită cu succes având aceleași efecte aduse prin tehnica dropout, dar elimină nevoia unei non-liniarități tradiționale.

Cu toate că au fost propuse diverse alternative ale funcției de activare ReLU, niciuna nu a reușit să o înlocuiască din cauza câștigurilor inconsistente. Swish este una din primele funcții de compoziție [3] care tinde să dea rezultate mai bune decât ReLU pe modele mai dense. La fel ca ReLU, Swish este mărginită în partea inferioară adică pe măsură ce x se apropie de $-\infty$, y se apropie de o anumită valoare constantă, dar nemărginită în partea superioară, ceea ce înseamnă că x se apropie de $+\infty$, y se apropie și el de $+\infty$. Cu toate acestea, spre deosebire de ReLU, Swish este netedă adică nu are modificări bruște sau un vârf. În plus, Swish este non-monotonă, ceea ce înseamnă că nu există întotdeauna o derivată pozitivă sau negativă unică și continuuă pe întreaga funcție. Funcția Swish are o derivată negativă în anumite puncte și o derivată pozitivă în alte puncte, în loc să aibă doar o derivată pozitivă în toate punctele cum au funcțiile Softplus și Sigmoida.

Având în vedere că majoritatea cercetătorilor folosesc metode ce implică fie funcții individuale (o singură funcție, de exemplu, sigmoida, *tanh*, Softplus), fie funcții compuse (Swish, Mish, E-Swish), aceste funcții sunt combinații de mai multe funcții

existente, am hotărât să folosesc atât funcții individuale cât și funcții compuse. Mai mult decât atât, am folosit inclusiv conceptul de funcții pe porțiuni (precum funcția APL), concept destul de puțin de studiat în Învățarea Profundă, care aduce o flexibilitate mai mare rețelei.

Atunci când ne dorim o funcție rapidă, se recomandă FTS, care are proprietăți similare cu ReLU și Swish, dar ce apare nou este un prag, introdus în scopul îmbunătățirii preciziei de clasificare într-o rețea de tip propagare înainte și s-a constat că această funcție de activare aduce o îmbunătățire majoră în ceea ce privește viteza antrenării rețelei, fiind de 2 ori mai rapidă decât funcția ReLU.

Pentru a avea o rețea profundă stabilă și performantă se recomandă Softplus care și ea este o versiune a funcției ReLU dar care are proprietăți de înclinare și gradient diferit de 0 (ReLU are gradient egal cu 0 pentru intrări negative). Totodată se consideră a fi o funcție învățabilă, crescătoare în două din argumentele sale și convexă pe unul dintre ele.

Am putut vedea o versiune de funcție netedă, non-monontonă, Mish care depășește din punct de vedere al performanței pe ReLU și Swish, fiind recomandată în rețele profunde pentru performanțele aduse.

ISRLU la fel ca și funcția SQNL sunt o bună opțiune atunci când se dorește să se evite supraspecializrea prin îndepărtarea zgomotului, datorită operatorului pătrat, conduc la performanțe de viteză comparabile cu ELU și de asemenea beneficiază, de multe proprietăți ale funcției ELU. Ambele funcții au valori negative, ceea ce le permite să aducă activarea medie a unității mai aproape de zero și să aproprie valorile gradienților către gradientul natural al unităților.

Funcția SC [109] a fost dezvoltată în vederea rezolvării minusului funcției ELU care nu poate produce valori exponențiale mari. Această funcție de tăiere ușoară este aproximativ liniară pentru $x \in (0, 1)$ și devine asimptotă foarte repede în afara acestui interval. Dar trebuie să ținem cont de parametrul în plus pe care îl aduce, parametru responsabil pentru cât de aproape de liniaritate este regiunea centrală și cât de brusc se transformă regiunea liniară către valori asimptotice.

Atunci când dorim o funcție fără restricții asupra parametrilor săi învățabili, se recomandă SReLU care poate învăța atât funcții convexe cât și non-convexe, astfel rețeaua neuronală profundă cu SReLU va avea o capacitate de învățare a caracteristicilor mai puternică.

Atunci când dorim să substituim straturile de normalizare a lotului se recomandă folosirea funcției BReLU, care împreună cu o inițializare corectă a ponderilor poate avea același efect asupra rețelei neuronale.

Pentru o caracterizare matematică a straturilor de convoluție în vederea proprietății de reconstrucție se recomandă CReLU, care este un indicator important pentru a vedea cât de expresive și generalizabile sunt caracteristicile rețelelor de convoluție.

Atunci când ne dorim păstrarea normalizată a gradienților înapoi, care reprezintă un potențial bun pentru formarea rețelelor neuronale profunde, foarte profunde, chiar și a celor recurente se recomandă funcția OPLU. În plus față de funcțiile de activare tradiționale care anulează proprietatea de ortogonalitate în cazul propagării înapoi, care conduce implicit la o anihilare a păstrării gradienților normalizați, lucru însă care e asigurat de OPLU.

APL se recomandă atunci când este necesară învățarea independentă pentru fiecare neuron, folosind GD. S-a observat că atât unitățile Maxout, cât și rețeaua nu pot învăța orice funcție de activare liniară pe când unitățile APL o pot face, dar cu nevoia unui plus de parametri.

Pentru stratul final se recomandă Softmax datorită interpretării sale probabilistice.

Funcția L-Softmax este o bună alegere atunci când se dorește în mod explicit compactitatea intra-clasă și separabilitatea inter-clase între caracteristicile învățate.

Atunci când scopul urmărit constă în obținerea unor ieșiri sub forma unor probabiliăți aleatorii se recomandă SparseMax [131].

În vederea descalificării claselor non-țintă se recomandă funcția Dropmax [132] care la fiecare iterație scade clasele non-țintă în funcție de probabilitatea decisă pentru fiecare instanță.

S-a putut vedea că funcția Softsign converge polinomial în timp ce funcția *tanh* converge exponențial.

În cazul în care rețeaua are nevoie de un set de funcții flexibile se recomandă funcțiile KAFs [137] care se bazează pe o expansiune a nucleului fiecărui neuron.

Un alt aspect important de care trebuie să se țină cont este dat de gradul polinoamelor, doarece funcțiile SLAF conțin polinoame iar polinoamele au termeni de grad mai mare, ceea ce face ca intrarea să crească la o rată foarte mare, conducând la activarea explozivă a datelor de test, efect care însă poate fi combătut prin aplicarea regularizării L_2.

În cazul unei probleme de varianță mare a oscilațiilor, se recondă SL-ReLU care evită problema supraspecializării în timpul antrenării și are capacitatea să reducă problema oscilațiilor.

În vederea extinderii conceptului de învătare dinamică a pantei și deplasarea medie a parametrilor pentru modificarea pantelor funcției de activare în vederea îmbunătățirii regiunii totale de răspuns într-o rețea, se recomandă funcțiile DP ReLU și Dual Line [147].

Pentru a îmbunătăți expresivitatea și performanța rețelei, se recomandă FReLU [152] (care adaugă un parametru învățabil) datorită faptului că FReLU tinde să conveargă la o valoare negativă.

Funcțiile de activare s-au dovedit a fi un factor determinant al performanței în rețelele neuronale profunde (numeroase funcții analizate în această teză) motiv pentru care se dezvoltă în permanență.

3. Tehnici de optimizare

3.1. Descendența gradientului

Algoritmul descendența gradientului (*Gradient Descent-GD*) atribuit matematicianului Cauchy, a fost menționat pentru prima oară în 1847 [160], dar proprietățile sale de convergență în problemele de optimizare au fost studiate ulterior în 1944 de Haskel Curry.

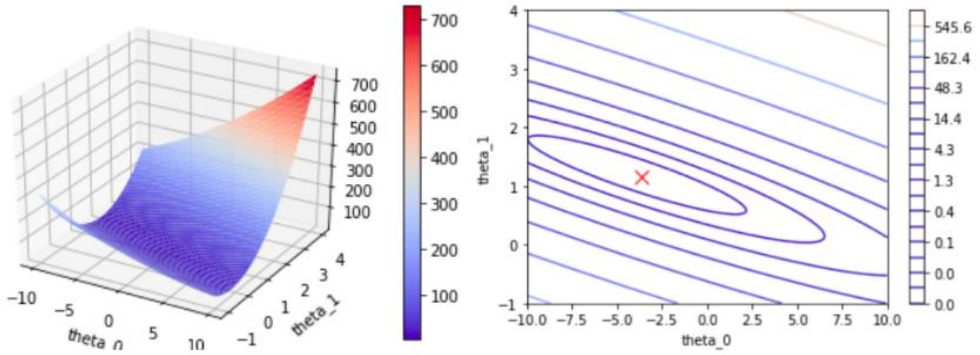


Fig.3.1. 1 Descendența gradientului

GD este un algoritm de optimizare iterativă de prim ordin pentru găsirea unui minim local a unei funcții derivabile. GD a devenit un algoritm cheie pentru optimizare oricărui model de Învățare Profundă, care efectiv presupune reducerea iterativă a erorii prin actualizarea parametrilor în direcția în care funcția de cost scade.

Pe o suprafață a funcției de cost convexă, acesta va converge la un minim global, caz care constituie cazul cel mai dorit, în caz opus acesta va conduce spre un minim local. Minusul acestei tehnici este dat de faptul că trebuie luată derivata funcției de cost adevărată, care este o medie a funcțiilor de cost calculate pentru fiecare exemplu în parte din setul de date. Acest lucru conduce la un consum mare de timp. Dar ca o soluție la această problemă vine algoritmul descendența stocastică a gradientului (Stochastic Gradient Descent - SGD).

Scopul este găsim ponderile în rețea neuronală care conduc la o minimizare a erorii de antrenare între etichetele adevărate și cele estimate ale unor exemple din antrenament.

Actualizarea ponderilor cu GD se face după relația:

$$w \leftarrow w - \alpha \frac{dE}{dw} \tag{3.1.1}$$

Unde E- eroarea se poate calcula astfel:

$$E(w) = \sum_{i=1}^{N} (y_i - f_w(x_i))^2 \tag{3.1.2}$$

Gradienții sunt calculați în direcția de la ieșire spre straturile de intrare folosind algoritmul BP împreună cu regula înlănțuirii.

3.2. Descendența stocastică a gradientului

Descendența stocastică a gradientului (*Stochastic Gradient Descent* - SGD), spre deoserbire de GD, consideră un lot mic aleatoriu de exemple de fiecare dată când avem nevoie să facem actualizările ponderilor.

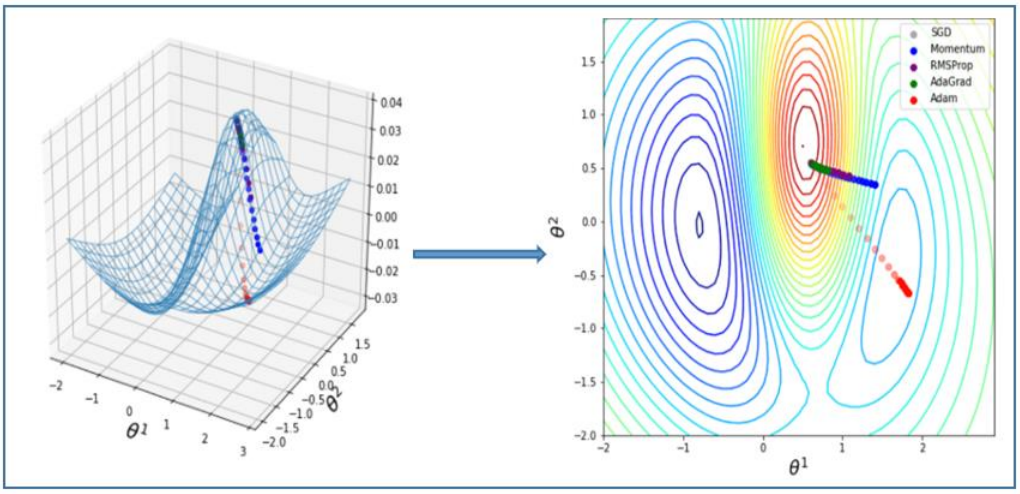


Fig.3.2. 1 Descendența stocastică a gradientului (https://medium.com/@thushv89/a-practical-guide-to-understanding-stochastic-gradient-descent-methods-workhorse-of-machine-50c9d94cb34)

În fiecare iterație, mai întâi luăm, în mod aleatoriu, un lot mic $\mathcal{B}$ care reprezintă un număr fix de exemple din setul de antrenament. Apoi calculăm derivata (gradientul) funcției de cost medie pe acel lot în ceea ce privește parametrii modelului. În cele din urmă, înmulțim gradientul cu dimensiunea unui pas, predefinită $\eta > 0$, η este rata de învățare și scădem termenul rezultat din valorile parametrului curent.
De obicei, acești parametri $|\mathcal{B}|$ și η sunt predefiniți manual și nu învățați în timpul antrenării. Acești parametri care pot fi reglați, dar nu sunt actualizați în bucla de antrenare, se numesc ***hiperparametri***. Reglarea hiper-parametrului este procesul prin care ajustarea hiper-parametrelor pe care o facem în funcție de rezultatele din faza de antrenare, în funcție de rezultatele în urma evaluării pe un set de validare separat a datelor. [59]

Pașii acestui algoritm sunt:

1. inițializarea ponderilor modelului (w) cu valori aleatorii.
2. Alegem iterativ loturi aleatorii din date, acest pas se face de multe ori, în fiecare iterație actualizăm parametri în direcția gradientului negativ.

După antrenament pentru un număr predefinit de iterații, care este întâlnit în literatura de specialitate ca un număr numit epoci, înregistrăm parametri modelului estimat. Chiar dacă avem cazul unei funcții liniare și fără zgomot, acești parametrii estimați nu vor fi valorile minime ale funcțiilor de cost, deoarece, deși algoritmul converge lent spre un minim local, nu îl poate atinge exact într-un număr finit de pași. Valorile obținute cu SGD prezintă mai mult zgomot decât cele cu GD tocmai din natura întâmplătoare a SGD, dar din punct de vedere computațional SGD este mai rapid. Problemele apar atunci când rata de învățare este prea mică sau prea mare. În practică, o rată de învățare adecvată este adesea găsită numai după mai multe

experimente. Când avem mai multe date în setul de date de antrenare, durează mai mult calcularea fiecăre iterații pentru a micșora gradienții, astfel că SGD este o soluție preferată în aceste cazuri în detrimentul GD. Optimizarea cu SGD nu este, în general, o soluție în cazurile non-convexe, deoarece numărul minimelor locale care trebuie verificate ar putea fi de ordin exponențial. Folosirea SGD este de obicei o soluție mult mai rapidă pentru a converge, decât dacă folosim metoda eșantionării aleatorie.

3.3. Evitarea supraspecializării

Un model care a învățat zgomotul în locul informațiilor relevante este considerat „supraspecializat" (*overfit*), deoarece se potrivește cu setul de date de antrenament, dar are o capacitate slabă pe seturi de date noi.

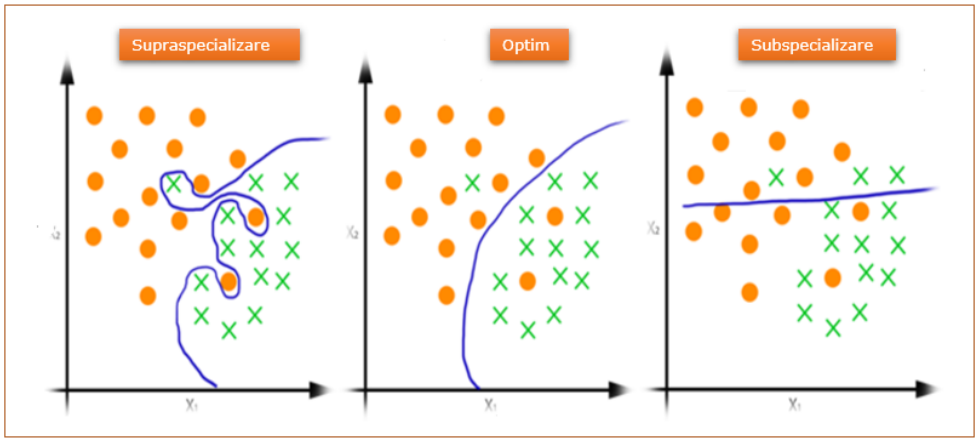


Fig.3.3. 1 Supraspecializare (https://miro.medium.com/max/1750/1*cdvfzvpkJkUudDEryFtCnA.png)

În Fig.3.3.1. se poate vedea în imaginea din stânga supraspecializarea este ilustrată cu linie albastră, asta înseamnă că a învățatat atât de bine pe aceste date de antrenament dar pe un set de date pe care nu l-a mai văzut niciodată nu va fi capabilă să extragă informațiile relevante pentru noi. La pol apus se află subspecializarea care apare atunci când un model este prea simplu, are prea puține caracteristici sau regularizat prea mult, ceea ce îl face să fie inflexibil în învățarea din setul de date.

O provocare esențială în ceea ce privește supraspecializarea, este că nu putem ști cât de bine va funcționa modelul nostru pe datele noi până în faza de testare. Pentru a rezolva acest lucru, putem împărți setul de date inițial în set de antrenare și set de test. [161] În cele ce urmează prezint câteva soluții pentru această problemă des întâlnită în Învățarea Profundă.

a) Validarea încrucișată a k-subseturi

Validarea încrucișată [162] [163] [164] este o tehnică de validare a modelului pentru evaluarea modului în care rezultatele unei analize statistice se vor generaliza într-un set de date independent. Aceasta este o tehnică foarte cunoscută în Învățarea Profundă care luptă împotriva supraspecializării. [165].

Practic, se folosesc datele de antrenare inițială pentru a genera mai multe grupuri de test mici. Apoi se folosesc aceste grupuri pentru a potrivi modelul.

Într-o serie de validare încrucișată împărțim un eșantion de date *k* în subseturi complementare, efectuarea analizei pe un subset numit set de antrenament și validarea analizei pe celălalt subset denumit set de validare sau set de test.

Validarea încruțisată a k-subseturi este un tip de validare încrucișată *k*l*-subseturi atunci când *l = k - 1*. Se folosește o validare încrucișată cu un singur *k* atât cu un set de validare cât și cu un set de test. Setul total de date este împărțit în *k* seturi. Fiecare fiind selectat progresiv ca ca set de test.

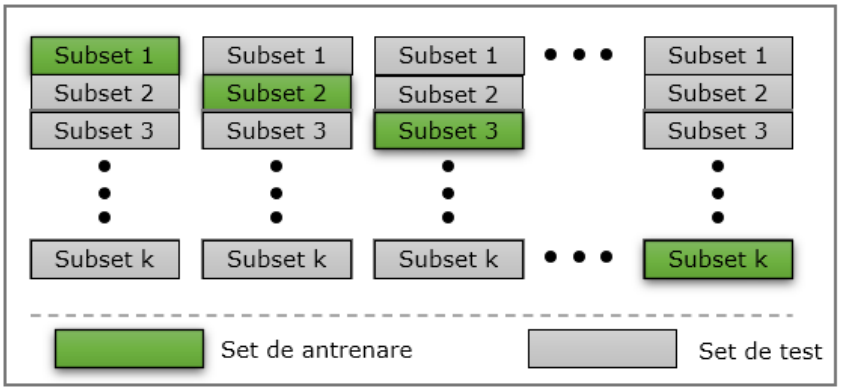


Fig.3.3. 2 Validarea încruțisată a k-subseturi [166]

b) Regularizarea

Regularizarea este procesul prin care se adaugă informații pentru a rezolva o problemă prezentată în mod eronat sau pentru a preveni efectul supraspecializării. [167]

De asemenea, există și cazul în care avem modificări mici ale valorilor de intrare, dar cu toate astea vedem fluctuații foarte mari în ieșiri. Aceeași problemă poate apărea în rețelele neuronale unde intrările sunt combinate liniar cu parametri pentru a transmite informațiile între straturi; când acești parametrii sunt foarte mari, determină modificări mici ale intrărilor. O soluție care vine ca o măsură pe care putem să o luăm împotriva acestei probleme constă penalizarea parametrilor din funcția de cost pentru a preveni ca parametri să devină excesiv de mari în timpul antrenării. Putem face acest lucru printr-o serie de expresii, cele două cele mai populare sunt normalizarea L_1 și normalizarea L_2. Un termen de regulizare notat cu $R(f)$ se adaugă la funcția de cost astfel:

$$\min_f \sum_{i=1}^{n} V(f(x_i), y_i) + \lambda R(f) \qquad (3.3.1)$$

Unde V este o funcție de cost de bază care descrie funcția de cost prezisă $f(x)$ atunci când eticheta este y, cum ar fi funcția de cost pătratică sau funcția de cost marginea-maximă, iar λ este un parametru care controlează importanța regulatorului, adică controlează gradul în care alegem să penalizăm parametrii mari. În general, $R(f)$ este ales astfel încât să penalizeze complexitatea lui f, care includ restricții pentru netezirea și limitele la normala spațială vectorială. [125] Regularizarea L_1 cunoscută și ca *regresie Lasso* [168], se folosește pentru a îmbunătăți exactitatea și interpretarea predicțiilor pe care le produce modelul. L_1este folosită de regulă pentru selecția caracteristicilor, deoarece tinde să producă vectori de parametri reduși, în cazul în care doar caracteristicile importante iau valori diferite de 0. Regularizarea L_2 introdusă de matematicianul *Tikhonov* [169] [170] este cunoscută și ca regresie "creastă" (*ridge regression*), scăderea ponderii (*weight decay*) sau regularizarea

Tikhonov, este o funcție de cost cu normalizarea L_2 pătratică a ponderilor. Cu alte cuvinte L_2 folosește normalizarea euclidiană pentru termenul de rugularizare. L_2 este foarte utilă pentru atenuarea problemei multi-colinearității în regresia liniară, care apare frecvent la modelele cu un număr mare de parametri. [171] Metoda este eficientă în problemele de estimare a parametrilor în schimbul unui bias admisibil. [172] Pentru majoritatea problemelor de clasificare și predicție se recomandă L_2 spre deosbire de L_1 care totuși este superior pentru cazul în care avem o mulțime de date irelevante. Aceste date pot fi fie date cu foarte mult zgomot, fie caracteristici care nu sunt informative, sau pot fi și date prea puține.

c) Oprirea timpurie

Oprirea timpurie (Early stopping) este o metodă de optimizare, fiind o formă de regularizare care presupune întreruperea antrenării atunci funcția de cost de validare nu mai scade, cu posibilitatea salvării celui mai bun model în timpul antrenării. [173] În timpul fiecărei iterații de antrenare, efectuăm o propagare înainte pentru a calcula ieșirile și propagarea înapoi pentru a calcula erorile [174]. Cu alte cuvinte, această metodă ajută la oprirea antrenării când metrica pe care o monitorizăm s-a oprit din îmbunătățire pentru un număr de epoci predefinit. Oprirea timpurie, de asemenea, ajută și la întreruperea antrenării modelului imediat ce se observă că modelul începe să devină supraspecializat.

În mod evident, ca în figura Fig.3.3.3, oprirea timpurie este cea mai simplă tehnică de a evita supraspecializarea, practic urmărim curba de validare în timpul antrenării și oprim actualizarea ponderilor odată ce eroarea de validare începe să crească. Eroarea de validare poate încă să coboare mai departe după ce a început să crească deoarece curbele erorii de validare reale au aproape întotdeauna mai mult de un minim local. [173]

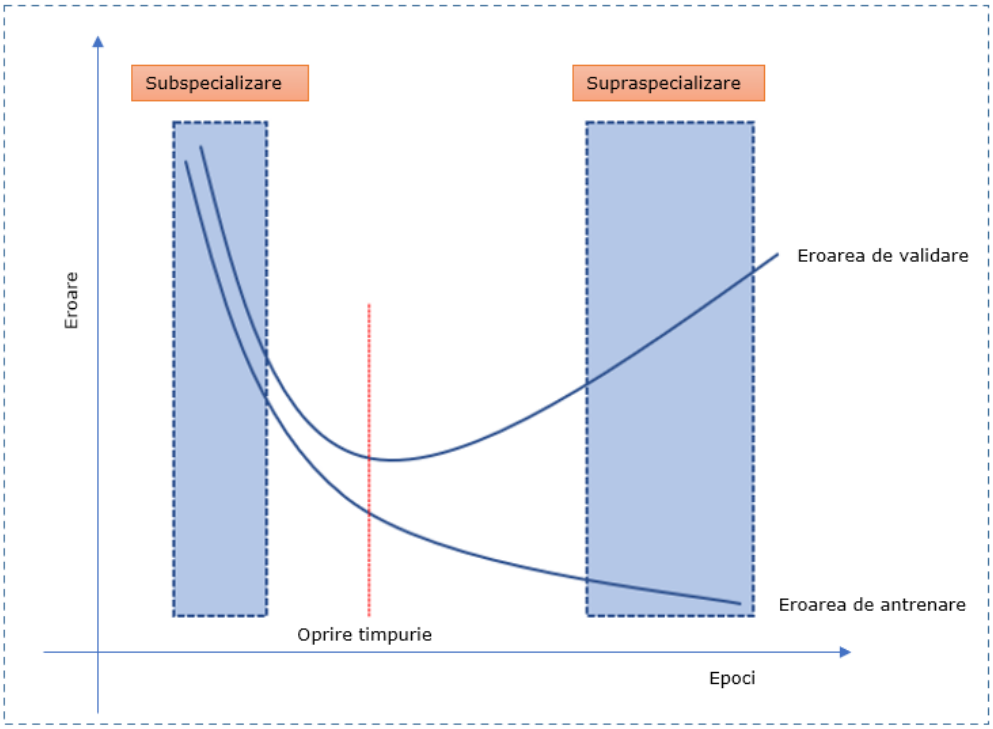


Fig.3.3. 3 Oprirea timpurie

d) Dropout

Dropout [175] [176] [177] [178] întâlnit și sub denumirea de diluare (*dilution*) este o tehnică foarte utilă a rețelelor neuronale și are efecte similare cu regularizarea care luptă împotriva supraspecializării, nu face altceva decât să reducă distanța dintre eroarea de antrenament și eroarea de test.
Termenul diluare se referă la micșorarea ponderilor. [179] Termenul dropout se referă la „abandonarea” sau la omiterea unităților (atât a celor ascunse cât și a celor vizibile) în timpul procesului de antrenament al unei rețele neuronale. [180] Regularizările L_1 și L_2 sunt considerate regularizări numerice în timp ce droput este o regularizare structurală.

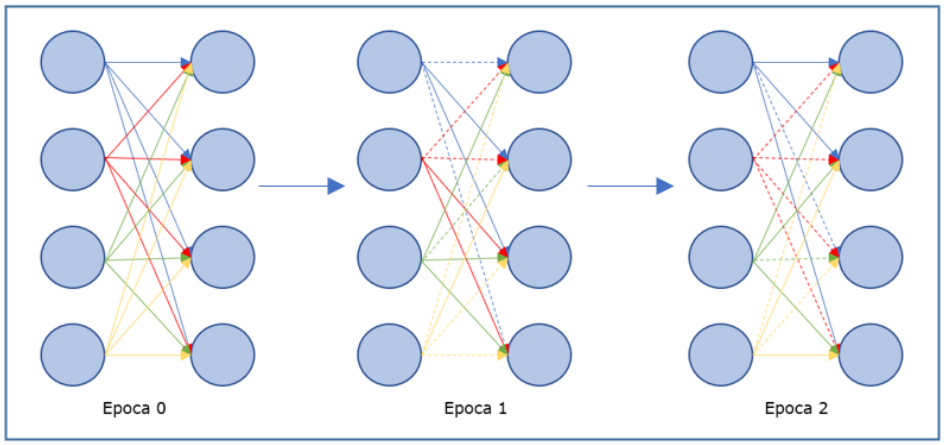


Fig.3.3. 4 Dropout $d = 0.4$

Se poate observa în Fig.3.3.4 că avem un parametru dropout de 0.4, acesta poate lua valori în intervalul [0,1] pentru a fi interpetrat ca o probabilitate, reprezentat prin liniile discontinue (punctate). Dacă la epoca 0 avem toți neuronii complet conectați, la epoca 0 avem 40% din neuroni sunt inactivi, doar 60% dintre neuroni fiind conectați. Astfel la fiecare pondere care este setată la zero îi corespunde o probabilitate d. Dropout forțează rețeaua să învețe reduceri, astfel se asigură o izolare bună a proprietăților necesare ale setului de date. O valoare recomandată pentru d este între 0.2 și 0.5 [5], dar la fel ca toți ceilalți hiperparametri se setează după mai multe experimente în funcție de setul de validare.

Dropout-ul a fost utilizat de o mare varietate de modele care utilizează o reprezentare distribuită; a fost utilizat de rețelele cu propagare înainte, de mașini Boltzmann și de rețelele neuronale recurente. Efectul principal acestei metode este încorporarea regularizării în faza de învățare. [181]

O variantă de Dropout este DropConnect, care aplică o abordare similară asupra ponderilor rețelei neuronale. [182]

4. Arhitecturi ale rețelelor neuronale

În cadrul acestui capitol voi prezenta pe scurt tipurile de arhitecturi pe care le-am folosit la implementarea propunerilor mele din capitolul 6 cu Funcții de activare propuse.

4.1. Rețele neuronale de convoluție

Rețelele neuronale de convoluție (*Convolutional Neural Network* - CNN) [15] apărute încă din 1988 nu a fost utilizate pe scară largă, din cauza limitărilor hardware privind puterea de calcul pentru antrenarea rețelei. LeCun [183] a folosit CNN-uri aplicate cu un algoritm de învățare bazat pe gradient și a obținut rezultate foarte bune pentru o sarcină de clasificare a cifrelor scrise de mână.

CNN-urile au două părți principale: extractoare de caracteristici (*feature extractor*) și un clasificator. În straturile de extragere a caracteristicilor, fiecare strat al rețelei primește ieșirea din stratul anterior ca intrare și ieșirea lui devine intrare pentru următorul strat. Arhitectura CNN constă dintr-o combinație de trei tipuri de straturi: convoluție, eșantionare maximă și clasificare. [184]

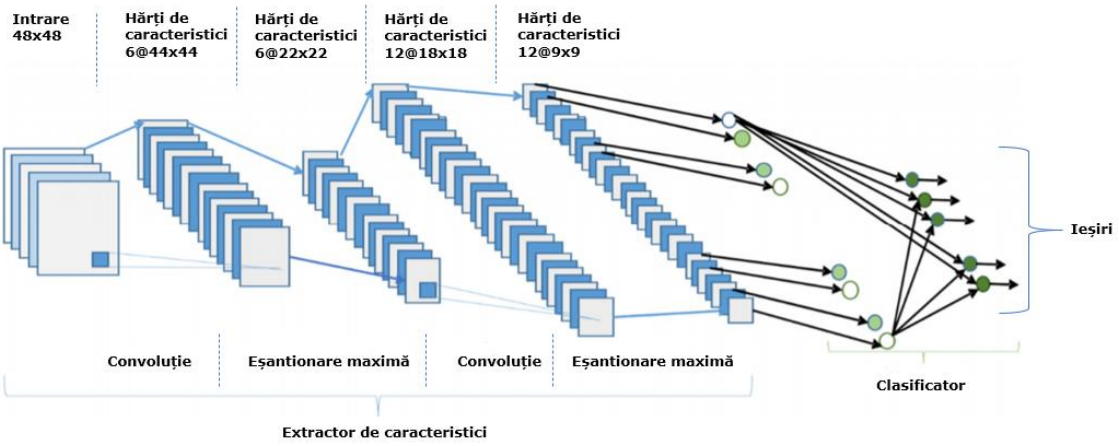


Fig.4.1. 1 Arhitectura CNN

4.2. Arhitectura LeNet-5

Arhitectura LeNet-5 deși a fost propusă încă din anii 1990, capacitatea de calcul și memoria limitată au făcut dificilă această implementare urmând ca în 2010 să fie folosită. [183] Arhitectura LeNet-5 se compune din intrare, două straturi de convoluție, două straturi de mostre, două straturi complet conectate (*straturi dense*) și un strat de ieșire.

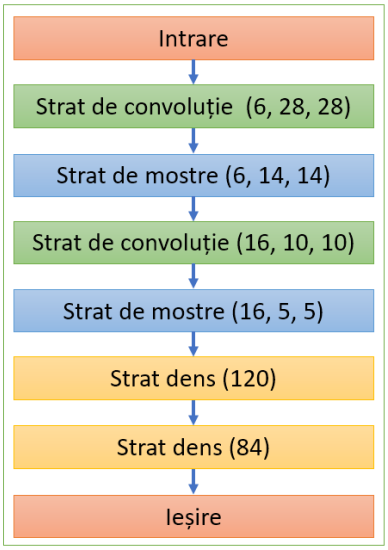


Fig.4.2. 1 Arhitectura LeNet-5

4.3. Rețele neuronale folosind blocuri

Rețelele neuronale folosind blocuri (*Visual Geometry Group* - VGG) [31] au apărut în 2014 în cadrul competiției "ImageNet Large Scale Visual Recognition Challenge (ILSVRC)".

Arhitectura VGG este formată din două straturi de convoluție (***conv*** în figura Fig.4.3.1) ambele folosesc funcția de activare ReLU, urmate de eșantionare maximă și straturi dense(***fc*** în figura Fig.4.3.1). Stratul final al modelului este un strat cu funcția de activare Softmax pentru clasificare. Există două tipuri de rețele VGG: VGG16 (16 straturi) și VGG19 (19 straturi).

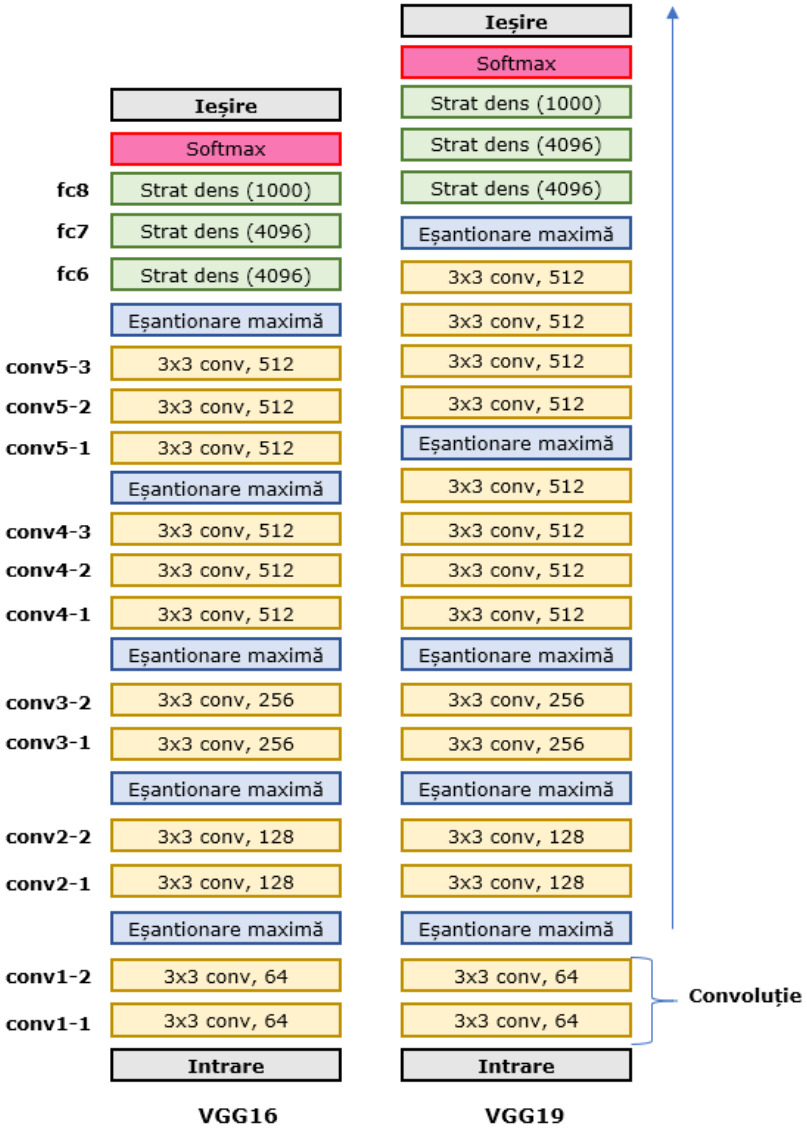


Fig.4.3. 1 Arhitectura VGG

4.4. Rețea în rețea

Rețea în rețea (*Network in Network* - NiN) este o arhitectură în care s-a propus înlocuirea activării liniare rectificate simple în rețele convoluționale cu o rețea complet conectată ai căror parametrii sunt învățați din date. Acest strat „*MLPConv*" leagă ieșirile tuturor filtrelor aplicate pe un grup și permite în mod arbitrar transformări complexe ale intrărilor. [123]

NiN este primul tip de rețea care utilizează convoluții pe mai multe straturi, unde convoluțiile sunt realizate cu un filtru 1 × 1 care ajută la adăugarea de mai multă non-liniaritate modelelor. Acest lucru ajută la creștere în adâncime a rețelei, care poate fi apoi regularizată cu tehnica Dropout. Tot în această arhitectură se

folosește și eșantionare globală medie (*Global Average Pooling* - GAP) ca o alternativă pentru straturile dense. Acest lucru ajută la reducerea semnificativă a numărului de parametri ai rețelei. GAP modifică semnificativ structura rețelei. Prin aplicarea GAP pe o hartă mare a funcțiilor, putem genera un vector de caracteristici de dimensiune mică fără a reduce dimensiunea hărților de caracteristici.

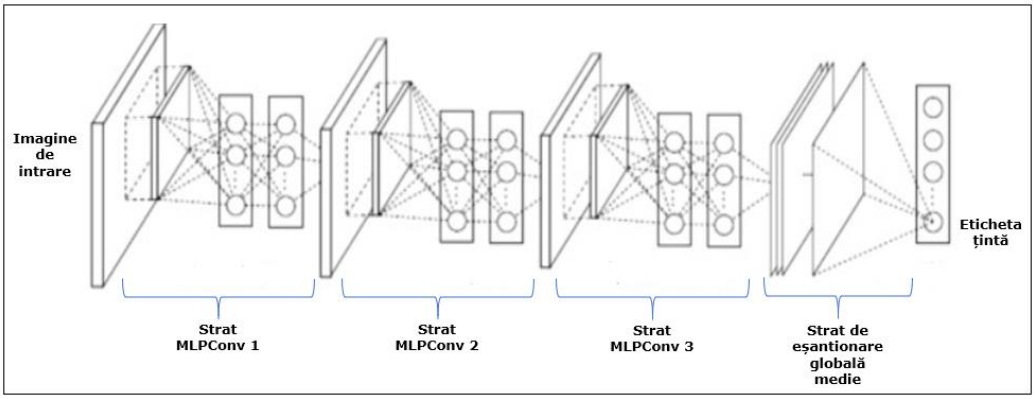


Fig.4.4. 1 Arhitectura NiN [185]

4.5. Rețele reziduale

Rețelele reziduale (*Residual Networks - ResNet*) [33] au apărut în 2015 și au fost dezvoltate pentru a facilita antrenarea rețelelor care sunt foarte profunde și care au milioane de parametri. Arhitectura ResNet are mai multe versiuni, versiunea 1 și 2 și poate fi dezvoltată cu mai multe straturi: 34, 50, 101, 152 și 1202. De exemplu, ResNet50 conține 49 de straturi de convoluție și 1 strat complet conectat (întâlnit în literatura de specialitate și ca strat dens) la finalul rețelei.

ResNet este o rețea de propagare înainte cu conexiune reziduală. Rețeaua reziduală este formată din mai multe blocuri reziduale de bază. Cu toate acestea, operațiunile din blocul rezidual pot fi variate în funcție de arhitectura diferită a rețelelor reziduale. [33]

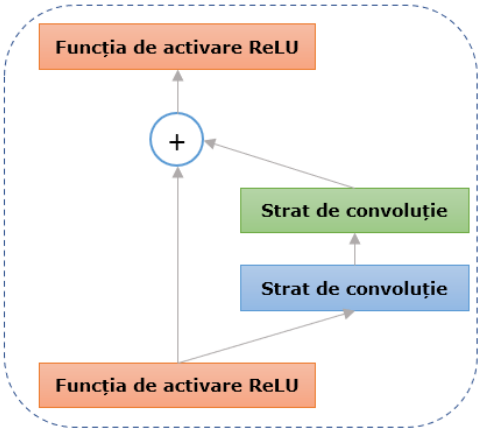


Fig.4.5. 1 Blocul rezidual [184]

4.6. Arhitectura MobileNet

Arhitectura MobileNet [158] dezvoltată de Google în 2017, se bazează pe o arhitectură simplificată care folosește convoluții separabile pe adâncime pentru a crea o pondere ușoară rețelelor neuronale.

MobileNet este o arhitectură eficientă pentru a construi modele mici, cu latență mică, care pot fi ușor adaptate la cerințele de proiectare pentru aplicații mobile și aplicații de vizualizare încorporate.

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Fig.4.6. 1 Arhitectura MobileNetV1 [158]

Prima arhitectură este MobileNet versiunea 1 [158], apoi a apărut MobileNetV2 [186] și între timp a apărut chiar și MobileNetV3 [187] care este adaptată pentru telefoane mobile cu unitate centrală de prelucrare (***C**entral **P**rocessing **U**nit* CPU).

MobileNetV2 [186] este o arhitectură dezvoltată de Google ca în Fig.4.6.2 în care avem o intrare ingustată (*bottleneck*), Dwise (strat de adâncime), Conv-Convoluție, liniar (se aplică funcție de activare liniară), scurtătură (*shortcut*), această arhitectură aduce o îmbunătățire semnificativă față de MobileNetV1 și lărgește aria de studiu pentru recunoaștere vizuală pe mobil, incluzând clasificarea, detecția obiectelor

și segmentarea semantică. Segmentarea semantică este o sarcină de grupare a părților dintr-o imagine care aparțin la aceiași clasă de obiecte. [188]

Voi folosi și o sarcină de segmentare semantică pe setul de date 2018 Data Science Bowl [189] pentru a testa funcționalitatea funcției de activare.

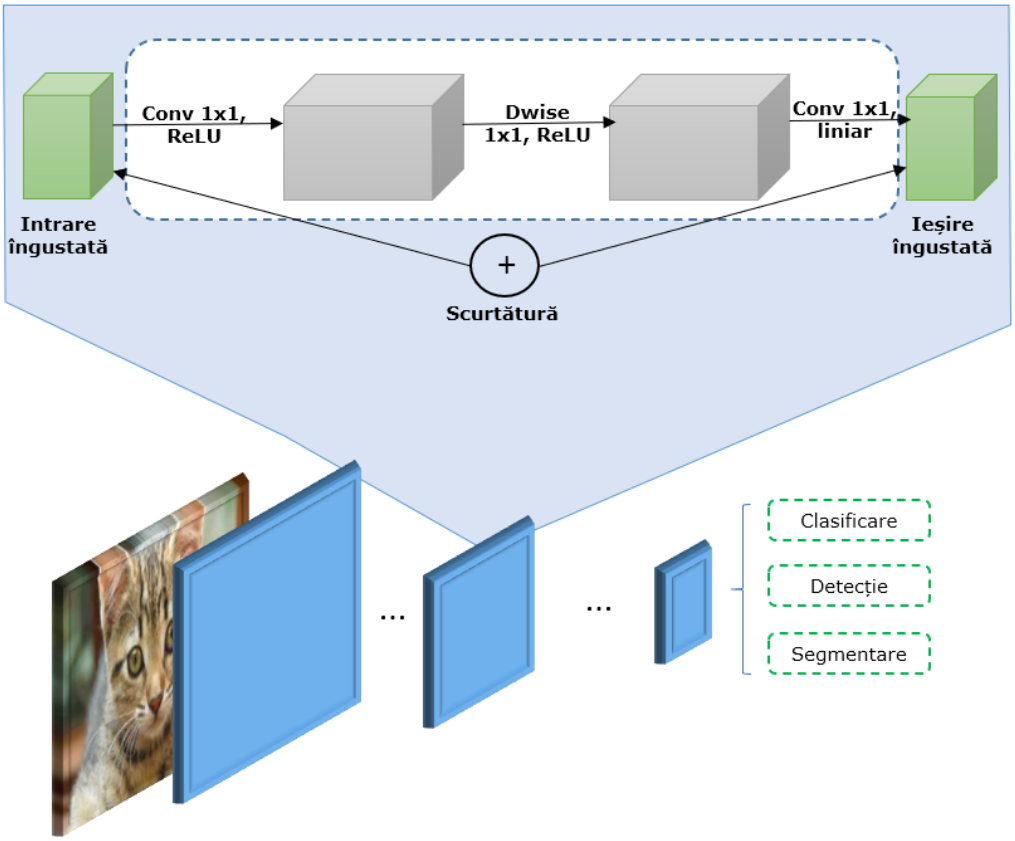


Fig.4.6. 2 Arhitectura MobileNetV2

4.7. Arhitectura Xception

Arhitectura Xception [159] este un nou tip de arhitectură pentru rețeaua neuronală de convoluție profundă inspirată de arhitectura Inception [32], dar în arhitectura Xception modulele Inception au fost înlocuite cu convoluții separabile pe adâncime.
Arhitectura Xception are 36 de straturi de convoluție care formează baza de extracție a caracteristicilor rețelei neuronale. Cele 36 de straturi de convoluție sunt structurate în 14 module, toate având conexiuni reziduale liniare în jurul lor, cu excepția primelor și ultimelor module. Cu alte cuvinte, arhitectura Xception este o stivă liniară de straturi de convoluție separabile pe adâncime cu conexiuni reziduale ca în figura următoare.

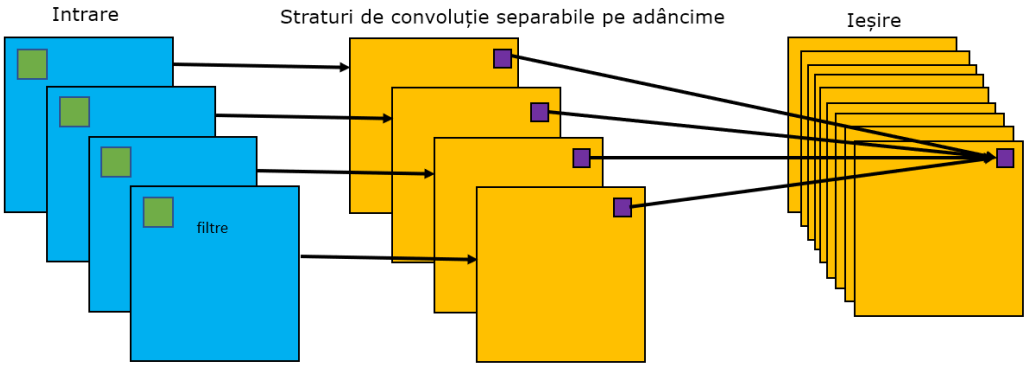


Fig.4.7. 1 Arhitectura Xception

4.8. Arhitectura AlexNet

Arhitectura AlexNet [26] este o arhitectură de rețea neuronală care a câștigat competiția ILSVRC în 2012. Rețeaua a fost antrenată pe 1,2 milioane de imagini de înaltă rezoluție având 1000 de clase diferite, cu 60 de milioane de parametri și 650.000 de neuroni. Arhitectura AlexNet se compune din cinci straturi de convoluție, dintre care unele sunt urmate de straturi de eșantionare maximă, urmate de trei straturi complet conectate și în final de un clasificator Softmax cu 1000 de unități.

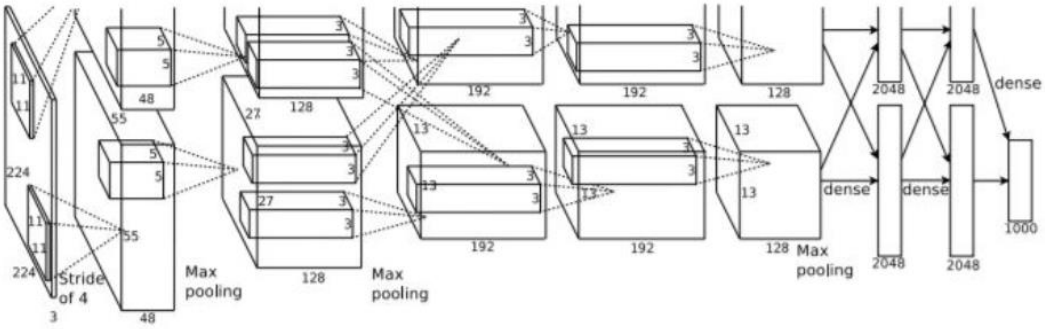


Fig.4.8. 1 Arhitectura AlexNet [26]

5. Învățarea Profundă pentru text și secvențe

În această teză am analizat și sarcini din domeniul Procesării Limbajului Natural, prin analiza sentimentelor pe mai multe seturi de date pe care le voi prezenta în Capitolul 7 în cadrul experimentelor privind acest capitol. Procesarea Limbajului Natural (*Natural Language Processing* - NLP) [190] este un domeniu care ajută la înțelegerea umană. Acesta implică un vector de cuvinte în care se reprezintă fiecare cuvânt. Tehnica prin care se face maparea între cuvinte și vectorii de numere reale este cunoscută și sub denumirea de încapsularea cuvintelor ("word embedding"). [191] Am folosit arhitecturi personalizate pentru sarcini de NLP pe seturile de date: Reuters [192], IMDB (*IMDb Rating of Movies*) [193] și FastText crawl 300d 2M [194].

CNN ajută și în cazul NLP, la extragerea de caracteristici de nivel superior din cuvinte. Caracteristicile abstracte extrase pot fi folosite în mod eficient în sarcini precum: analiza sentimentelor, traducerea automată și răspunsul la întrebări.

În 2008 s-au aplicat pentru prima dată CNN-uri pe sarcini de NLP. Această arhitectură de rețea neuronală de convoluție era dată de o propoziție, care era capabilă să producă o serie de predicții de procesare a limbajului precum etichete parțiale ale vorbirii, bucăți din text, roluri semantice, cuvinte similare semantic și probabilitatea ca propoziția să aibă sens din punct de vedere gramatical și semantic, folosind un model de procesare a limbajului natural. [195] Scopul metodei lor a fost transformarea cuvintelor într-o reprezentare vectorială a cuvintelor care învățau ponderi în timpul antrenării rețelei.

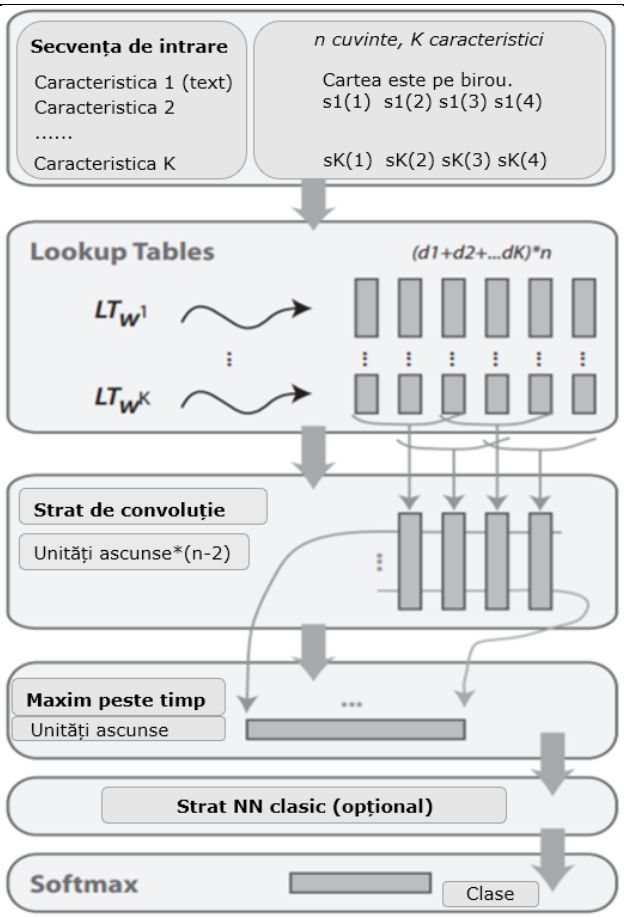


Fig.5. 1 Arhitectura unei rețele neuronale profunde în NLP [195]

În acest capitol prezint algoritmii de Învățare Profundă cu care putem prelucra textul care este definit ca secvențe de cuvinte sau secvențe de caractere. Cei doi algoritmi fundamentali de Învățare Profundă pentru procesarea secvențelor sunt rețelele neuronale recurente (*Recurent neural networks* - RNN) și rețelele de convoluție 1D, care sunt versiunea unidimensională a rețelelelor de convoluție 2D folosite la sarcini pe imagini.
Acești agloritmi se pot aplica în diverse domenii care cuprind exemple de cazuri precum:

- Clasificarea documentelor și clasificarea timpului, cum ar fi identificarea subiectului unui articol sau autorul unei cărți, data când a fost publicat articolul.
- Sumarizarea textului.

- Învățare pe secvență, cum ar fi decodificarea unei propoziții din engleză în altă limbă.

- Analiza sentimentelor, cum ar fi clasificarea recenziilor filmelor ca fiind pozitive sau negative, caz pe care îl voi testa în cadrul capitolului 7.

- Analiza sentimentelor, cum ar fi clasificarea recenziilor primite în cadrul unei oferte promoționale, privind satisfacția clientului de produsul sau serviciul respectiv.
- Previziunile meteorologice, cum ar fi prezicerea vremii viitoare într-o anumită locație, având în vedere datele meteorologice recente.
- Previziunile privind evoluția vânzărilor într-o anumită perioadă de timp impactată de diverși factori precum o criză financiară.
- Previziunile privind evoluția stocurilor într-un anumit sezon impactat de rata inflației.

Pentru modelarea propozițiilor cu un CNN, propozițiile sunt simbolizate mai întâi în cuvinte (*tokens*), care sunt transformate apoi într-un vector care conține totalitatea cuvintelor (*vector embedding*).

Într-o rețea neuronală recurentă simplă (*Recurent Neural Networks* - RNN) [196], fiecare intrare este evaluată pe un singur strat și este dată o ieșire. Pot fi cazurile unu-la-unu(*one-to-one*), unu-la-mai-mulți(*one-to-many*), mulți la unu (*many-to-one*) sau mulți la mulți (*many-to-many*) pentru maparea intrare-ieșire.
RNN analizează caracteristicile secvențiale ale intrării și o ieșire este returnată la pasul de analiză dintr-o buclă de feedback, permițând analiza caracteristicii curente în contextul caracteristicilor anterioare. Cu toate acestea deoarece fiecare etapă necesită feedback din etapa anterioară, RNN-urile nu pot profita de procesarea masivă în paralel (Massive Parallel Processing - MPP), așa cum pot CNN-urile.

În partea experimentală pentru o sarcină de analiză a sentimentelor am folosit și LSTM (*Long Short-Term Memory*) pe setul de date FastText crawl 300d 2M [194]. LSTM are porți de „uitare" suplimentare față de o rețea recurentă simplă. LSTM rezolvă atât problema dispariției cât și problema exploziei gradientului. LSTM permite erorii să se propage înapoi într-un număr nelimitat de pași în timp. Se compune din trei porți: de intrare, uitare și ieșire. [197] [198]

LSTM nu se folosește doar la imagini ci și în sarcini de voce sau video. De asemenea, rețelele LSTM sunt potrivite pentru clasificarea, procesarea și realizarea predicțiilor datelor din seriile de timp, deoarece pot exista decalaje de durată necunoscută între evenimentele importante dintr-o serie de timp. Din punct de vedere al modelului, o rețea neuronală recurentă care utilizează unități LSTM poate fi antrenată în mod supervizat [199] [200] [46], pe un set de date de antrenament, folosind un algoritm de optimizare, cum ar fi GD, combinat cu propagare înapoi pentru calculul gradienților necesari în timpul procesului de optimizare, pentru a modifica fiecare pondere a rețelei LSTM proporțional cu derivata erorii ponderii corespunzătoare. Minusul rețelelor LSTM este dat de faptul că procesează fluxuri de intrare continuă care nu sunt segmentate în subsecvențe cu limite marcate explicit la care starea internă a rețelei să poată fi resetată. Fără resetări, starea rețelei poate crește la nesfârșit și, în cele din urmă, va conduce la disfuncționalitatea rețelei. La acest minus, în [201] s-a propus este o „poartă uitată" nouă, adaptativă, care permite unei celule LSTM să învețe să se reseteze la orele corespunzătoare, eliberând astfel resursele interne.

6. Funcții de activare propuse

După cum s-a văzut în capitolul 2, alegerea funcției de activare are o influență ridicată asupra performanței rețelei neuronale. Un scop fundamental în Învățarea Profundă îl reprezintă alegerea corectă a funcției de activare care are ca rol introducerea neliniarității pentru a asigura învațarea de sarcini mai complexe. Așa cum sublinia și Pedamonti, funcțiile de activare joacă un rol important întrucât înțelegerea corectă a funcționării lor conduce la îmbunătățiri ale performanței. De asemenea, a arătat că performanța este influențată inclusiv de cât de profundă este rețeaua și de inițializarea ponderilor folosind distribuția Gaussiană uniformă [202]. În cadrul acestui capitol, voi prezenta mai multe propuneri de funcții de activare care aduc o îmbunătățire de performanță în antrenarea rețelelor neuronale.

6.1. Modificarea dinamică a funcției de activare folosind algoritmul de propagare înapoi în rețelele neuronale artificiale

Prin acest studiu propun modificarea dinamică a funcției de activare folosind o tehnică de învățare cunoscută, mai exact algoritmul propagării înapoi (*backpropagation-BP*). Modificarea constă în schimbarea dinamică a pantei pentru funcția de activare sigmoidă pe baza creșterii sau micșorării erorii într-o epocă de învățare.

Studiul a fost realizat folosind platforma **WEKA** (**W**aikato **E**nvironment for **K**nowledge **A**nalysis) prin adăugarea acestei funcții în clasa Perceptron multistrat (Multi-layer Perceptron -MLP). Acest studiu urmărește modificarea dinamică a funcției de activare care s-a schimbat în funcție de eroarea relativă a gradientului și un aspect de menționat este că în definirea arhitecturii rețelei neuronale nu am folosit straturi ascunse pentru acest studiu.

Comportamentul rețelelor neuronale artificiale a fost o zonă care a fost studiată pe larg în timp, oferind rezultate consistente. S-a studiat și puterea de calcul a rețelelor neuronale artificiale prin activarea sigmoidului recurent. [203]

În aplicații, funcția de activare și selectarea gradientului afectează direct convergența rețelei. [204] În acest context, comportamentul universal al rețelelor neuronale Turing a fost studiat pentru o funcție specifică de activare, care a fost numită funcție liniară de forma:

$$f(x) = a \cdot x \tag{6.1.1}$$

Unde $\alpha \in \boldsymbol{R}$. Funcția de activare joacă un rol important în calculul rețelelor artificiale neuronale. [205] [206]

Rețelele neuronale reprezintă o metodă importantă pentru analiza funcției de activare din cauza capacității lor de a face față seturilor de date care se schimbă. De exemplu, în MLP, cel mai frecvent sistem, neuronii sunt organizați în straturi. [207] Una dintre funcțiile de activare studiate pe scară largă în literatura de specialitate este funcția sigmoidă:

$$f(x) = \frac{1}{1+e^{-x}} \tag{6.1.2}$$

Aceste funcții reprezintă principalele funcții de activare, fiind în prezent cele mai utilizate de rețelele neuronale, reprezentând din acest motiv un standard în construirea unei arhitecturi de tip rețea neuronală. [38] [39]

Funcțiile de activare care au apărut de-a lungul anilor s-au analizat prin prisma aplicabilității algoritmului BP. S-a remarcat faptul că o funcție care este diferențiabilă în rețeaua neuronală permite ca eroarea funcției să fie și ea diferențiabilă. [154]

Scopul principal al funcției de activare constă în scalarea ieșirilor neuronilor în rețelele neuronale și introducerea unei relații neliniare între intrarea și ieșirea neuronului. [208]

Pe de altă parte, funcția sigmoidă este folosită de obicei pentru straturile ascunse, deoarece combină comportamentul liniar, curbiliniu, constant și depinde de valoarea de intrare. [209] De asemenea, s-a demonstrat că funcția sigmoidă nu este eficientă pentru o singură unitate ascunsă, dar când sunt implicate mai multe unități ascunse, aceasta devine mai utilă. [210] Motiv pentru care în următoarele studii voi folosi arhitecturi complexe, cu mai multe straturi ascunse (profunde).

Abordarea mea în această lucrare constă în modificarea dinamică a funcției de activare folosind algoritmul BP pentru antrenarea unei rețele neuronale artificiale.

Algoritm de propagare înapoi cu modificarea dinamică a funcției de activare

Elementul principal al unei rețele neuronale artificiale este neuronul artificial, care poate fi reprezentat printr-o sumă și o funcție, deci rețeaua va fi compusă din mai multe funcții interconectate. Mai clar, aceste funcții sunt filtrele prin care trec informațiile. Și în funcție de modul în care dorim să învățăm rețeaua neuronală, aceste funcții au caracteristici specifice scopului ales. [211] [212] [213]

Indiferent de tipul funcției de activare și caracteristicile sale, neuronul rămâne neschimbat în timpul procesului de antrenare.

Consider că acest aspect reprezintă o restricție majoră. Cum învățarea este un proces de schimbare dinamică a ponderilor care caracterizează numeric conexiunile dintre neuronii rețelei, este în mod normal ca acest proces de învățare să fie reflectat dinamic în funcția de activare prin schimbarea dinamică a caracteristicilor corespunzătoare unui neuron. [214]

Propun modificarea ecuației funcției sigmoide cu un parametru β care este modificat dinamic în timpul antrenamentului, cu alte cuvinte acest parametru β este un parametru învățabil.

$$f(x) = \frac{1}{1+e^{-\beta x}} \tag{6.1.3}$$

Funcția propusă nu are niciun fel de constrângere, $x \in (-\infty, +\infty)$, este o funcție monotonă și nu este centrată în 0, iar ieșirea este în intervalul $y \in (0, 1)$. La fel ca tanh și sigmoida fără acest parametru modificat dinamic, această propunere este mărginită în partea superioară și inferioară, proprietate care necesită o atenție sporită la inițializarea rețelei pentru a rămâne în limitele impuse de funcție și a nu afecta performanța.

6.2. Cele mai utilizate funcții de activare: clasic versus curent

Prin acest studiu propun să ofer o imagine de ansamblu sumarizată asupra celor mai utilizate funcții de activare, funcții clasice și funcții curente. Când spun clasic, mă refer la primele funcții de activare, cele mai populare și folosite în trecut. Dar, din cauza dezavantajelor lor, au apărut alte noi funcții de activare pe care le numesc curente.

Aceste funcții sunt printre cele mai cunoscute funcții de activare ale Inteligenței Artificiale, ale Învățării Automate și ale Învățării Profunde.

Cu fiecare funcție, ofer o scurtă descriere a funcției de activare, discut impactul acesteia și arăt domeniul unde este aplicabilă, avantajele și dezavantajele acesteia și mai multe detalii pentru o clarificare amplă. Aceste funcții acoperă mai multe probleme cum ar fi dispariția gradientului, explozia gradientului, când folosim GD și așa mai departe.

Aceste soluții la aceste probleme reprezintă unul din topicurile cele mai importante din aria de cercetare și dezvoltare a Inteligenței Artificiale.

În acest subcapitol voi menționa doar funcțiile curentului studiu, deoarece prezentarea în detaliu a avantajelor și dezavantajelor acestor funcții s-a făcut în cadrul capitolul 2, subcapitolul 2.3 la prezentarea stadiului actual privind funcțiile de activare.

Printre funcțiile de activare clasice am avut ca obiect de studiu: funcția de activare sigmoidă și funcția de activare tangentă (*tanh*). Cele două funcții sunt foarte similare, ambele necesită calcul exponențial, explozia/dispariția gradientului, diferența este dată de intervalul ieșirii activării, în cazul sigmoidei este *[0,1]*, iar în cazul funcției tanh *[-1,1].*

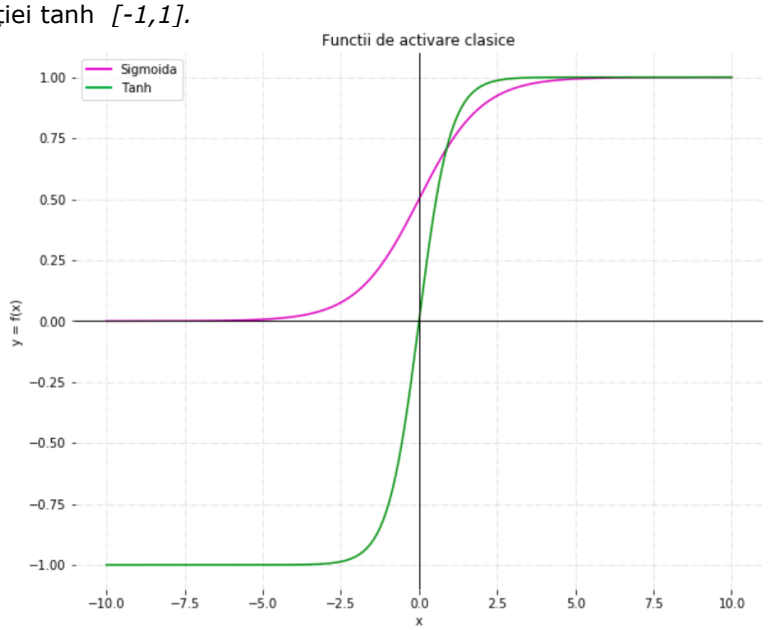


Fig.6.2. 1 Funcții de activare clasice

Printre funcțiile de activare curente am avut ca obiect de studiu: ReLU, PReLU, LReLU, ELU, SELU și Softmax.

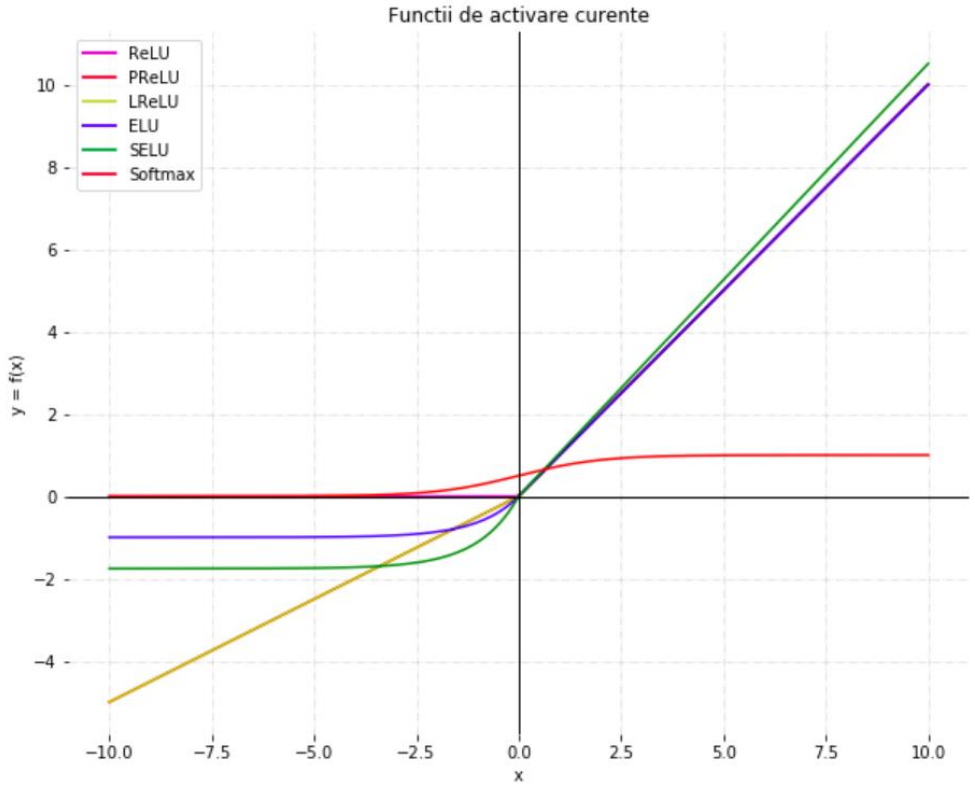


Fig.6.2. 2 Funcții de activare curente

După cum se poate observa mai sus, în Fig.6.2.2., am reprezentat funcțiile curente pe care le-am prezentat în cadrul acestui studiu propus, mai exact: ReLU, PReLU, Leaky ReLU, ELU, SELU și Softmax. În tabelele de mai jos se pot vedea cele mai remarcabile proprietăți, avantaje și dezavantaje ale funcțiilor clasice și curente ale funcțiilor de activare. Fiecare nouă funcție de activare care apare aduce o rezolvare problemelor pe care le au funcțiile precedente.

Tabel 6.2. 1 Proprietățile funcțiilor de activare clasice și curente

Tip	Funcție de activare	Interval	Monoton	Monoton și derivabil	Aproximare către origine
Clasic	Sigmoida	(0,1)	Da	Nu	Nu
	Tanh	(-1,1)	Da	Da	Da
Curent	ReLU	[0, +∞)	Da	Nu	Nu
	PReLU	(-∞,+∞)	Da, dacă $\alpha \geq 0$	Da, dacă $\alpha = 0$	Da, dacă $\alpha = 1$
	LReLU	(-∞,+∞)	Da	Nu	Nu
	ELU	(-α, +∞)	Da, dacă $\alpha \geq 0$	Da, dacă $\alpha = 1$	Da, dacă $\alpha = 1$
	SELU	(-αλ, +∞)	Da	Nu	Nu
	Softmax	(-1,1)	Da	Da	Da

Tabel 6.2. 2 Avantaje și dezavantaje ale funcțiilor de activare clasice și curente

Tip	Funcție de activare	Avantaje	Dezavantaje
Clasic	Sigmoida	• Este neliniară. • Este diferențiabilă peste tot, se poate folosi la o propagare înapoi bazată pe gradient. • Intervalul său variază de la 0 la 1, astfel poate genera probabilități.	• Gradientul pentru intrările care sunt departe de origine este aproape zero, astfel învățarea gradienților este lentă pentru neuronii saturați. • Când este utilizată ca activare finală într-un clasificator, suma tuturor claselor nu este egală 1. • Necesită calcul exponențial • Derivatele funcției sigmoide sunt foarte mici, maximul fiind ≈0.25, care în cazul operațiilor de multiplicare conduce la numere și mai mici, cauzând problema dispariției gradientului.
	tanh	• Este o redimensionare a funcției sigmoide, ieșirile sale fiind cuprinse între -1 și 1, acest domeniu fiind mai convenabil pentru rețelele neuronale. • Este importantă pentru optimizare [215] • Datorită datelor centrate în 0, derivatele sunt mari. Funcțiile derivate sunt în intervalul [0,1] astfel că derivata ia mai multe valori. • Are un interval mai larg pentru un antreament mai rapid.	• Dispariția gradientului.
	ReLU	• Îmbunătățește minusurile sigmoidei, astfel încât permite lucrul cu variații de intensitate mare, care	• Saturație pentru valorile negative, permițând acestor valori să fie inactive, altfel zis problema exploziei gradientului când valorile x <0, gradientul este egal cu 0.

Curent		au un caracter mai natural decât în cazul unităților binare. [73] • Mai eficientă computațional • Accelerează convergența folosind algoritmul gradientul stocastic descendent comparativ cu sigmoida și tanh [26]. • Caracteristica principală a funcției ReLU este evitarea problemei de dispariție a gradientului, minus întâlnit la funcțiile clasice sigmoidă și tanh. • ReLU este mai potrivită și dă rezultate mai bune în formarea rețelei neuronale profunde [40].	
	PReLU	• Regiunea negativă a antrenării aduce valori • Are un parametru învățabil • Cost computațional marginal.	• Saturație pentru valori negative mari, devenid din nou inactive.
	LReLU	• Puțin balansat pentru valorile negative din aproprierea lui 0.	• Viteza de antrenare este redusă • Spre deosebire de PReLU parametrul leaky este predefinit și rețelele neuronale nu pot decide valoarea lui. •
	ELU	• Nu are problema ReLU "mort". • Funcția de cost tinde să conveargă spre 0 mai rapid. • Rezultate ale performanței mai mari • Viteza învățării profunde mari a rețelei	• Saturație pentru valorile negative cu valori mici ale intrărilor care se propagă înainte și pierd informații.

		conduce la performanțe ridicate de clasificare. • Atenuează problema dispariției gradientului. • Are o putere de supraspecializare ridicată în comparație cu ReLU și PReLU pentru rețelele cu mai mult de 5 straturi.	
	SELU	• Un fel de ELU • A și λ sunt 2 parametri fixi, deci nu putem să propagăm înapoi prin ei • Convergență mai rapidă și rezultate mai bune pe MNIST	• Nu poate fi folosită singură, necesită o tehnică personalizată de inițializare a ponderilor
	Softmax	• Foarte similară cu funcția sigmoidă • Este preferată pentru stratul de ieșire în modelele profunde mai ales când se clasifică mai mult de două clase • Foarte rapidă pentru antrenare și predicție	• Nu va funcționa dacă datele nu sunt liniar separabile • Nu suportă rejecția nulă

6.3. TeLU: o nouă funcție de activare pentru Învățarea Profundă

Funcția de activare joacă un rol important în definirea arhitecturii pentru o rețea neuronală artificială, impactul său nu numai că are impact în faza de antrenament, ci și în faza de testare. În zilele noastre, cea mai cunoscută și folosită funcție de activare este funcția ReLU pentru proprietățile sale benefice. Acest studiu are ca scop introducerea a două funcții de activare noi, pe care le-am numit TeLU și TeLU învățabilă (TeLU learnable în figura 6.3.1). Aceste propuneri sunt o combinație de ReLU, *tanh* și ELU fără și cu un parametru învățabil. Arăt că funcțiile de activare TeLU și TeLU învățabilă pot da rezultate mai bune decât alte funcții populare de activare, inclusiv ReLU, Mish, TanhExp [216], folosind arhitecturile actuale testate pe seturile de date din domeniul Vederii Artificiale.

În experimente voi folosi funcția de activare Swish, respectiv swish_1. Swish_1 corespunde cazului când β = 1 și, de asemenea, am testat Swish cu parametru învățabil.
Funcția de activare TanhExp (Exponential Tanh Exponential) este definită astfel:

$$f(x) = x \cdot \tanh(e^x) \tag{6.3.1}$$

După cum s-a văzut în capitolul 2, funcția Swish a fost combinația dintre o funcție sigmoidă unipolară și funcția de activare ReLU.

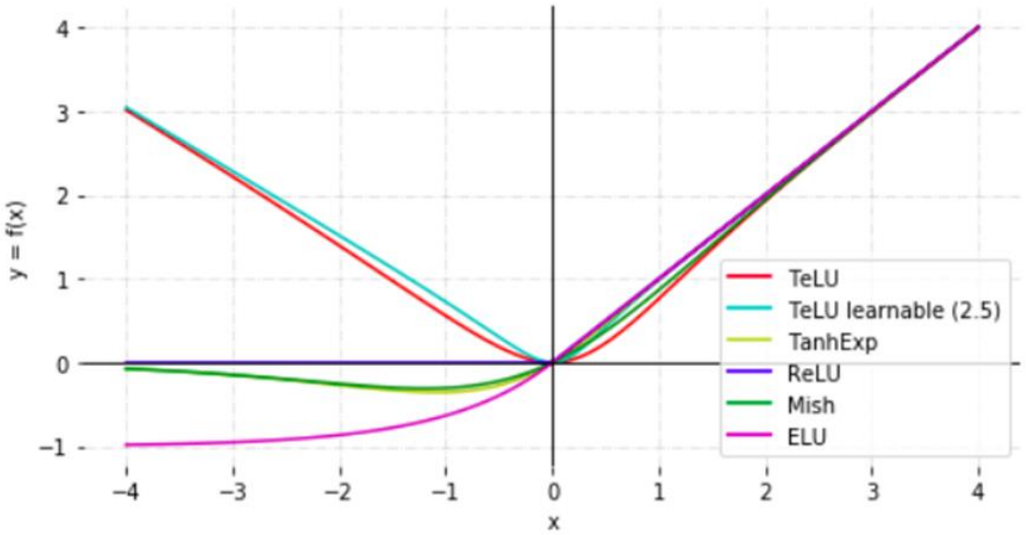


Fig.6.3. 1 Funcții de activare

După cum s-a văzut mai devreme, funcția TanhExp a fost o combinație între ReLU, *tanh* și argumentul exp. Propunerea mea constă în dezvoltarea a două funcții noi care derivă din ReLU, tanh și funcția de activare ELU, acestea sunt destul de asemănătoare cu funcția TanExp apărută în 2020, dar diferența principală constă în utilizarea funcției ELU în loc de argumentul exp și de asemenea, proprietatea de fi învățabilă introdusă prin setarea unui parametru învățabil α. Ecuația funcției TeLU învățabilă este dată de următoarea relație:

$$f(x) = x \cdot \tanh(elu(\alpha \cdot x)) \tag{6.3.2}$$

Unde este α un parametru învățabil, pe care l-am inițializat cu 0,1. Funcția TeLU este doar un caz particular al funcției TeLU învățabilă cu setarea parametrului α la 1.
Unde ELU la rândul ei poate fi scrisă astfel:

$$f(x) = elu(x) = \alpha \cdot e^x - 1 \tag{6.3.3}$$

Unde α> 0 este un hiper-parametru care controlează valoarea la care funcția se saturează pentru intrări negative.
Ecuația derivatei funcției TeLU este dată de următoarea relație:

$$f'(x) = \tanh(elu(x)) + elu(x) \cdot \operatorname{sech}^2(elu(x)) \tag{6.3.4}$$

Unde funcția secantă este dată de relația:

$$f(x) = \text{sech}(x) = \frac{2}{e^{x} + e^{-x}} \tag{6.3.5}$$

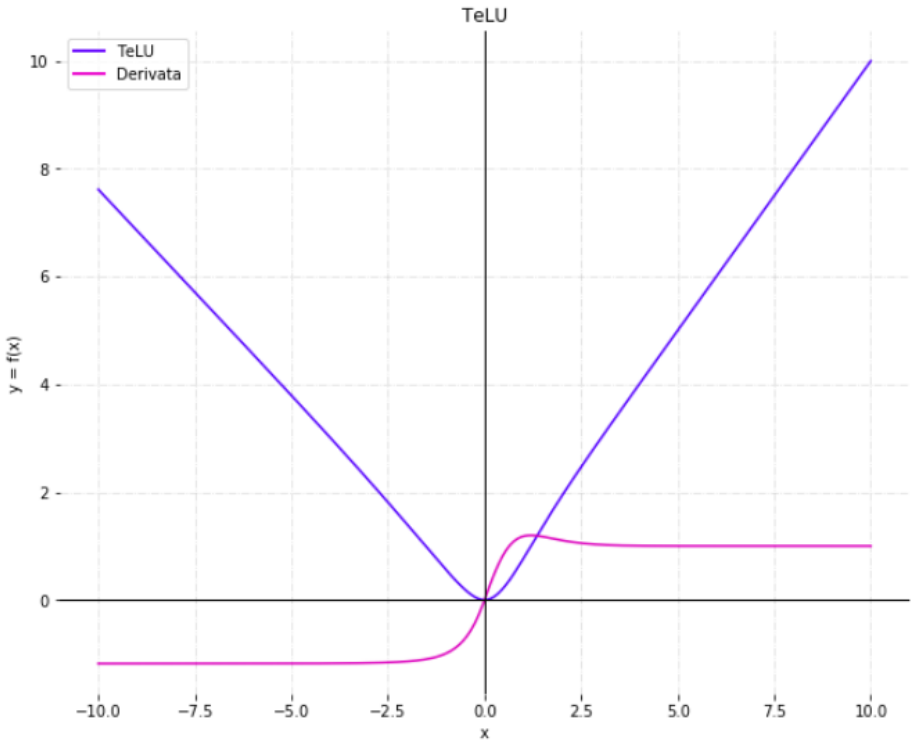


Fig.6.3. 2 Funcția de activare TeLU și derivata sa

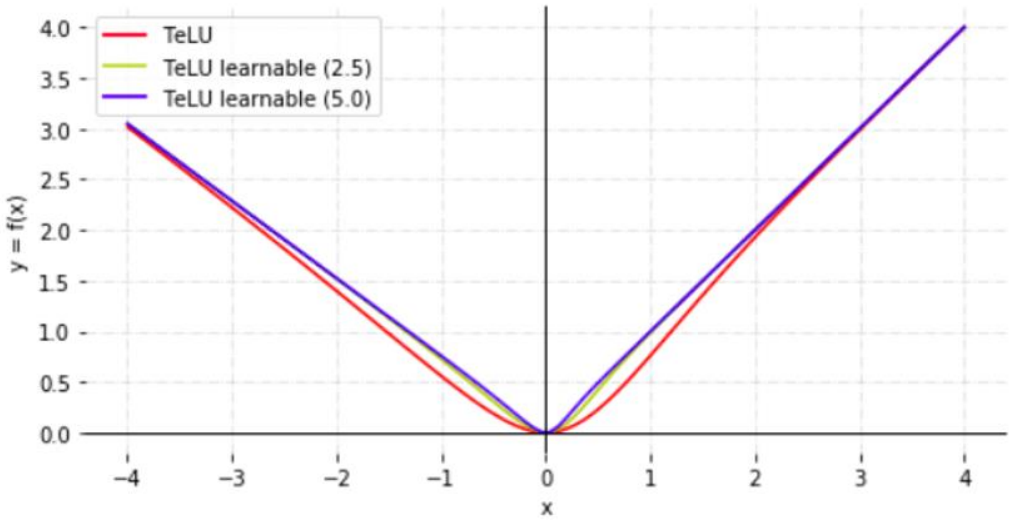


Fig.6.3. 3 Funcțiile de activare TeLU și TeLU învățabilă

Parametrul α învățabil l-am inițializat cu 0,1 pentru seturile de date MNIST, Fashion-MNIST, CIFAR-10 și CIFAR-100. La proiectarea funcțiilor TeLU și TeLU învățabilă m-am inspirat din funcția TanExp și funcția Mish. Implementarea funcțiilor am realizat-o în Tensorflow 2.2.4, folosind API-ul Keras. Pentru antrenare, voi avea în vedere arhitecturile ResNet18 [33], LeNet-5 [21], MobileNet [158], AlexNet [26], cu diferite profunzimi. În urma experimentelor, demonstrez că TeLU și TeLU învățabilă depășesc alte funcții de activare populare pe o varietate de rețele neuronale artificiale profunde.

Din punct de vedere al proprietăților funcțiilor de activare, TeLU și TeLU învățabilă sunt funcții centrate în 0, continue, netede, non-monotone și mărginite în partea inferioară. Graficele funcțiilor TeLU și TeLU învățabilă sunt destul de similare, dar TeLU învățabilă variază în funcție de parametrul său care pentru valori mari se aproprie tot mai mult de axa $0y$. TeLU ca și funcția TanhExp necesită mai puține calcule. Prima derivată a lui TeLU învățabilă poate fi calculată după cum urmează:

$$f'(x) = \tanh(\alpha \cdot (e^x - 1)) + \alpha \cdot x \cdot e^x \cdot \operatorname{sech}^2(\alpha(e^x - 1)) \tag{6.3.5}$$

Proprietățile funcțiilor de activare propuse

TeLU și TeLU învățabilă fiind două funcții mărginite inferior aduc un plus printr-o regularizare foarte puternică care conduce la o optimizare a rețelei.

Pe partea pozitivă, TeLU este aproape egală cu o transformare liniară, când intrarea este mai mare de 1, valabil și pentru partea negativă când intrarea este mai mică de -1 funcția devine aproape o transformare liniară. TeLU și TeLU învățabilă arată un gradient mai abrupt aproape de zero, care poate accelera actualizarea parametrilor din rețea. Pe parcursul propagării înapoi, rețeaua își actualizează parametri, nu are probleme de oprire a antrenamentului.

6.4. Îmbunătățirea preciziei rețelelor neuronale profunde prin dezvoltarea de noi funcții de activare

Propun patru funcții de activare care aduc îmbunătățiri pentru seturi de date diferite în sarcini din Vederea Artificială. Aceste funcții sunt o combinație a funcțiilor populare de activare, cum ar fi sigmoida, sigmoida bipolară, ReLU și *tanh*. Permițând funcțiilor de activare să fie învățabile, obținem modele mai robuste. Pentru validarea acestor funcții, am testat folosind mai multe seturi de date și mai multe arhitecturi cu adâncimi diferite, arătând că proprietățile lor sunt semnificative și utile. De asemenea, le-am comparat cu alte funcții puternice de activare pentru a vedea cum propunerile mele au impact asupra îmbunătățirii performanței.

Având atât de multe date, avem nevoie și de un model puternic pentru a obține o performanță bună și trebuie să acordăm o atenție sporită în alegerea potrivită a funcției de activare. Principalele arhitecturi de învățare profundă utilizate pentru procesarea imaginilor sunt CNN, sau cadre precum VGG [31], Inception [32], ResNet [33]. De obicei, aceste arhitecturi sunt foarte profunde, au nevoie de putere de calcul, astfel încât GPU-urile sunt utilizate pentru a reduce timpul de calcul, eu am folosit Colab de la Google pentru a antrena cu GPU-urile puse la dispoziție.

Pentru toate sarcinile, rețeaua încearcă să înțeleagă și să clasifice volumul mare de informații în informații bune sau rele. Acest lucru joacă un rol important, deoarece nu toate informațiile sunt utile pentru sarcinile noastre. Unele dintre aceste informații reprezintă un volum mare de zgomot care va avea impact asupra performanței modelului. Deci, funcția de activare ajută modelul să păstreze informațiile bune și să suprime informațiile irelevante.

Deși au fost dezvoltate multe funcții de activare, nu putem spune care este cea mai bună alegere, deoarece depinde de sarcină, setul de date, profunzimea arhitecturii și așa mai departe. De exemplu, ReLU, în ciuda performanței sale, are problema "ReLU mort", atunci când neuronii devin inactivi și returnează doar 0 pentru orice intrare. [217] În cercetarea lor, autorii propun o nouă procedură de inițializare, numită inițializare asimetrică randomizată, pentru a preveni inactivitatea ReLU. Performanța funcțiilor de activare depinde de diferite sarcini și seturi de date. [3]

În cadrul acestui subcapitol descriu noile funcții de activare propuse și arăt impactul lor asupra mai multor arhitecturi folosind mai multe seturi de date, precum setul de date Câini și Pisici, CIFAR-10, CIFAR-100. Printre arhitecturile utilizate menționez ResNet18, ResNet34 [33], de asemenea, am folosit tehnica Învățării prin Transfer (*Transfer Learning* -TL) în cazul utilizării arhitecturii VGG16 [218], arhitectură pre-antrenată cu mărirea datelor (*Data Augmentation* -DA). Mărirea datelor este o tehnică pentru evitarea supraspecializării. Pentru extragerea caracteristicilor am folosit rețeaua reziduală (ResNet) și arhitectura VGG.

Învățarea reziduală ușurează antrenarea în rețele neuronale mai profunde și facilitează optimizarea. Am folosit ResNet18 (având o adâncime egală cu 18 straturi) și ResNet34 (având o adâncime egală cu 32 de straturi). Un lucru remarcabil este faptul că performanța crește odată cu creșterea adâncimii rețelei, dar devine obligatoriu să știm cum să adăugăm straturi care conduc la o creștere a complexității și expresivității rețelei.

Practic, un bloc VGG conține o secvență de straturi de convoluție, urmată de un strat de selecție a valorii maxime (*maxpooling*) pentru eșantionarea spațială. Avantajul VGG constă în utilizarea blocurilor care conduc la o reprezentare compactă mare în definirea modelului.

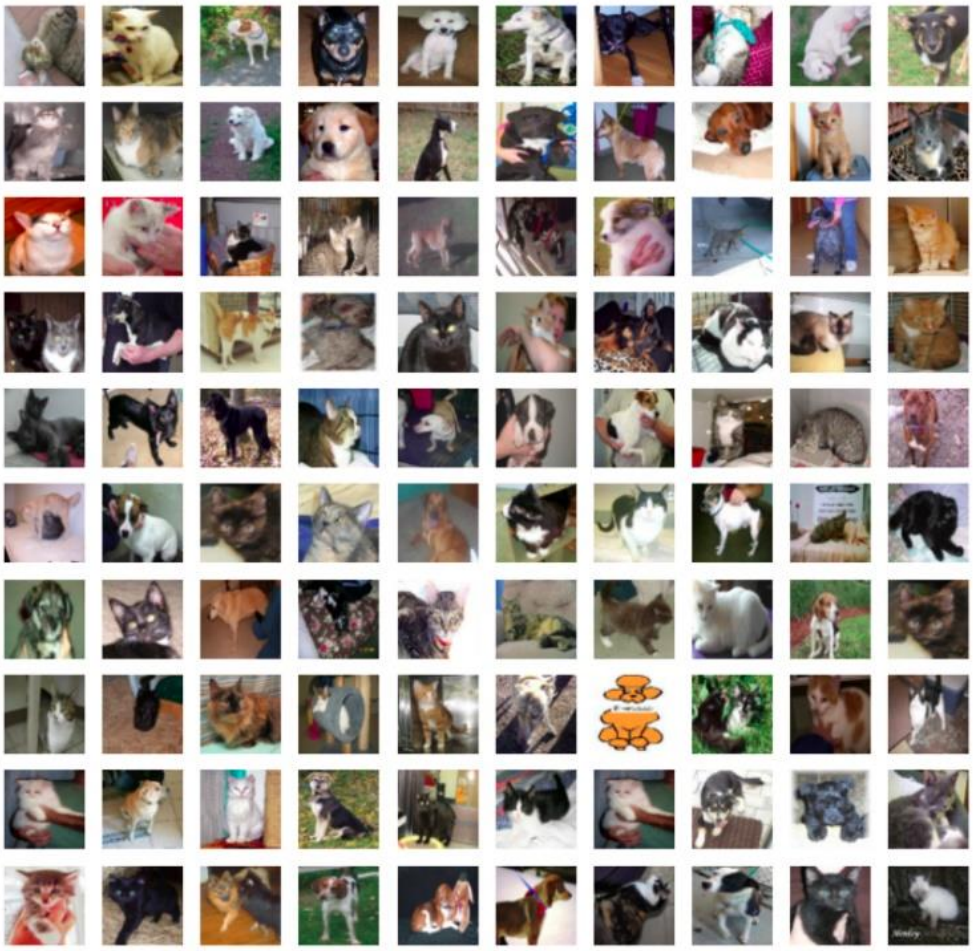


Fig.6.4. 1 Setul de date Câini și Pisici (Sursa: www.kaggle.com)

6.4.1. Funcția de activare TSReLU

Această funcție de activare constă în combinarea funcției ReLU, funcția *tanh* și funcția sigmoidă, pe care am denumit-o TSReLU (**T**angent-**S**igmoid-**ReLU**). Deci, această funcție este o combinație între două funcții clasice (sigmoida și *tanh*) și o funcție curentă, dată de ReLU. [219]
Expresia sa este foarte ușoară și poate fi văzută în ecuația de mai jos:

$$f(x) = x \cdot \tanh\left(sigmoid(x)\right) \tag{6.4.1.1}$$

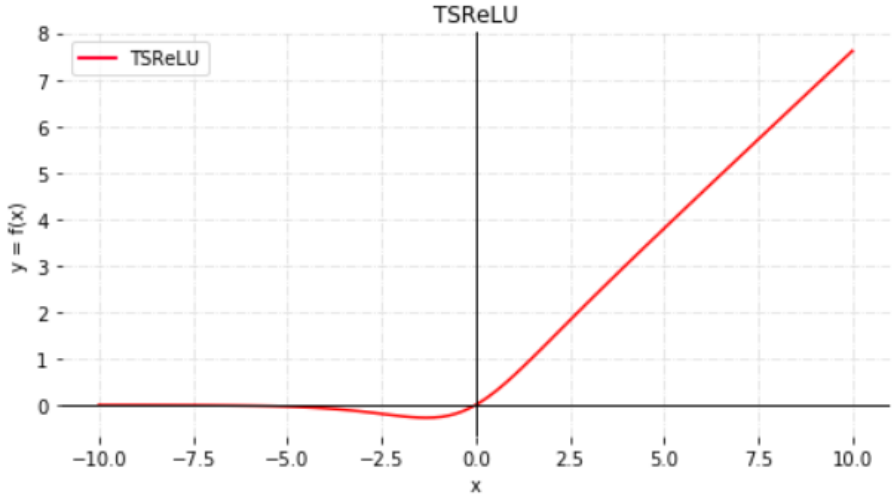


Fig.6.4.1. 1 Funcția de activare TSReLU

Pentru a valida această funcție de activare am folosit seturile de date CIFAR-10, CIFAR-100 și setul de date Câini și Pisici folosind tehnica TL cu o arhitectură VGG pre-antrenată. Am antrenat rețeaua pe 3000 de imagini și 1000 de imagini de validare.

Din graficul său se poate vedea că TSReLU este o funcție continuă, netedă, non-monotonă, mărginită în partea inferioară, nemărginită în partea superioară, proprietate foarte dorită datorită faptului că evită un timp de antrenament lent în timpul gradienților aproape de zero.

6.4.2. Funcția de activare TSReLU învățabilă

Această funcție de activare este destul de similară cu TSReLU, singura diferență este dată de un parametru care este învățabil. Inițializez acest parametru $\alpha = 0.5$.
Expresia funcției TSReLU învățabilă este dată de relația:

$$f(x) = x \cdot \tanh\left(\alpha \cdot sigmoid(x)\right) \tag{6.4.2.1}$$

În Fig.6.4.2.1. putem vedea pentru funcția de activare TSReLU învățabilă în care parametrul α este setat pe 1, 2.5 sau 5, pentru a avea o vizualizare mai bună a modului în care curba se schimbă în funcție de acest parametru α.

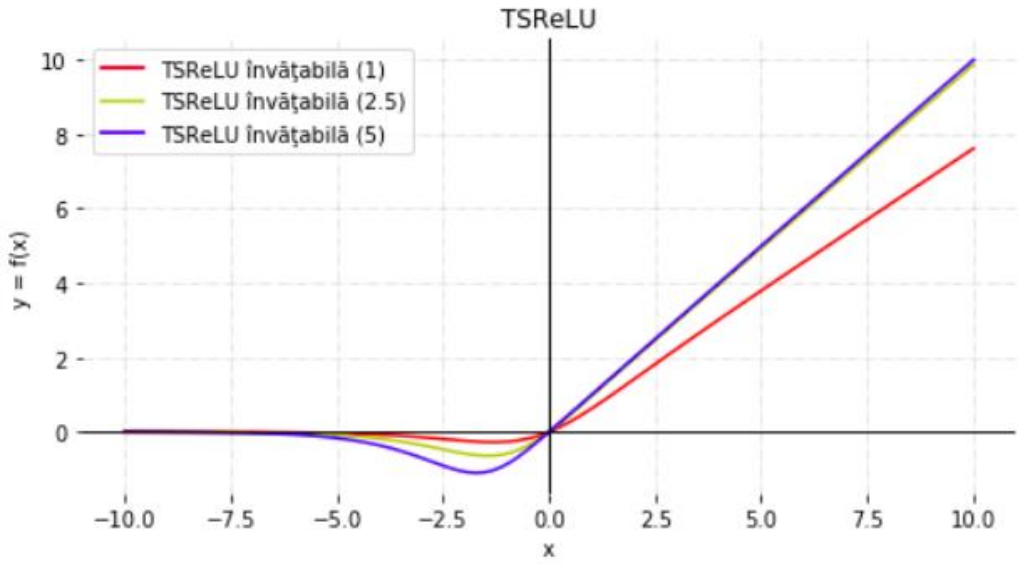


Fig.6.4.2. 1 Funcția de activare TSReLU învățabilă

Atât TSReLU cât și TSReLU învățabilă sunt funcții cu curbe netede, ceea ce înseamnă că ieșirea lor va fi de asemenea lină. Această proprietate oferă o mulțime de avantaje atunci când optimizăm rețelele neuronale pentru a obține o convergență spre funcția de cost minimă, unul fiind dat de valorile negative din cadranul III în jurul valorii -1, care spre deosebire de ReLU nu inhibă în totalitate valorile negative. Graficul acestor propuneri seamănă foarte mult cu graficele funcțiilor de activare Swish și Mish, lucru care denotă că și funcțiile mele propuse dețin și ele proprietățile funcțiilor Swish și Mish.

6.4.3. Funcția de activare TBSReLU

Această funcție de activare este dată de combinația dintre funcția ReLU, funcția tangentă și funcția sigmoidă bipolară, funcție pe care denumit-o TBSReLU (**T**anh-**B**ipolar-**S**igmoid-**Relu**). Inițializez această funcție cu 0,5.
Expresia funcției TBSReLU este dată de relația:

$$f\left(x\right) = \mathrm{x}\cdot\tanh(\frac{1-e^{-x}}{1+e^{-x}}) \tag{6.4.3.1}$$

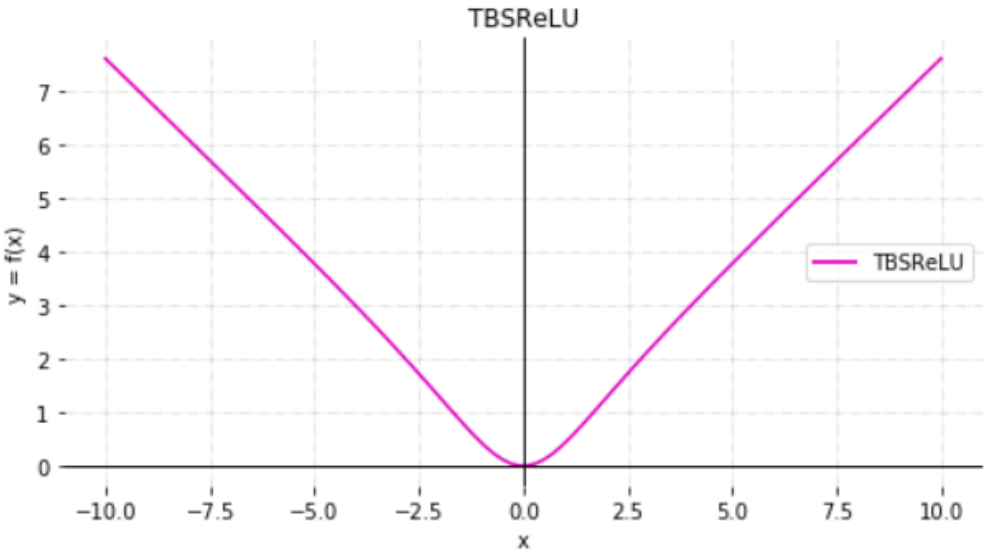


Fig.6.4.3. 1 Funcția de activare TBSReLU

Funcția de activare TBSReLU este o funcție simetrică, centrată în 0, continuă, non-monotonă, nemărginită în partea superioară și mărginită în partea inferioară. Din graficul funcției TBSReLU putem vedea că seamănă cu o altă funcție propusă de mine, funcția TeLU, ambele fiind centrate în 0, totuși diferența majoră fiind dată de simetria funcției TBSReLU. TBSReLU este o funcție mărginită inferior aducând și ea un plus pentru optimizare printr-o regularizare puternică la fel ca funcția TeLU.

6.4.4. Funcția de activare TBSReLU învățabilă

Această funcție de activare este destul de similară cu TBSReLU, singura diferență este un parametru α care este un parametru învățabil. Inițializez acest parametru α = 0,5. În Fig.6.4.4.1.putem vedea funcția de activare TBSReLU învățabilă în care parametrul α este setat cu 1, 2 sau 4 în vederea observării mai clare a modului în care curba se schimbă în funcție de acest parametru α.
Expresia funcției TBSReLU învățabilă este dată de relația:

$$f(x) = \mathrm{x} \cdot \tanh(\alpha \cdot \frac{1-e^{-x}}{1+e^{-x}}) \qquad (6.4.4.1)$$

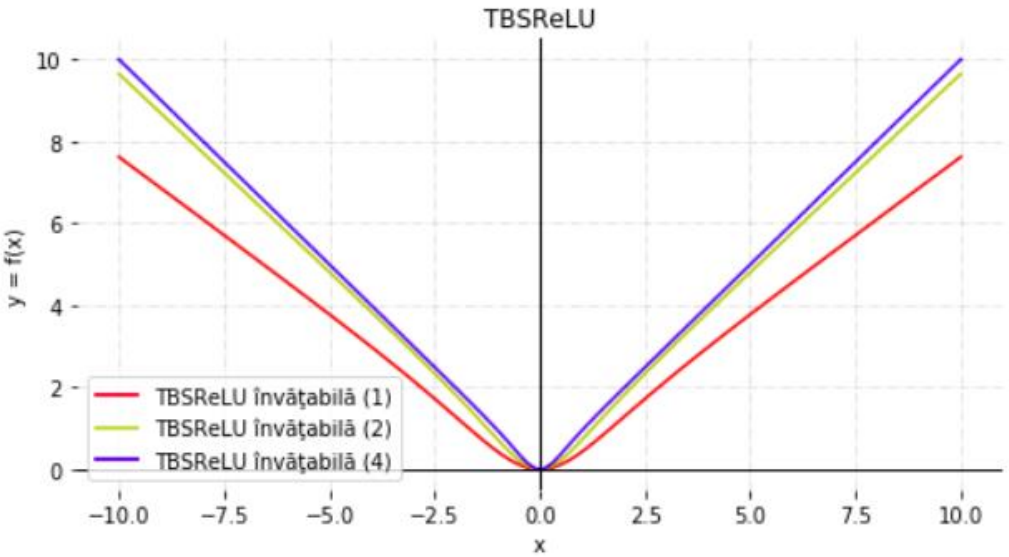


Fig.6.4.4. 1 Funcția de activare TBSReLU învățabilă

Funcția de activare TBSReLU învățabilă are aceleași proprietăți cu funcția de activare TBSReLU neparametrizată. Se poate vedea din graficul său că TBSReLU este o funcție simetrică, centrată în 0, pentru valori mari ale parametrului α funcția se aproprie de axa Oy, în timp ce pentru valori mici se îndepărtează. De asemenea, se poate observa că ambele funcții atât în jurul valorii 1 cât și valorii -1 devin aproape transformări liniare.

6.5. P-Swish: Funcție de activare cu parametri învățabili bazată pe funcția de activare Swish în Învățarea Profundă

Pentru a îmbunătăți performanțele unei rețele neuronale profunde, funcția de activare reprezintă un aspect important căruia trebuie să-i acordăm atenția în mod continuu, motiv pentru care am extins cercetarea în această direcție.

Vă propun o nouă funcție de activare P-Swish (*Parametric Swish*), care este capabilă să aducă îmbunătățiri de performanță pe sarcini de clasificare a obiectelor folosind seturi de date precum CIFAR-10, CIFAR-100, dar vom vedea că am folosit și seturi de date pentru Procesarea Limbajului Natural. Pentru a testa această funcție nouă, am folosit mai multe tipuri de arhitecturi, printre care amintim LeNet-5, NiN și ResNet34. Am testat propunerile în comparație cu funcții populare, cum ar fi sigmoida, ReLU și Swish.

În literatura de specialitate a fost studiat și modul de alegere a unei funcții de activare pentru a atinge gradul dorit de precizie. [220] În legătură cu această abordare, o teoremă generală, conform căreia o funcție de activare mai ușoară conduce la o viteză mai mare în vederea atingerii acestui obiectiv. De obicei, rețelele neuronale preferă neliniaritățile introduse de funcțiile de activare, cum ar fi funcția sigmoidă, funcția tanh, ReLU.

ReLU tinde încă să fie favorizată datorită simplității și fiabilității sale, deoarece îmbunătățirile de performanță ale celorlalte funcții de activare tind să nu rămână atât de robuste și consistente pe diferite arhitecturi și seturi de date.

Funcția de activare a APL s-a dezvoltat din faptul că rețelele neuronale artificiale au, de obicei, o funcție fixă, non-liniară de activare în fiecare neuron. APL este o nouă formă de funcție liniară de activare, care este învățată independent pentru fiecare neuron, folosind GD. GD a devenit un algoritm cheie pentru optimizarea oricărui model de Învățare Profundă, care presupune reducerea iterativă a erorilor prin actualizarea parametrilor în direcția în care scade funcția de cost.

Cu toate acestea, în timp ce funcțiile de activare au fost intens explorate pentru a vedea impactul lor asupra învățării, funcția de activare pe porțiuni a fost mai puțin explorată, așa că atenția mea se concentrează pe dezvoltarea unei astfel de funcții de activare.

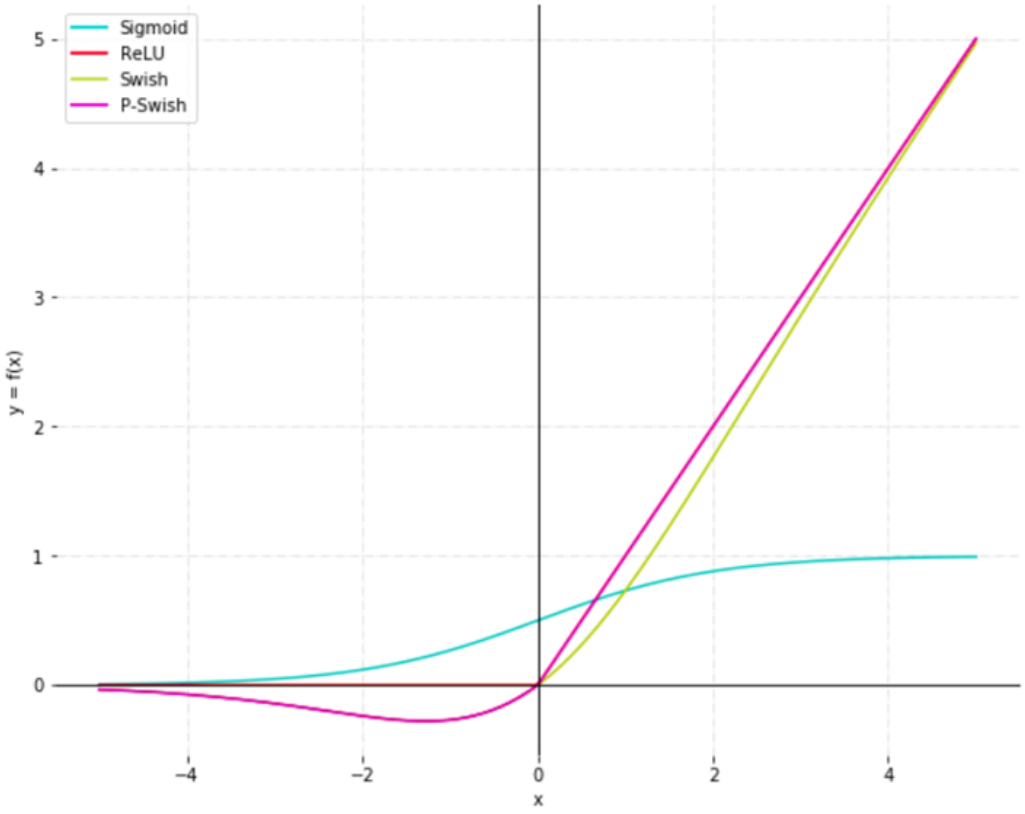


Fig.6.5. 1 Funcții de activare

În acest subcapitol, voi prezenta o nouă funcție de activare pe care o numesc Parametric Swish (*P-Swish*) reprezentată în Fig.6.5.1 cu magenta. Este o funcție de activare derivată din funcția de activare Swish [3]. P-Swish este o funcție de activare pe porțiuni (*piecewise*), este o combinație între ReLU și Swish. Funcția de activare P-Swish poate fi definită ca:

$$f(x) = \begin{cases} \alpha \cdot x \cdot sigmoid(\delta \cdot x) & x \leq \beta \\ x & x > \beta \end{cases} \tag{6.5.1}$$

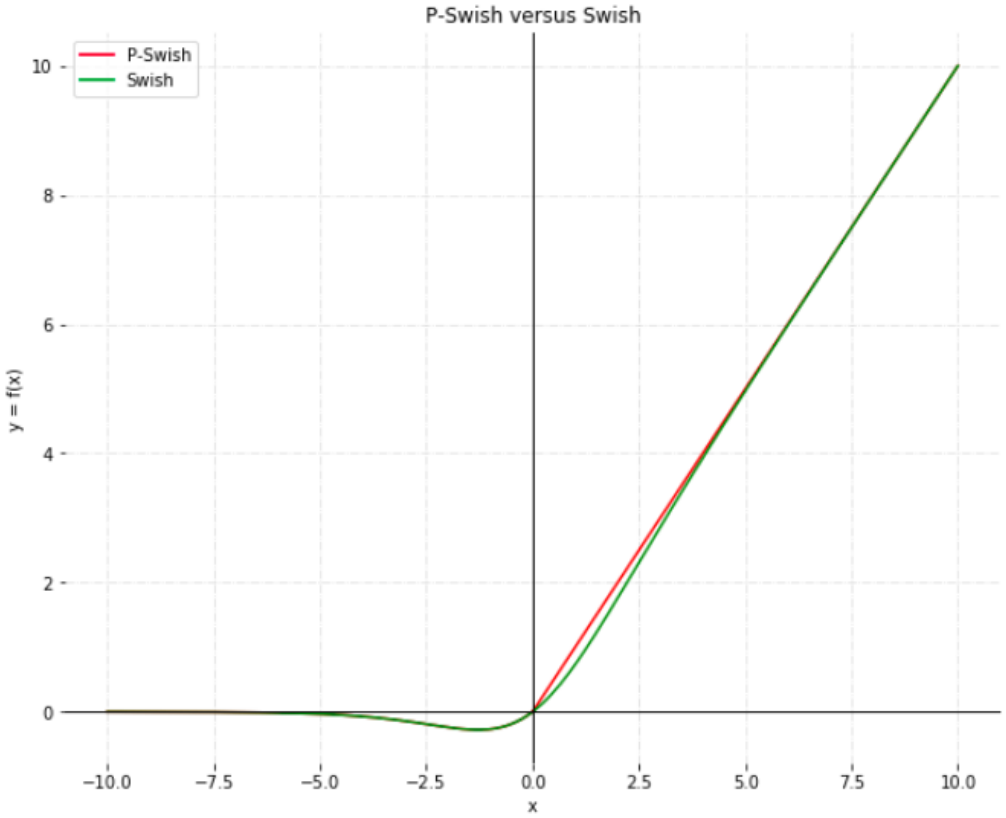


Fig.6.5. 2 P-swish (roșu) pentru $\alpha = 1$, $\beta = 0$, $\delta = 1$ suprapusă peste funcția de activare Swish (verde)

P-Swish este o funcție cu 3 parametri care pot fi predefiniți sau învățabili α , β și δ atunci când $x \leq \beta$, iar pentru $x > \beta$ avem o funcție care derivă din funcția ReLU, dar cu parametrul β învățabil, dar în cazul când $\beta = 0$ funcția devine chiar funcția ReLU. Această propunere beneficiază de proprietățile funcțiilor din care derivă, Swish și ReLU, P-Swish fiind o funcție continuă, non-monotonă, nemărginită în partea de sus și mărginită în partea de jos. Pentru a valida această funcție am folosit mai multe arhitecturi, dar și tehnica Transfer Learning [221] aplicate pe sarcini variate, incluzând chiar și sarcini din domeniul NLP.

Principalele avantaje ale funcției P-Swish sunt simplitatea și precizia îmbunătățită și faptul că nu are probleme cu dispariția gradientului, dar oferă o bună propagare a informației în timpul antrenamentului în Învățarea Profundă.

În ceea ce privește această funcție de activare am definit mai multe posibilități de reprezentare a acestei funcții pe care le-am și testat și anume:

1. P-swish învățabilă când parametrii α și β sunt învățabili și parametrul δ = 1 este constant.
2. P-swish constantă când α, β și δ sunt parametri constanți α = 1, β = 0 și δ=1.
3. P-swish învățabilă parțial când parametrul α învățabil și se inițializează cu 1 și parametrii β și δ sunt constanți, β = 0, δ =1.
4. P-swish învățabilă parțial când parametrul α învățabil și se inițializează cu 1

parametru β constant β = 0 și parametrul δ învățabil și se inițializează cu 1.

Am testat funcția de activare P-Swish și pe o sarcină de segmentare semantică folosind o arhitectură de rețea complet conectată tip U-Net (Fully Connected Convolutional Network - U-Net).

6.6. Noi funcții de activare bazate pe funcția de activare TanhExp în Învățarea Profundă

Propunerea mea constă într-o scurtă sinteză a celor mai frecvente și recente funcții de activare pentru a vedea modul în care funcția de activare ce impact are într-un model de Învățare Profundă. În acest scop, voi testa funcțiile populare pe modelele de Învățare Profundă și folosind informațiile obținute pentru a vedea care activare este cea mai potrivită pentru a crește performanța în funcție de tipul sarcinii alese. Am introdus trei funcții noi de activare, care aduc îmbunătățiri de performanță pe sarcini de clasificare a obiectelor folosind seturi de date precum MNIST, Fashion-MNIST, CIFAR-10, CIFAR-100, dar vom vedea că am folosit și un set de date pentru detectarea anomaliilor în serii de timp. Am folosit ca arhitecturi LeNet-5 și AlexNet pentru CIFAR-10 și CIFAR-100, pentru MNIST și Fashion-MNIST o arhitectură personalizată ca la implementarea funcției TanhExp și o arhitectură personalizată pentru o sarcină aplicată pe date tip serii de timp. Am comparat propunerile mele cu ReLU, *tanh* și TanhExp.

Chiar și în 2020, funcțiile de activare reprezintă o zonă de interes datorită rolului pe care îl joacă într-o rețea neuronală. [222] Funcțiile de activare alimentează ieșirea unui neuron, mapează datele liniare într-un interval neliniar și, prin urmare, introduc non-liniaritatea de care sistemul în ansamblu are nevoie pentru a învăța date neliniare.

De-a lungul timpului, în literatura de specialitate, găsim diferite abordări cu privire la modul de alegere a unei funcții de activare. [223] Acoperim funcțiile de activare tradiționale precum *tanh* și ReLU, dar și o funcție mai recentă, cum ar fi TanhExp (*Tanh Exponential*) [222]. Pornind de la ideea modului de funcționare a creierul uman atunci când este alimentat simultan cu o mulțime de informații, în care se face o clasificare binară a informației în „relevant” sau „irelevant”. Informațiile „irelevante” sunt de cele mai multe ori doar zgomot și acestea afectează procesul de învățare. Această abordare ne ajută să reducem volumul mare de informații și să venim cu o soluție la sarcina noastră. În mod similar acționează și funcția de activare. Funcția de activare ajută rețeaua neuronală să utilizeze informații importante și să suprime datele irelevante. Un alt aspect important este tipul de funcție, deoarece neuronul nu poate învăța numai cu o funcție de activare liniară. O funcție neliniară permite învățarea din diferența de eroare.

În studiul meu, folosesc o funcție de activare tradițională *tanh* [46], care vine să rezolve dezavantajele funcției sigmoide [64]. Într-un studiu recent, *tanh* a depășit performanțele în aplicații din Învățarea Automată, dovedindu-se per ansamblu că această funcție conduce la o performanță mai bună. [224]

În cadrul acestei propuneri prezint eficiența funcțiilor noi pe mai multe seturi de date din domeniul Vederii Artificiale [225] [226] dar și în cadrul unei sarcini de detecție a anomaliilor [227].

Datele privind anomaliile apar în probleme, cum ar fi frauda bancară [228], fraude în domeniul sănătății [229], erori în text [230]. În acest scop, am folosit o

tehnică numită autoencoder variațional (*Variational Autoencoder* - VAE) [231] [232] pentru a proiecta rețelele neuronale. Am folosit VAE deoarece această tehnică este potrivită pentru învățarea spațiilor latente bine structurate, ceea ce ne ajută să avem direcții specifice pentru a codifica o axă care are o variație semnificativă în date. VAE utilizează parametri medie și varianță pentru a preleva aleatoriu un element al distribuției și decodează acel element înapoi la intrarea originală. (Fig.6.1. 1) Comportamentul aleatoriu al acestui proces este robust și forțează spațiul latent să codifice reprezentări semnificative peste tot: fiecare punct eșantionat în spațiul latent este decodat la o ieșire validă.

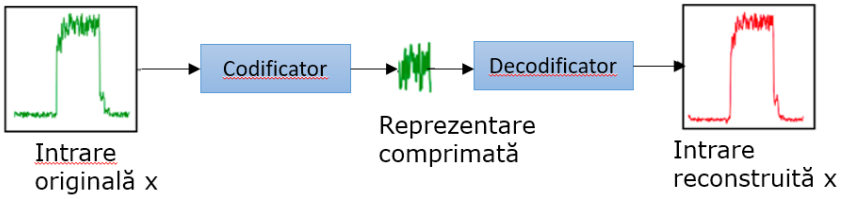


Fig.6.6. 1 Un autoencoder în serii de timp

Anomaliile sunt întâlnite, de asemenea, ca valori aberante, zgomot, abateri sau excepții. [233] De-a lungul timpului, multe definiții au fost date conceptelor de anomalie, dar cea mai potrivită pare a fi definiția care spune că "valorile aberante pot fi văzute ca observații care nu au comportamentul așteptat". [234] Tehnicile de detectare a valorilor anterioare în datele seriilor temporale variază în funcție de tipul de date de intrare, tipul anomaliei și natura metodei, pe care o vom vedea în datele experimentale. Seria mea temporală de intrare este o serie temporală univariată, deoarece avem o singură intrare (prețul).

Funcția de activare tangetă exponențială (*TanhExp*) [222] este o funcție de activare recentă, se caracterizează prin netezime și viteză mare de convergență pentru rețelele neuronale ușoare. Este destul de similară cu funcția Mish [99], o altă funcție de activare recentă. TanhExp este definită după cum urmează:

$$f(x) = x \cdot \tanh(e^x) \tag{6.6.1}$$

Din ecuația 6.6.1, putem vedea că TanhExp este o combinație de funcții: ReLU, tanh și argumentul exponențial.

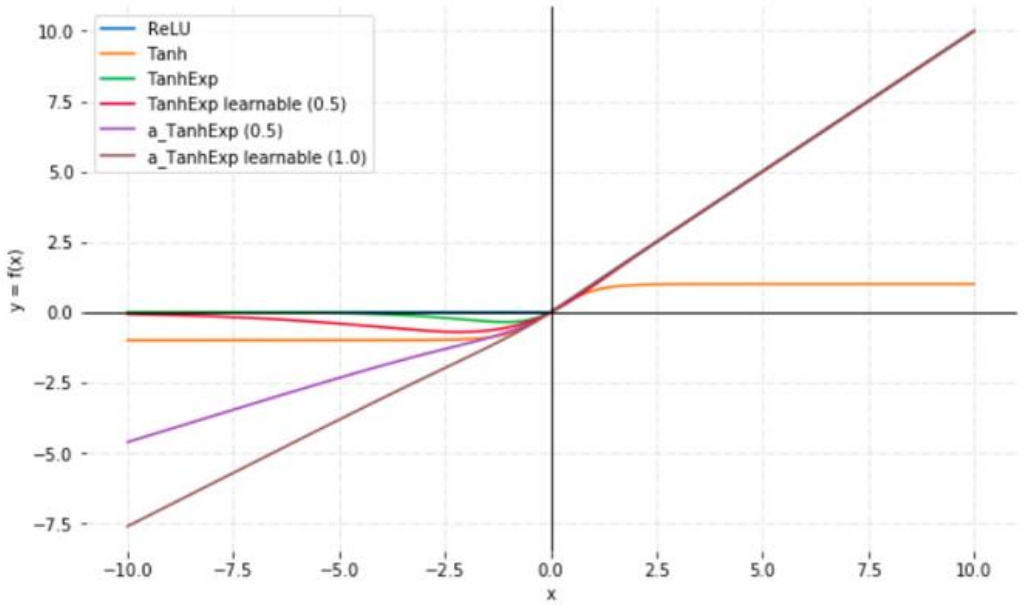


Fig.6.6. 2 Funcții de activare noi versus curente

Propunerile mele au constat în dezvoltarea de funcții de activare noi care derivă din TanhExp, dar diferența principală constă în utilizarea unui parametru învățabil (TanhExp learnable în Fig. 6.6.2), care lipsește în proiectarea funcției TanhExp. Funcțiile noi propuse în Fig. 6.6.2 sunt TanhExp learnable, a_tanhExp și tanhExp learnable.

6.6.1. Funcția de activare TanhExp învățabilă

Designul funcției TanhExp învățabilă (*TanhExp learnable*) a fost inspirat din funcția TanhExp. Funcția am implementat-o în Tensorflow 2.2.4, folosind Keras API. Această propunere beneficiază de proprietățile TanhExp, fiind o funcție continuă, non-monotonă, nemărginită în partea de sus și mărginită în partea de jos. Pentru a valida această funcție am folosit mai multe arhitecturi pe care le-am mapat pe diferite tipuri de sarcini.
Funcția de activare TanhExp învățabilă este destul de similară cu TanhExp, dar care aduce ca noutate parametrul său învățabil, care este capabil să aducă îmbunătățiri de precizie. Graficul său poate fi văzut în Fig.6.6.2 reprezentat cu culoarea roșu pentru un parametru α = 0.5. Ecuația funcției TanhExp învățabilă este dată de următoarea relație:

$$f(x) = x \cdot \tanh(e^{\alpha \cdot x}) \tag{6.6.1.1}$$

Unde α este un un parametru care poate fi predefinit sau învățabil.

6.6.2. Funcția de activare a_TanhExp

Funcția de activare a_TanhExp este o funcție de activare parametrică inspirată din funcția de activare TanhExp. Parametrul *a* controlează concavitatea minimelor globale ale funcției de activare, atunci când *a* = 0 acestă funcție este chiar funcția de activare TanhExp. Variația unei scări negative reduce concavitatea și pe scara pozitivă mărește concavitatea. *a* este introdus pentru a combate scenariile de gradient "mort" datorită minimelor globale ascuțite ale funcției de activare TanhExp. Ecuația sa este definită după cum urmează:

$$f(x) = x \cdot \tanh(e^x + a) \tag{6.6.2.1}$$

a_TanhExp este o funcție continuă, monotonă, nemărginită. Fiind o funcție complet nemărginită evită un antrenament lent, proprietate foarte dorită pentru funcțiile de activare.

6.6.3. Funcția de activare a_TanhExp învățabilă

Funcția de activare a_TanhExp învățabilă este destul de similară cu funcția de activare a_TanhExp. În acest caz, parametrul *a* devine un parametru care poate fi învățat. a_TanhExp învățabilă are aceleași proprietăți ca funcția a_TanhExp, cu mențiunea că paramtrul *a* cu cât scade se aproprie de axa Ox, iar cu cât crește se aproprie de axa Oy.

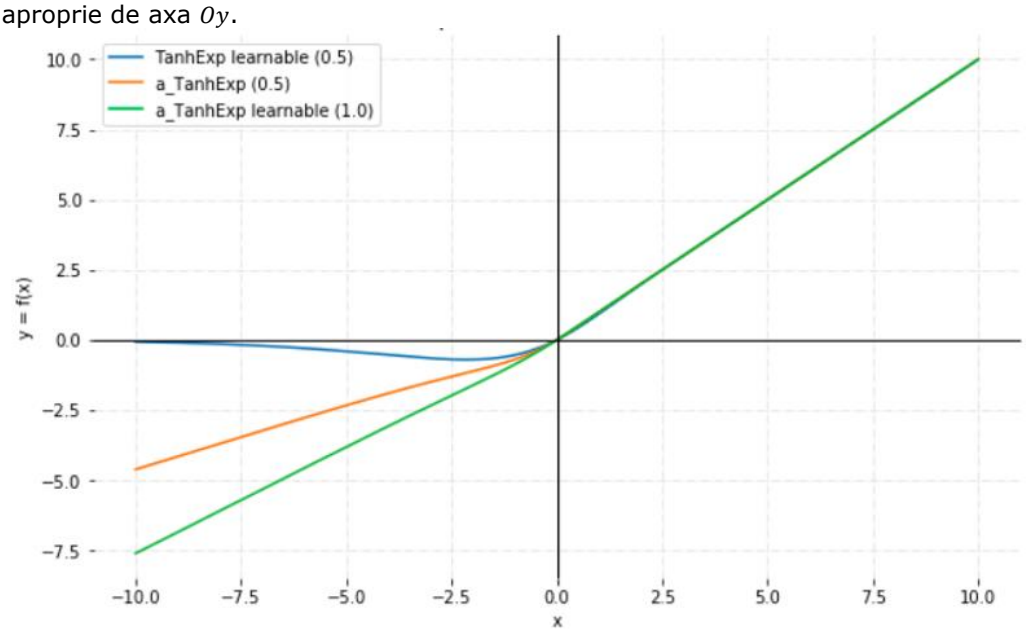


Fig.6.6.3. 1 Funcțiile de activare propuse: TanhExp învățabilă (*TanhExp learnable*), a_TanhExp și a_TanhExp învățabilă (a_TanhExp learnable)

În Fig. 6.6.3.1 a_TanhExp învățabilă este reprezentată cu culoarea verde, fiind inițiazat parametrul *a* cu 1, urmând să fie învățat în timpul antrenării rețelei.

Funcțiile a_TanhExp și a_TanhExp învățabilă au proprietatea de aproximare a identității în apropierea originii, proprietate care conferă rețelei neuronale posibilitatea să învețe eficient atunci când ponderile sale sunt inițializate cu valori aleatorii mici.

6.7. Dezvoltarea de noi funcții de activare în detecția anomaliilor în serii de timp cu autoencoder LSTM

Propunerea mea constă în dezvoltarea a două funcții de activare noi în detectarea anomaliilor în seriile de timp, acestea având capacitatea să reducă funcțiile de cost pe setul de validare. Pentru a atinge acest scop, am folosit un autoncoder LSTM. Punctul cheie din propunerea mea este dat de parametrul care poate fi învățat. Am comparat propunerea mea cu ReLU, *tanh* și TaLu. De asemenea, noutatea acestei propuneri constă în luarea în considerare a comportamentului pe porțiuni al unei funcții de activare pentru a crește performanța unei rețele neuronale în Învățarea Profundă.

Detecția anomaliilor [227] poate fi definită ca o sarcină de identificare a evenimentelor suspecte sau rare în setul de date, tip de sarcină abordat și în cadrul subcapitolului anterior. Această problemă este întâlnită în domenii diverse, cum ar fi detectarea fraudelor bancare [228], detecția bolilor în imagistica medicală [229], erori în secvențele scrise de text [230], anomalii în date [235], detectarea traficului ilegal [236], detectarea deteriorării retinei [237], detectarea fraudei cibernetice [238], detectarea anomaliilor în seturi de date mari în Internet of Things (IoT) [239].

De-a lungul timpului, au fost propuse mai multe abordări supervizate [240] și nesupervizate [241] pentru detecția anomaliilor. Prin aceste abordări menționăm: rețelele Bayesiene [242], analiza clusterelor [243], SVM-urile dintr-o clasă [244] și rețelele neuronale [245].

Sursa anomaliilor poate fi cauzată de erori în date, dar de cele mai multe ori indică un proces derivat nou, necunoscut anterior. În ultimii ani, putem vedea că domeniul Învățării Automate a crescut prin multiplicarea rețelelor neuronale profunde, cu rezultate fără precedent în domeniile de aplicare multivariate. Metodele de Învățare Profundă (Fig.6.7.1) depășesc [246] metodele tradiționale de Învățare Automată atunci când folosim seturi de date mari [247]. De asemenea, s-a demonstrat că unele metode tradiționale au fost complet suprimate prin metode de Învățare Profundă. [238] [248] [249]

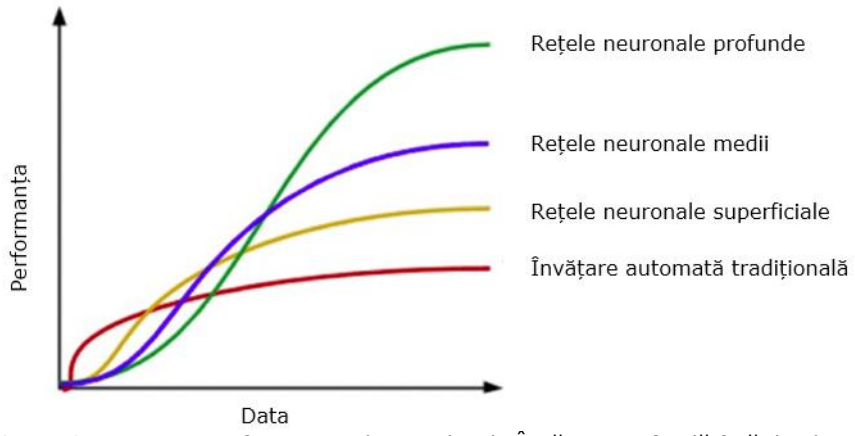


Fig.6.7. 1 Compararea performanței algoritmilor de Învățare Profundă față de algoritmii tradiționali

Identificarea anomaliilor scot la suprafață idei interesante despre date și adesea acestea transmit informații valoroase despre date. Prin urmare, detecția anomaliilor este considerată un subiect esențial în diferite sisteme de luare a deciziilor. [250]

Am fost motivată în alegerea unei tehnici de Învățare Profundă pentru detecția anomaliilor, deoarece această tehnică poate învăța caracteristicile ierarhice de eliminare din date. Această proprietate oferă o capacitate automată de învățare a caracteristicilor care elimină necesitatea identificării manuale a caracteristicilor.
Limita dintre datele normale și datele anormale în cele mai multe cazuri nu este stabilită cu precizie. Această lipsă de delimitare bine definită constituie o provocare atât pentru algoritmii tradiționali, cât și pentru cei din Învățarea Profundă [251].

În al doilea rând, arhitectura rețelei neuronale profunde depinde de cât de complexă este învățarea relațiilor de caracteristici ierarhice în cadrul datelor de intrare brute de înaltă dimensiune. [2]

În acest studiu, am făcut detecția anomaliilor a seriilor de timp cronologice. [252-257] Seriile cronologice reprezintă date colectate continuu în timp. Datele seriilor de timp pot fi clasificate în serii de timp univariate și multivariate. În cazul seriilor temporale univariate, doar o singură variabilă sau caracteristică variază în timp. În cazul meu, am o singură caracteristică (preț) care se modifică în timp, datele sunt colectate zilnic. În cazul unei serii temporale multivariate, avem mai multe caracteristici, care se schimbă în timp.

Până în prezent, au fost propuse multe modele de Învățare Profundă pentru detecția anomaliilor în cadrul seriilor de timp univariate și multivariate. Modelele de tip autoencoder pentru rețelele neuronale încearcă să învețe reprezentarea datelor din intrarea sa. Practic, vrem să facem un codificator eficient care este definit în mod ideal de mai puțini parametri pentru a utiliza mai puțină memorie. Codificatorul ar trebui să poată reproduce o ieșire similară cu intrarea originală. Cu alte cuvinte, constrângem modelul să învețe cele mai importante caracteristici din date folosind cât mai puțini parametri.

Pasul principal pe care l-am urmat pentru detecția anomaliilor utilizând un codificator automat a fost stabilirea unui prag, care l-am calculat prin diferența dintre datele reconstruite și cele originale și, dacă eroarea de reconstrucție este peste acest prag (valoarea maximă), decid că aceste date constituie o anomalie.

O proprietate particulară a seriilor de timp este dată de caracterul temporal al datelor, așa că trebuie să ținem cont de acest aspect, iar folosind metoda LSTM putem face acest lucru [67]. LSTM este utilizat pe scară largă în rețelele neuronale recurente (RNN) [196].

Arhitectura LSTM a venit pentru a rezolva problema memoriei pe termen scurt pe care o are o arhitectură simplă RNN. A fost concepută pentru a atenua problema dispariției și exploziei gradientului. Fiecare LSTM menține un vector care descrie starea celulei și de fiecare dată următorul LSTM poate alege să citească din el sau să reseteze celula. Fiecare LSTM are 3 porți: poarta de intrare, poarta de uitare [258] [259] și poarta de ieșire, fiecare dintre acestea este o poartă binară. Poarta de intrare controlează dacă celula de memorie este actualizată, poarta de uitare este responsabilă pentru resetarea la 0, iar poarta de ieșire controlează dacă informațiile stării curente sunt făcute vizibile. Notația *x* (Fig. 6.7.2) reprezintă operația de multiplicare care decide ce date vor merge mai departe. De asemenea, putem vedea că avem trei funcții sigmoide și o funcție *tanh*. Sigmoida, datorită intervalului său, decide ce informații trebuie să uite celula folosind 0 și 1 pentru informațiile care decid să rămână. Și utilizarea *x* înseamnă actualizarea stării celulei renunțând la elementele care au fost decise să se uite în operația anterioară. Funcția *tanh* având un interval centrat în zero, lucru care conduce la o distribuție foarte bună a gradienților, ceea ce înseamnă că permite producerea de informații despre starea celulei pe mai mult timp fără să apară problema dispariției sau exploziei gradientului.

Unul dintre cele mai remarcabile avantaje ale LSTM constă în utilizarea porților care conduce la ignorarea anumitor intrări. De asemenea, LSTM poate menține unele valori care sunt protejate de porți și pe care nu le trec printr-o funcție de activare. [260] Așa cum am spus mai devreme, LSTM este utilizat pe scară largă, chiar și pentru recunoașterea vocii pe Google Voice [261], recunoașterea scrisului de mână [262], traducerea textului [263] [264] [265]. Deși LSTM a fost atât de utilizat pe scară largă, studiile continuă cu dezvoltarea de noi arhitecturi care depășesc performanța LSTM, și anume RNN cu ferestre de timp continue [266]. De asemenea, de-a lungul timpului multe arhitecturi au derivat din LSTM, este și cazul GRU (*Gated Recurrent Unit*) [68], diferența vizibilă față de LSTM constă în lipsa porții de ieșire în arhitectura GRU.

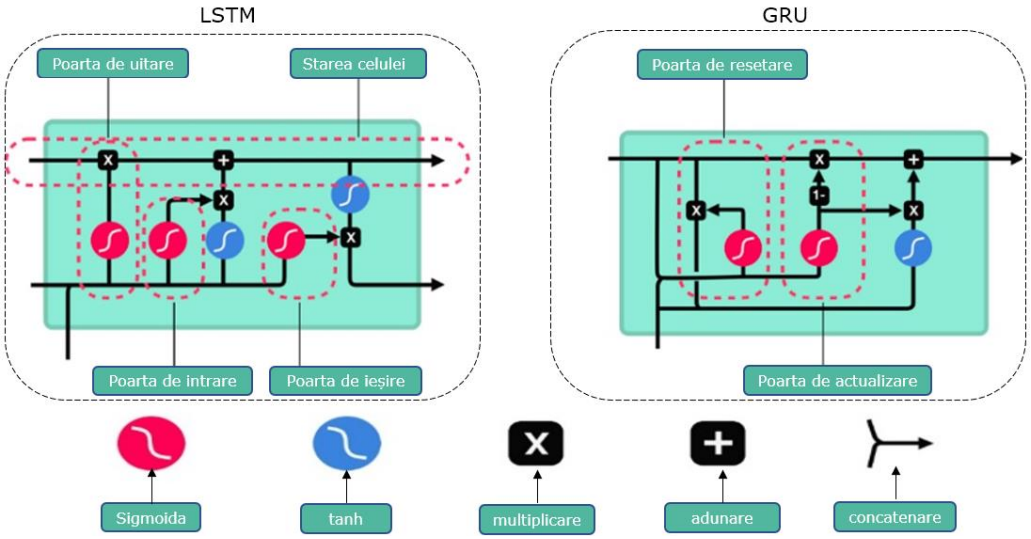


Fig.6.7. 2 Arhitectura LSTM versus arhitectura GRU

Propunerile mele au fost inspirate din funcția de activare TaLu (*tangent linear unit*) [267] care este o nouă funcție de activare bazată pe *tanh* și ReLU, dezvoltată pentru rețelele neuronale și s-a dovedit că dă rezultate mai bune decât funcțiile ReLU, LReLU și ELU. Ecuația funcției TaLu este dată de relația:

$$f(x) = y_i = \begin{cases} x_i & \text{if } x_i \geq 0 \\ \tanh(x_i) & \text{if } \alpha < x_i < 0 \\ \tanh(\alpha) & \text{if } x_i \leq \alpha \end{cases} \tag{6.7.1}$$

Unde α este un parametru fix cu valori negative. A fost testat de la -0,50 până la -0,01. Din ecuația sa, putem vedea că această funcție este o combinație de *tanh* și ReLU, fiind un tip de funcție de activare pe porțiuni la fel ca și funcția APL (**A**daptive **P**iecewise **L**inear units) [120].

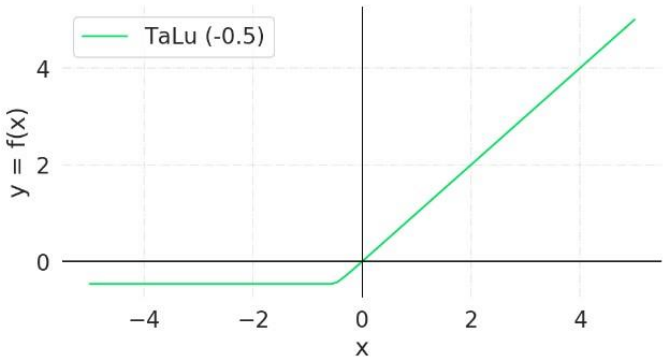


Fig.6.7. 3 Funcția de activare TaLu

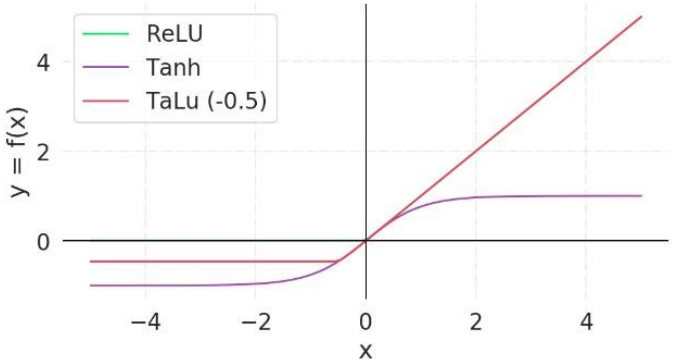


Fig.6.7. 4 Funcția de activare TaLu versus funcțiile ReLU și tanh

După cum s-a văzut, funcția TaLu este o combinație de două funcții, fiind o funcție pe porțiuni. Propunerea mea constă în dezvoltarea unor funcții noi care derivă din TaLu, dar principala diferență constă în utilizarea proprietății care le conferă noilor funcții capacitatea să fie învățabile, proprietate care lipsește în proiectarea funcției TaLu.

6.7.1. Funcția de activare Talu învățabilă

Talu învățabilă (*Talu learnable*) este o funcție de activare care este destul de similară cu TaLu, dar aduce ca noutate parametrul său învățabil. Ecuația funcției de activare Talu învățabilă este dată de relația:

$$f(x) = Talu \text{ learnable (x)} = y_i = \begin{cases} x_i & \text{if } x_i \geq 0 \\ \tanh(x_i) & \text{if } \alpha < x_i < 0 \\ \tanh(\alpha) & \text{if } x_i \leq \alpha \end{cases} \quad (6.7.1.1)$$

Unde α este un parametru învățabil.

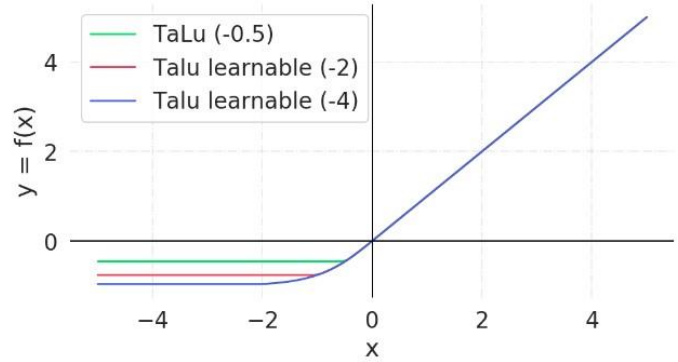


Fig.6.7.1. 1 Funcția TaLu versus funcția propusă - Talu învățabilă

În figura de mai sus putem vedea funcția propusă Talu învățabilă (*Talu learnable*) evidențiată în roșu pentru α = -2 și α = -4, suprapusă pe funcția TaLu evidențiată în verde. Designul funcției Talu învățabilă a fost inspirat de la funcția TaLu. Am

implementat funcția în Tensorflow 2.2.4, folosind Keras API și am inițializat α cu -0,75. Această funcție propusă numită Talu învățabilă este o funcție continuă, monotonă și nemărginită în partea superioară și mărginită în partea inferioară, fiind un indiciu că posedă o puternică capacitate de regularizare la începutul procesului de antrenare. Pentru validarea acestei funcții am folosit o arhitectură LSTM existentă aplicată pe o sarcină de detecție a anomaliilor.

6.7.2. Funcția de activare P_Talu învățabilă

O a doua propunere este dată de funcția de activare numită P_Talu învățabilă (*P_Talu learn*) care este similară cu Talu învățabilă, dar de această dată introduc doi parametri care pot fi învățabili. Ecuația sa este definită după cum urmează:

$$f(x) = P\_Talu(\mathrm{x}) = y_i = \begin{cases} x_i & \text{if } x_i \geq b \\ \tanh(x_i) & \text{if } a < x_i < b \\ \tanh(a) & \text{if } x_i \leq a \end{cases} \quad (6.7.2.1)$$

Unde α și *b* sunt doi parametri învățabili.

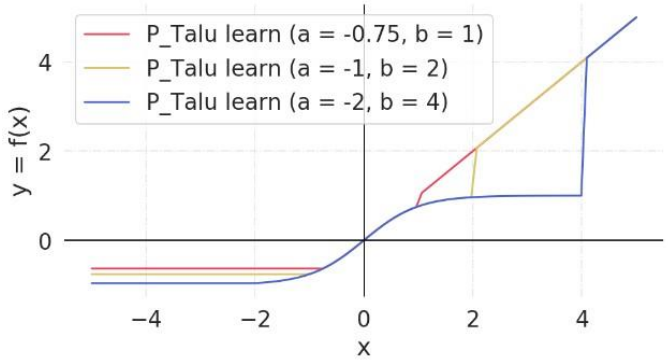


Fig.6.7.2. 1 Funcția propusă P_Talu învățabilă

Din figura 6.7.2.1, putem vedea că traiectoria sa se schimbă în funcție de valorile parametrilor setați. Am inițializat a = -0,75 și b = 1,0 înainte de antrenament, iar în faza de antrenament, aceștia se actualizează corespunzător în corelație cu actualizarea ponderilor rețelei pentru a conduce la o eroare finală cât mai mică. Abordarea mea s-a bazat pe strategii de ultimă generație, care constau în funcții de activare cu un singur parametru scalabil care se învață în timpul antrenamentului. [122] Funcția P_Talu învățabilă este o funcție continuă, monotonă, nemărginită la partea superioară și marginită la partea inferioară.

6.8. Soft Clipping Mish - o nouă funcție de activare

În cadrul acestui studiu propun o nouă funcție de activare, numită **Soft Clipping Mish** (**SC Mish**). După cum ne dăm seama din numele funcției, aceasta derivă dintr-o funcție recentă, apărută în 2019, numită Mish, funcție pe care am prezentat-o în cadrul capitolului 2 la secțiunea funcții de activare. Funcția Mish este o funcție compozițională, fiind alcătuită din 3 funcții: ReLU, tanh și Softplus.

Noua funcție de activare **Soft Clipping Mish** este și ea o funcție compusă și se definește astfel:

$$f(x) = max(0, x \cdot tanh(softplus(x))) \quad (6.8.1)$$

După cum putem vedea din relația funcției **Soft Clipping Mish** (6.8.1), la fel ca funcția ReLU este o funcție de prag care nu face altceva decât să ignore valorile de intrare negative pe axa Ox, iar în partea pozitivă a axei Ox aceasta va păstra forma funcției Mish. Am denumit această functie Soft Clipping deoarece renunță la partea din cadranul III a funcției Mish, lucru care se poate vedea în următoarea figură:

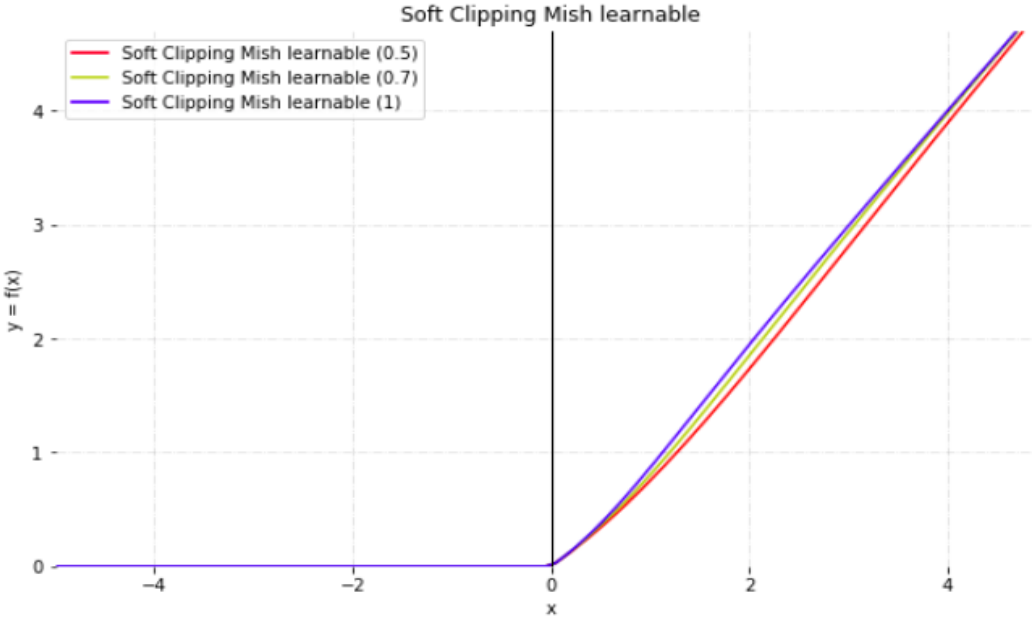


Fig.6.8. 1 Funcția propusă Soft Clipping Mish învățabilă

Din Fig. 6.8.1 se poate vedea că funcția SC Mish (Soft Clipping Mish) este cazul particular pentru funcția Soft Clipping Mish învățabilă (*Soft Clipping Mish learnable*) atunci când parametrul învățabil $\alpha = 1$. SC Mish este o funcție continuă, monotonă, mărginită în partea inferioară și nemărginită în partea superioară. Ia valori în domeniul $[0, +\infty)$.

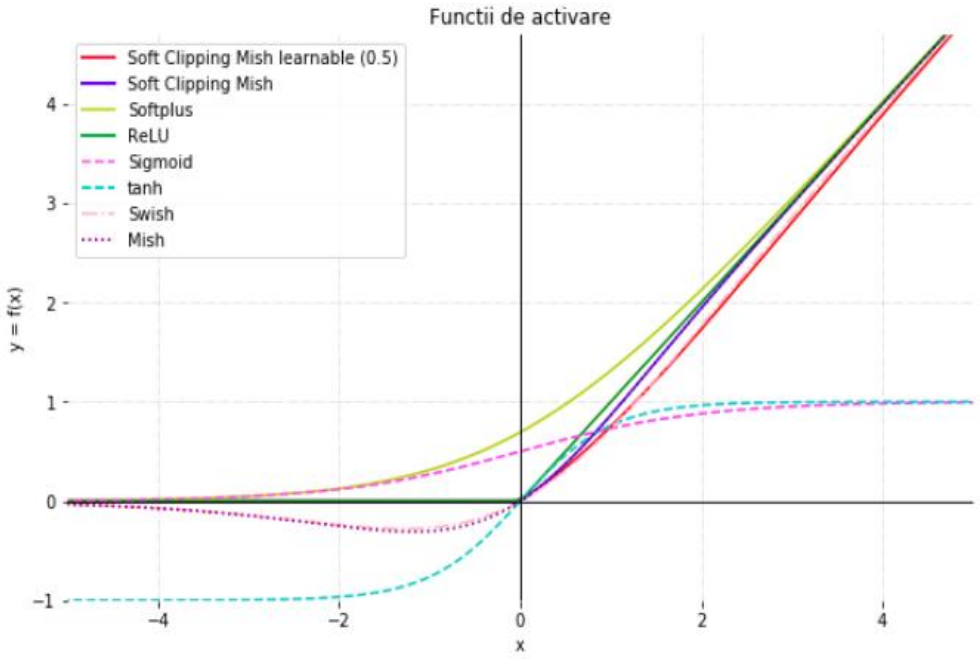


Fig.6.8. 2 Funcții de activare

De asemenea pornind de la această propunere, am propus și **Soft Clipping Mish învățabilă** (**Soft Clipping Mish learnable – SCL Mish**) care practic derivă din **Soft Clipping Mish**, diferența majoră constând în parametrul α învățabil, oferind mai multă flexibilitate rețelei.

$$f(x) = max(0, x \cdot tanh(softplus(\alpha \cdot x))) \tag{6.8.2}$$

SCL Mish are aceleași proprietăți cu funcția SC Mish, este de remarcat faptul că atunci când parametrul α ia valori mici, funcția se îndepărtează de axa Oy (Fig. 6.8.1 $\alpha = 0.5$ și $\alpha = 0.7$).

6.9. Concluzii

Acest capitol este cel mai important capitol din cadrul tezei, acesta a avut drept scop introducerea și crearea de noi funcții de activare. Un total de opt studii realizate în acest sens, conținând un total de 14 funcții de activare propuse:

- *2 funcții: TeLU* și *TeLU învățabilă* introduse prin studiul *„TeLU: o nouă funcție de activare pentru Învățarea Profundă"*
- *4 funcții: TSReLU, TSReLU învățabilă, TBSReLU* și *TBSReLU învățabilă* introduse prin studiul *„Îmbunătățirea preciziei rețelelor neuronale profunde prin dezvoltarea de noi funcții de activare"*

- *1 funcție: P-Swish*, introdusă prin studiul *„P-Swish: Funcție de activare cu parametri învățabili bazată pe funcția de activare Swish în Învățarea Profundă"*
- *3 funcții: TanhExp învățabilă, a_TanhExp* și *a_TanhExp învățabilă* introduse prin studiul *„Noi funcții de activare bazate pe funcția de activare TanhExp în Învățarea Profundă"*
- *2 funcții: Talu învățabilă* și *P_Talu învățabilă* introduse prin studiul *„Dezvoltarea de noi funcții de activare în detecția anomaliilor în serii de timp cu autoencoder LSTM"*
- *2 funcții: Soft Clipping Mish* și *Soft Clipping Mish învățabilă* introduse prin studiul *„Soft Clipping Mish - o nouă funcție de activare".*

Am arătat că funcțiile propuse TeLU și TeLU învățabilă pot da rezultate mai bune decât ReLU, Mish și TanhExp datorită proprietății lor de nemărginire superioară, dar mărginită la partea inferioară (regularizare puternică) și a centrării în zero.

În cazul optimizării rețelei recomand propunerile mele *TSReLU* și *TSReLU învățabilă* care sunt funcții netede, ceea ce înseamnă că ieșirea lor va fi de asemenea netedă, proprietate care conduce la o convergență spre funcția de cost minimă. Unul din avantajele acestor propuneri este dat de valorile negative din cadranul III în jurul valorii -1, care spre deosebire de ReLU nu inhibă în totalitate valorile negative. Graficul acestor propuneri seamănă foarte mult cu graficele funcțiilor de activare Swish și Mish, lucru care denotă că și funcțiile propuse moștenesc proprietăți de la funcțiile Swish și Mish.

S-a observat că proprietatea mărginirii inferioare poate fi un plus datorită unei regularizări puternice, funcțiile care se apropie de zero într-o limită la $-\infty$ sunt foarte bune pentru regularizare, deoarece intrările negative mari sunt eliminate. Acest lucru joacă un rol important la începutul antrenamentului, când intrările mari de activare negativă sunt comune. Această proprietate este întâlnită la Softplus, ReLU, Swish și P-Swish. Recomandăm funcția propusă P-Swish datorită faptului că nu este monotonă ceea ce conduce la creșterea expresivității unei intrări și îmbunătățește procesul gradienților.

Pentru o stabilitate a antrenării cu metoda gradientului se recomandă ca domeniul funcției de activare să fie finit, astfel că prezența tiparelor va afecta semnificativ doar ponderile limitate.

În cazul domeniului infinit ca în cazul funcțiilor propuse a_TanhExp și a_TanhExp învățabilă, caz mai dorit, deoarece favorizează antrenarea mai eficientă a rețelei astfel că prezența tiparelor va afecta semnificativ aproape toate ponderile. Parametrul predefinit sau învățabil din componența acestor funcții este introdus pentru a evita gradientul "mort" datorită minimelor globale ascuțite ale funcției de activare TanhExp.

Atunci când se face o inițializare cu valori aleatorii mici recomandăm funcțiile propuse a_TanhExp și a_TanhExp învățabilă care datorită proprietății de aproximare identitate în apropierea originii conduce rețeaua să învețe eficient, care dau rezultate mai bune ca ReLU, *tanh* și TanhExp.

Când este nevoie de o regularizare puternică la începutul procesului de antrenare recomandăm funcțiile propuse Talu învățabilă, P_Talu, SC Mish și SCL Mish, fiind superioare în anumite scenarii depășind funcțiile ReLU, TaLu, *tanh* și Mish.

7. Rezultate experimentale

7.1. Modificarea dinamică a funcției de activare folosind algoritmul propagarea înapoi în rețelele neuronale artificiale – Partea experimentală

În acest studiu am luat în considerare o funcție de activare de tip sigmoidă în contextul unei arhitecturi a rețelei neuronale fără straturi ascunse. Studiul a fost realizat ținând cont de:

- Valoarea parametrului β a funcției de activare care scade dacă eroarea crește în timpul unei epoci, respectiv crește dacă eroarea scade. Direcția se poate inversa și este definită la începutul antrenamentului.
- Procentul acestui parametru este stabilit la începutul antrenamentului, fiind definit de utilizator.

În experimente am în vedere scenariul:

- Coeficientul β al funcției de activare scade dacă valoarea erorii crește în timpul unei epoci, coeficientul crește, dacă eroarea scade.
- Procentul modificărilor pentru coeficientul β a fost de 10%, respectiv 5% din valoarea reală în perioada de antrenare.

Pentru a realiza scenariul de mai sus, Clasificatorul Perceptron Multistrat a fost schimbat prin intermediul platformei WEKA versiunea 3.8.3 prin adăugarea unui parametru care determină direcția schimbării valorii β pentru funcția de activare ca un parametru ce se poate modifica. Astfel cei doi parametri sunt:

- modRate care specifică procentul de modificare pentru coeficientul β.
- *variant* care definește direcția de schimbare pentru valoarea β. Este de tip boolean și poate lua două valori astfel:
    - *adevărat* - în cazul în care valoarea parametrului β pentru funcția de activare scade, dacă eroarea crește în timpul unei epoci și,
    - *fals* atunci când valoarea parametrului β scade, iar eroarea crește.

Codul adăugat poate fi vizualizat în următoarea figură:

```
if(m_variant == true){ // varianta 1

    for (int i=0 ; i < m_neuralNodes.length ; i++) {
        crt_node =  m_neuralNodes[i];
        m_sigmoidUnit.setZ(sigm_z[i]);
        crt_err = crt_node.errorValue(true);
        if(crt_err < prev_err_per_neuron[i] )
            m_sigmoidUnit.setZ(m_sigmoidUnit.getZ()+(m_sigmoidUnit.getZ()*getModRate()));
        if(crt_err >  prev_err_per_neuron[i] )
            m_sigmoidUnit.setZ(m_sigmoidUnit.getZ()-(m_sigmoidUnit.getZ()*getModRate()));

        prev_err_per_neuron[i] = crt_err;
    }
}

 if(m_variant == false){ // varianta 2

     for(int i=0 ; i < m_neuralNodes.length ; i++) {
        crt_node =  m_neuralNodes[i];
        m_sigmoidUnit.setZ(sigm_z[i]);
        crt_err = crt_node.errorValue(true);
        if(crt_err > prev_err_per_neuron[i] )
            m_sigmoidUnit.setZ(m_sigmoidUnit.getZ()+(m_sigmoidUnit.getZ()*getModRate()));
        if(crt_err <  prev_err_per_neuron[i] )
            m_sigmoidUnit.setZ(m_sigmoidUnit.getZ()-(m_sigmoidUnit.getZ()*getModRate()));

        prev_err_per_neuron[i] = crt_err;
        //m_sigmoidUnit.setZ(0); // for debugging
     }

 }
```

Fig.7.1. 1 Modificări adăugate la clasa MultilayerPerceptron

Din punct de vedere al utilizatorului, modificarea dinamică presupune setarea inițializării pentru procesul de antrenare, care poate fi vizualizat în figura Fig.7.1.2. cu valorile implicite care sunt β = 10% și varianta direcției de modificare setată pe Adevărat (*True*).

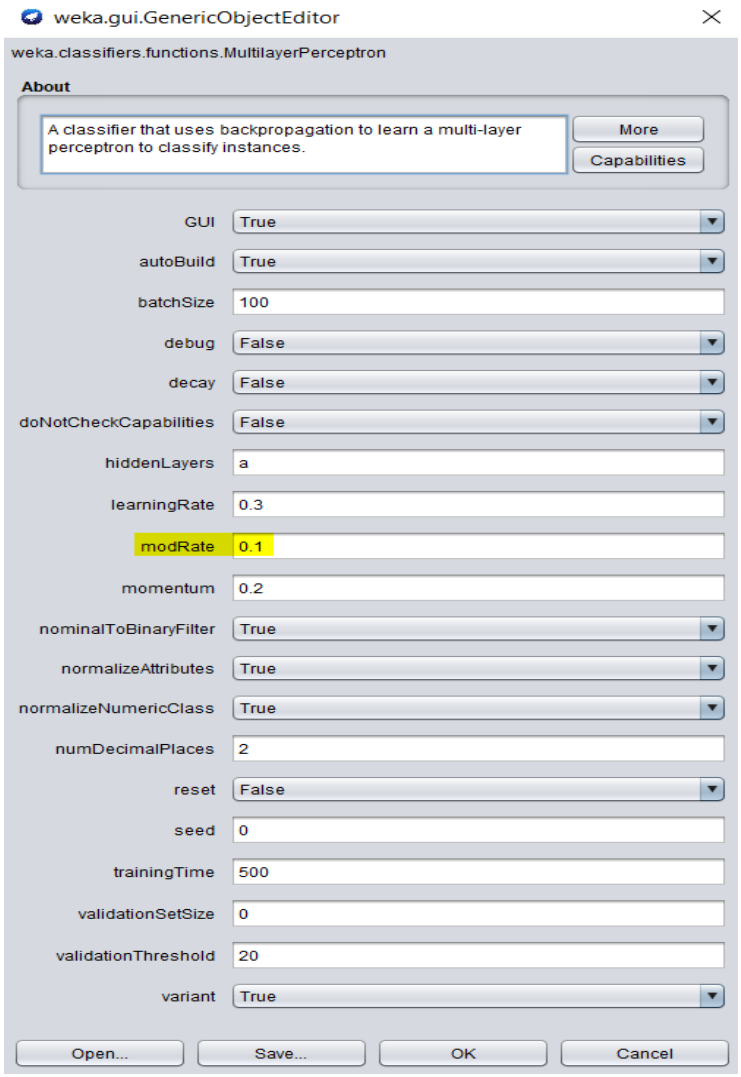


Fig.7.1. 2 Valorile β = 10% (*modRate*) și direcția de modificare (*variant*)

Studiul a fost realizat folosind un număr de 10 seturi de date, format .arff, furnizate de această platformă după cum urmează:

1. *diabetes.arff*: Pima Indians Diabetes Database, variabila diagnostic, evaluată cu valoare binară, este dacă: un pacient prezintă semne de diabet în conformitate cu Organizația Mondială a Sănătății după anumite criterii. [268] Mai multe constrângeri au fost plasate în selectarea acestor cazuri, de exemplu toți pacienții sunt de sex feminin de cel puțin 21 de ani din patrimoniul indian Pima. Are un total de instanțe egal cu 768 și un număr de atribute egal 8 (atribute precum: număr de sarcini, vârsta, indice de masă corporală și așa mai departe) la care se mai adaugă clasa.

2. *iris.arff*: Setul de date Iris introdus de Fisher [269] este un set de date multivariate. Setul de date constă din 50 de probe din fiecare din cele trei specii de Iris (Iris setosa, Iris virginica și Iris versicolor). Din fiecare eșantion au fost măsurate patru caracteristici: lungimea și lățimea sepalelor și petalelor, în centimetrii.

3. *credit_g.arff*: German Credit data [270] are 1000 de instanțe, 20 de atribute (7 numerice, 13 categorice).

4. *zoo.arff*: Zoo database [271] este o bază de date simplă care conține 17 atribute cu valoare booleană. Tipul" atribut care este atributul clasei.

5. *mushroom.arff*: Mushroom Database [272] conține recorduri de ciuperci extrase din Ghidul câmpului Societății Audubon pentru ciuperci din America de Nord (1981).

6. *anneal.arff*: Annealing Data [273] are 798 de instanțe și 38 de atribute (familie, tip produs, oțel, carbon și așa mai departe). Este un set de date care verifică dacă un material are consistența dorită, textura și duritatea în urma unui proces de încălzire și răcire treptată.

7. *balance_scale.arff*: Balance Scale Data Set [274], acest set de date a fost generat pentru modelarea rezultatelor experimentale psihologice. Fiecare exemplu este clasificat ca având echilibrul pe o scară fie spre dreapta, fie spre stânga sau cazul când este echilibrat.

8. *glass.arff*: Glass Identification Data Set [275], acesta este un set de date de identificare tipului de sticlă. Conține 10 atribute, inclusiv ID-ul. Răspunsul este tipul sticlei.

9. *sonar.arff*: Sonar, Mines vs. Rocks Dataset, acesta este un set de date folosit pentru studiul clasificării semnalelor sonare folosind o rețea neuronală. Fiecare model este un set de 60 de numere cuprinse între 0 și 1. Fiecare număr reprezintă energia dintr-o anumită bandă de frecvență, integrată într-o anumită perioadă de timp. [276]

10. *car.arff*: Automobile Dataset [277], acest set de date constă din date din 1985 Ward's Automotive Yearbook. Acest set de date constă din trei tipuri de date:
    (a) specificația unui automobil în termeni în funcție de diferite caracteristici,
    (b) ratingul de risc al asigurării alocat,
    (c) pierderile normalizate ale utilizării în comparație cu alte automobile.
11. *letter.arff*: Letter Recognition dataset [278] este un set de date de clasificare cu mai multe clase. Obiectivul este identificarea fiecărui număr mare de afișaje pixel dreptunghiulare alb-negru ca una dintre

cele 26 de litere majuscule din alfabetul englez, unde literele alfabetului sunt reprezentate în 16 dimensiuni.

Pentru aceste seturi de date putem observa rezultatele experimentale în următoarele tabele și diagrame, care sunt centralizate în tabelul 7.1.1., tabelul 7.1.2. și figurile Fig.7.1.3., Fig.7.1.4., Fig.7.1.5., Fig.7.1.6.

Tabel 7.1. 1 Modificarea dinamică a funcției sigmoide

Set de date	**Variația erorii/epocă β=5%**	**Variația erorii/epocă β=10%**
diabetes	0.0646	2.1728
iris	-0.4377	-0.3989
credit_g	27.8532	13.3432
zoo	136.9565	-30
mushroom	-25	-80
anneal	-2.3638	-85.0199
balance_scale	0.3788	1.0846
glass	0.9818	2.1394
sonar	1.0156	1.7077
car	-1.2320	-15.1951
letter	2.1341	1.9218

Tabel 7.1. 2 Clasificarea instanțelor

Set de date	**Variația dinamică a instanțelor clasificate corect β=5%**	**Variația dinamică a instanțelor clasificate corect β=10%**
diabetes	0.6461	-1.7770
iris	0	0
credit_g	-0.2014	-0.1007
zoo	0	0
mushroom	0	0
anneal	1.2835	4.4268
balance_scale	-0.1736	-0.1736
glass	0.5434	-1.0869
sonar	0	0
car	0	0
letter	-0.8913	-0.7701

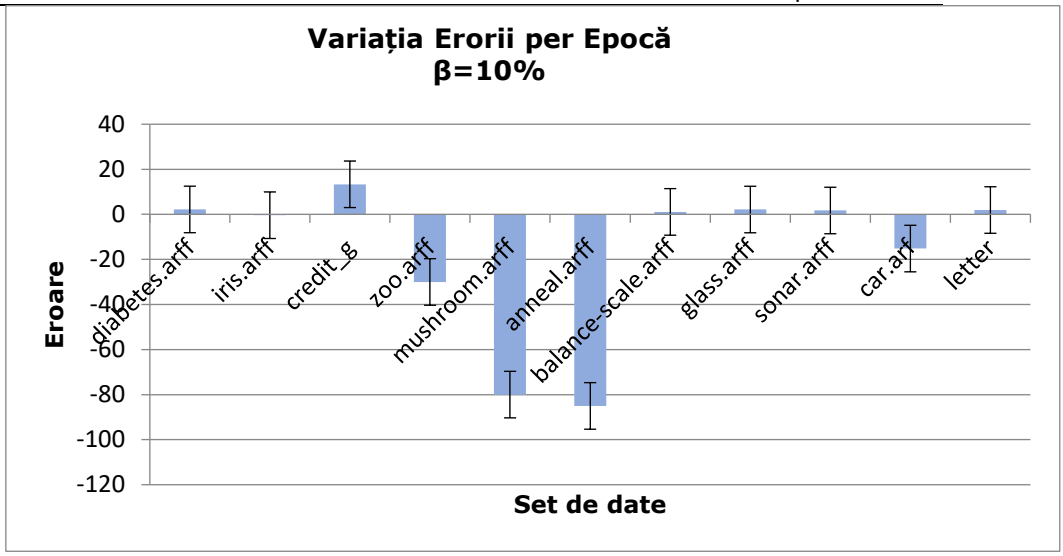


Fig.7.1. 3 Variația erorii pe epocă β=10%

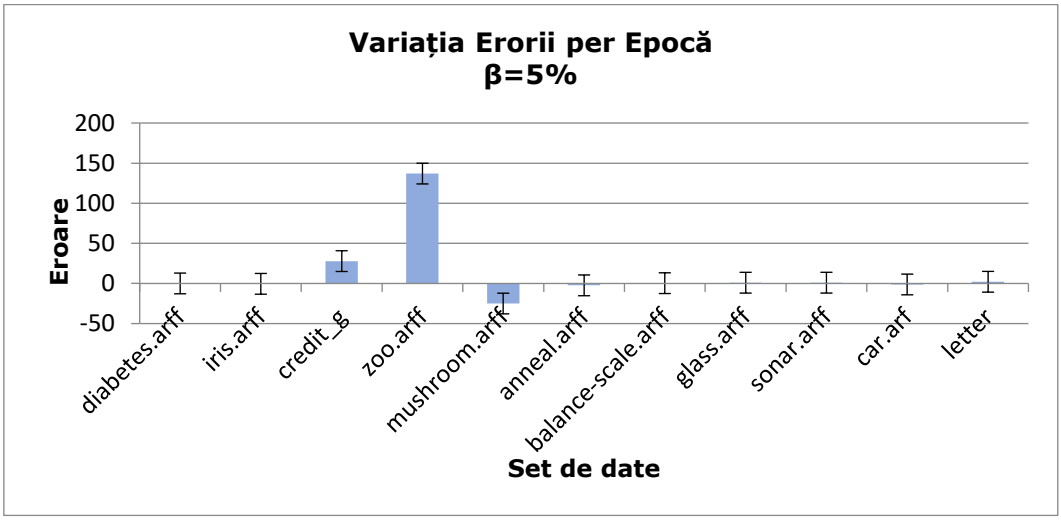


Fig.7.1. 4 Variația erorii pe epocă β=5%

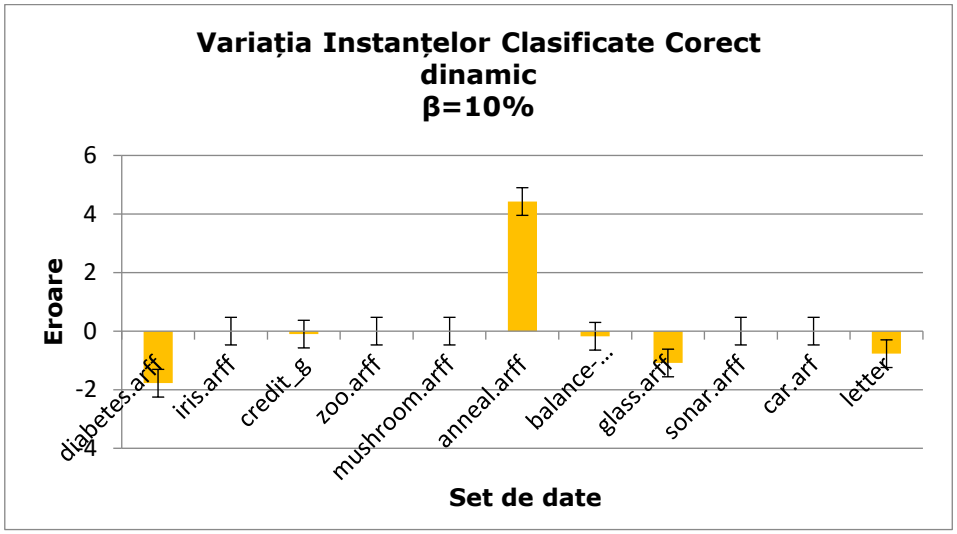


Fig.7.1. 5 Variația instanțelor clasificate correct în mod dinamic β=10%

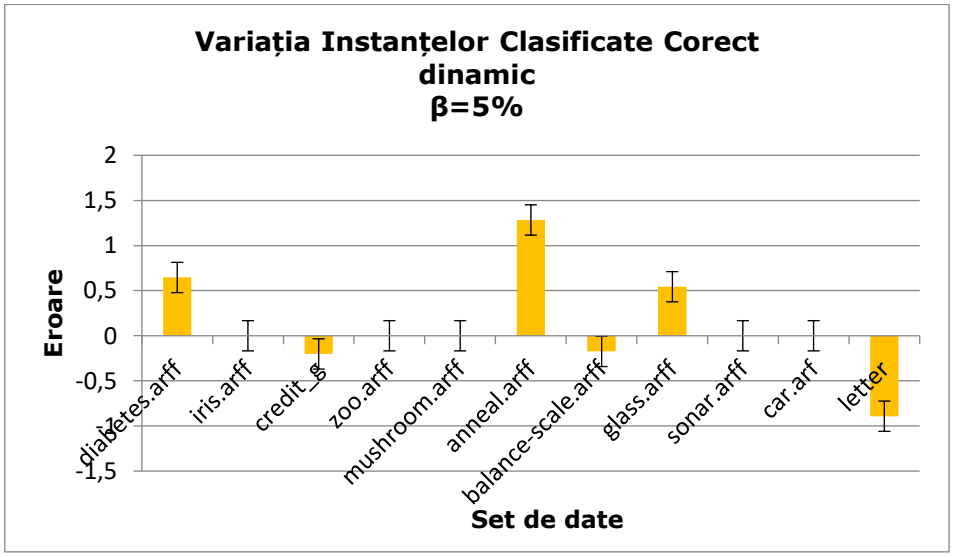


Fig.7.1. 6 Variația instanțelor clasificate correct în mod dinamic β=5%

7.1.1. Evaluarea impactului asupra antrenării prin introducerea funcției de activare modificată dinamic – Partea experimentală

După analiza rezultatelor experimentale se pot face câteva observații:

a) Introducerea conceptului dinamic în funcția de activare conduce la o schimbare a procesului de antrenare a unității (neuronului) care duce rapid la reducerea erorilor de învățare.
De exemplu, acest aspect este vizibil în graficul evoluției erorilor în cazul utilizării setului de date iris.arff (β = 10%, variant = *Adevărat*) așa cum se arată în figura următoare:

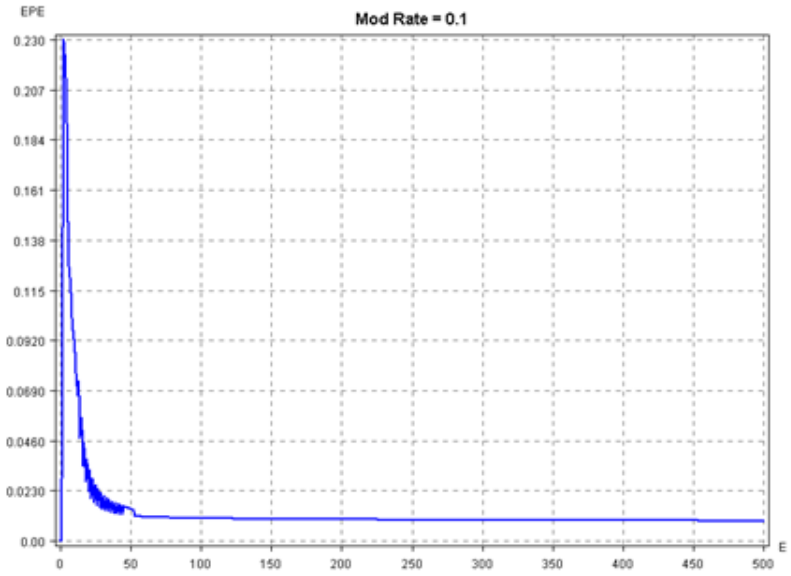


Fig.7.1.1. 1 Variația erorii per epocă pe setul de date Iris (β=10%, variant=*Adevărat*)

Se poate observa că în primele 50 de epoci modificarea dinamică a funcției de activare conduce la o scădere a erorii în timpul antrenării. Acest aspect este prezent și în cazul când β = 5%, variant = Adevărat, caz care poate fi vizualizat în graficul din figura Fig.7.1.1.2.

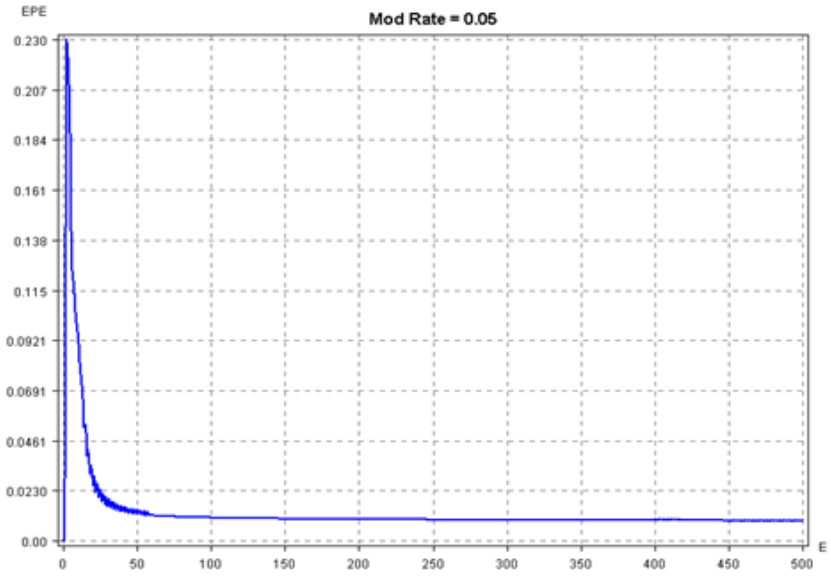


Fig.7.1.1. 2 Variația erorii per epocă pe setul de date Iris (β=5%, variant=*Adevărat)*

b) Pentru patru seturi de date (zoo, mushroom, anneal, carr) se poate observa o reducere certă a erorii obținute la sfârșitul celor 500 de epoci de antrenare. Eroarea de antrenare este redusă după cum urmează: 30% pentru zoo, 80% pentru mushroom, 85% pentru anneal respectiv cu 15% pentru carr în comparație cu situația în care antrenarea se efectuează cu o funcție de activare statică, funcție ale cărei caracteristici rămân constante în timpul antrenării.
Singurul set de date unde crește eroarea de antrenare este credit_g, a cărui eroare crește cu 15%. Pentru celălalte seturi de date variația erorii este nesemnificativă.

c) Un aspect interesant este legat de următorul lucru: pentru setul de date credit_g, a cărui eroare crește, putem vedea că în faza de testare păstrează același număr de instanțe corect clasificat ca în cazul utilizării unei funcții de activare neschimbată. Acest aspect este remarcat pentru setul de date zoo, care pentru parametrul β = 5% avem pe de o parte o creștere a erorii de antrenament egală cu 136,6%, dar pe de altă parte când trecem la faza de testare, nu apare nicio variație a numărului de instanțe clasificate corect. Acest aspect pare a fi rezultatul faptului că instanțele din setul de date pentru o funcție de activare statică nu sunt clasificate corect, deci în cazul acestei funcțiii de activare unele date din set sunt clasificate corect, în timp ce alte instanțe sunt clasificate incorect.

d) Putem observa că există diferențe semnificative între o activare dinamică a funcției cu β = 10% și o activare dinamică a funcției cu β = 5%. Dar ceea ce putem remarca în ambele cazuri este o îmbunătățire a performanței de clasificare. De asemenea, este evident faptul că va fi necesară găsirea

modificarii dinamice optime pentru funcția de activare de la o epocă de antrenament la o altă epocă.

7.2. TeLU: o nouă funcție de activare pentru Învățarea Profundă – Partea experimentală

Compar TeLU cu ReLU, Swish și alte funcții populare, folosind diferite tipuri de arhitecturi pe diferite seturi de date pentru a verifica dacă funcția de activare dă rezultate bune.
Voi folosi arhitectura ResNet18 [33], LeNet-5 [21], MobileNet [158] și AlexNet pentru a vedea rezultatele obținute pe setul de date MNIST [183], Fashion MNIST [279], CIFAR-10, CIFAR-100 [30]. Evaluez funcția TeLU pentru cazul cu parametrul predefinit $\alpha = 1$ dar și pentru cazul când acest parametru devine învățabil. În toate experimentele folosesc mărirea datelor în timp real (*data augmentation*) pentru a evita supraspecializarea.

7.2.1. Experimentul 1: Setul de date MNIST

Setul de date MNIST este format din cifre scrise de mână, acesta are un set de antrenare de 60.000 de exemple și un set de test de 10.000 de exemple. (Y. LeCun et al.). Am testat funcția de activare propusă pe mai multe arhitecturi, cum ar fi LeNet-5 și o arhitectură personalizată atât cazul cu creșterea datelor cât și fără creșterea datelor. Am folosit metoda de oprire timpurie (*early stopping*) cu răbdarea (*patience*) este 20, 100 epoci (epochs) și dimensiunea lotului (*batch size*) egală cu 64. Mărirea datelor (*data augmentation*) a făcut ca modelul nostru să fie mai robust. Precizia validării, așa cum se vede în tabelul următorul, depinde de arhitectură și date. Arhitectura personalizată pe care am creat-o pentru a vedea cum se comportă funcția mea de activare este de forma:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 128)       1280
_________________________________________________________________
batch_normalization (BatchNo (None, 26, 26, 128)       512
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 24, 24, 128)       147584
_________________________________________________________________
batch_normalization_1 (Batch (None, 24, 24, 128)       512
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 6, 6, 128)         0
_________________________________________________________________
dropout (Dropout)            (None, 6, 6, 128)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 256)         295168
_________________________________________________________________
batch_normalization_2 (Batch (None, 4, 4, 256)         1024
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 2, 2, 256)         0
_________________________________________________________________
dropout_1 (Dropout)          (None, 2, 2, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 256)               262400
_________________________________________________________________
batch_normalization_3 (Batch (None, 256)               1024
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 744,202
Trainable params: 742,410
Non-trainable params: 1,792
```

Fig.7.2.1. 1 Arhitectură personalizată pe setul de date MNIST

Tabel 7.2.1. 1 Acuratețea pe setul de validare pentru setul de date MNIST

Funcția de activare	**Tipuri de architecturi**		
	LeNet-5	***Arhitectură personalizată fără mărirea datelor***	***Arhitectură personalizată cu mărirea datelor***
LReLU	**98.79%**	99.58%	97.58%
PReLU	98.72%	99.58%	97.35%
Softplus	98.77%	**99.62%**	97.36%
ELU	98.56%	99.55%	97.61%
SELU	98.76%	99.55%	97.45%
GELU	98.75%	99.57%	97.29%
ReLU	98.47%	99.54%	97.45%
Swish-1	98.77%	99.52%	97.59%
Swish	98.67%	**99.62%**	97.62%
TeLU	98.17%	99.54%	**97.69%**

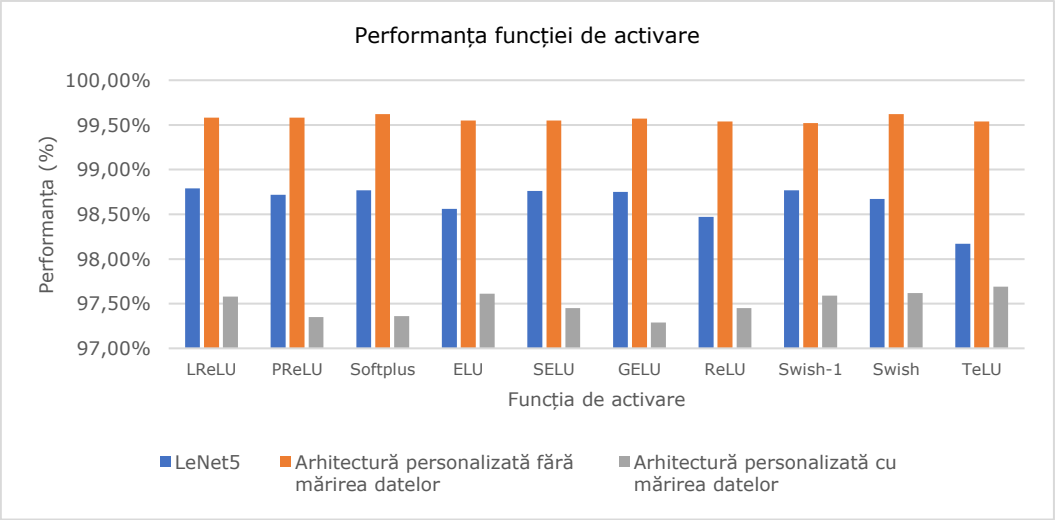


Fig.7.2.1. 2 Variația performanței pe setul de test pe setul de date MNIST

Tabel 7.2.1. 2 Funcția de cost pe setul de validare pentru setul de date MNIST

Funcția de activare	**Tipuri de architecturi**		
	LeNet-5	***Arhitectură personalizată fără mărirea datelor***	***Arhitectură personalizată cu mărirea datelor***
LReLU	0.0482	0.0209	0.0801
PReLU	**0.0458**	0.0215	0.0809
Softplus	0.0483	0.0207	0.0807
ELU	0.0535	0.0219	0.0737
SELU	0.0488	0.0209	0.0776
GELU	0.0506	0.0185	0.0806
ReLU	0.0621	0.0248	0.0770
Swish-1	0.0564	0.2191	0.0722
Swish	0.0587	**0.0200**	**0.0702**
TeLU	0.0824	0.0222	0.0712

Din tabelul 7.2.1.1. cu rezultatele privind acuratețea pe setul de test pentru MNIST se poate vedea că funcția TeLU nu a dat cele mai bune rezultate, dar totuși pentru arhitectura personalizată fără augmentarea datelor a condus la o precizie mai bună ca funcția Swish_1. După aplicarea tehnicii de mărire a datelor se poate vedea că această funcția a dat cele mai bune rezultate comparativ cu alte 9 funcții de activare populare, și anume: LReLU, PreLU, Softplus, ELU, SELU, GELU, ReLU, Swish-1 și Swish. Aceste rezultate ne certifică faptul că această propunere ar fi o soluție foarte bună pentru arhitecturi personalizate cu un număr mic de parametri.

În ceea ce privește funcția de cost, se poate vedea din tabelul 7.2.1.2. că TeLU are o funcție de cost mai mică decât funcțiile Swish_1, ReLU, GELU, SELU, ELU, Softplus, PReLU și LReLU, lucru care îi conferă acestei funcții potențial.

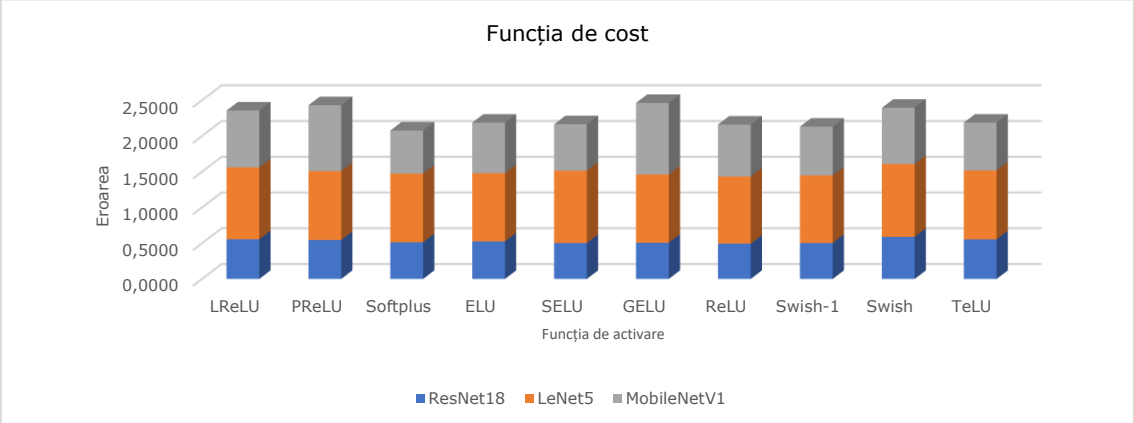


Fig.7.2.1. 3 Variația funcției de cost pe setul de date MNIST

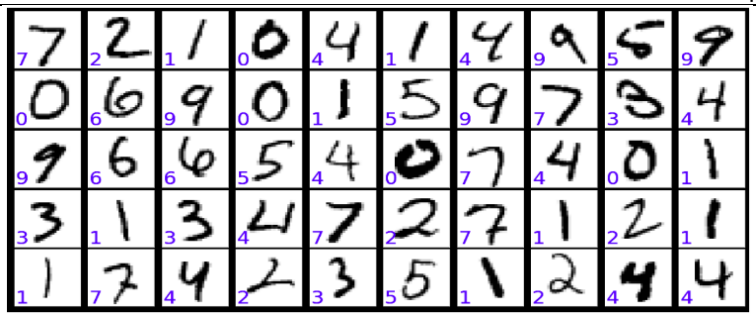


Fig.7.2.1. 4 Predicții pe setul de date MNIST

În Fig.7.2.1.4 s-au făcut predicții pe setul de date MNIST folosind arhitectura personalizată cu mărirea de date utilizând funcția TeLU și se poate observa că s-au identintificat toate clasele corect.

Pe lângă arhitectura personalizată am folosit și arhitectura utilizată la testarea funcției Mish și TanhExp, arhitectura conține 15 straturi, cu straturi dense de 500 de neuroni, straturi de normalizare a lotului, funcția de activare și o rată de dropout de 0,25 fiecare. De asemenea, am implementat Mish, TanExp și ReLU în aceleași condiții (doar am înlocuit activările „relu” cu funcția propusă). Ce am observat în urma experimentelor, că un număr mai mare de epoci conduce la o precizie pe setul de test mai mare, inclusiv la o funcție de cost care scade pentru toate funcțiile de activare. Cu toate acestea, în Fig.7.2.1. 5 precizia TeLU depășește Mish, TanhExp și ReLU după 10 epoci de antrenare, ceea ce dovedește că TeLU poate învăța să poată actualiza parametri rapid și acest lucru duce la o precizie ridicată.

Fig.7.2.1. 5 Acuratețea în epoca 10 pe setul de validare pe MNIST

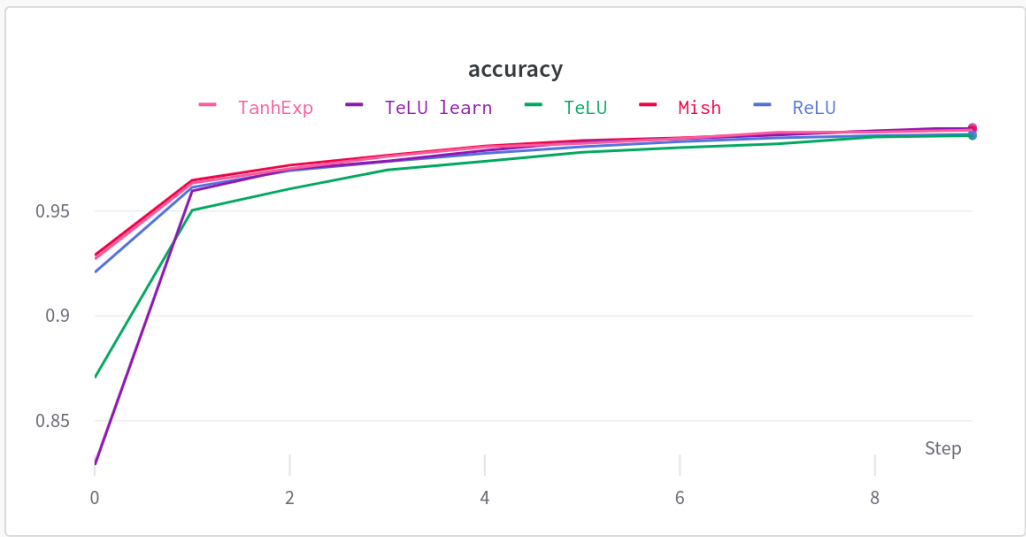


Fig.7.2.1. 6 Acuratețea pe setul de validare pe MNIST

Fig.7.2.1. 7 Funcția de cost în epoca 10 pe setul de validare pe MNIST

Fig.7.2.1. 8 Funcția de cost pe setul de validare pe MNIST

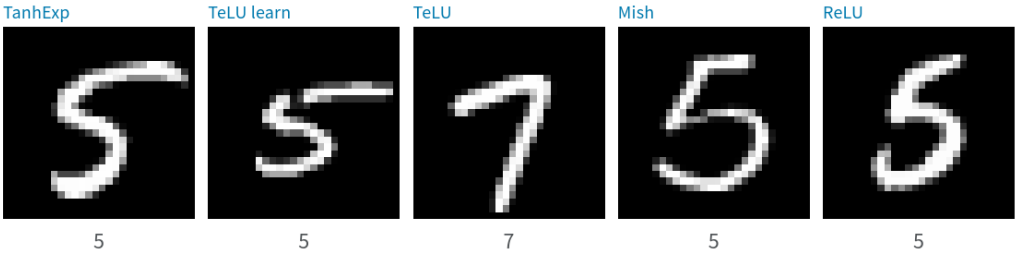


Fig.7.2.1. 9 Predicții pe setul de date MNIST pentru fiecare funcție de activare

Name (5 visualized)	Runtime	accuracy	epoch	loss
TeLU learn	1m 22s	0.9899	9	0.033
Mish	1m 25s	0.989	9	0.0359
TanhExp	1m 20s	0.9885	9	0.03672
ReLU	1m 24s	0.9865	9	0.04258
TeLU	1m 22s	0.9859	9	0.05112

Fig.7.2.1. 10 Sumarizarea rezultatelor de antrenare pe setul de validare pe MNIST

7.2.2. Experimentul 2: Setul de date Fashion-MNIST

Ca și în cazul funcției de activare Swish, testez funcția TeLU pe mai multe arhitecturi pe setul de date Fashion-MNIST care conține 70.000 de imagini gri, împărțite în 10 categorii. Imaginile prezintă articole vestimentare individuale la rezoluție mică (28 x 28 pixeli). În cazul arhitecturii LeNet-5, am folosit metoda de oprire timpurie cu răbdare de 5, 100 de epoci și dimensiunea lotului de 64.

De asemenea, am personalizat o arhitectură pentru a vedea cum se comportă funcția mea de activare, ca în figura următoare:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 128)       1280
_________________________________________________________________
batch_normalization (BatchNo (None, 26, 26, 128)       512
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 24, 24, 128)       147584
_________________________________________________________________
batch_normalization_1 (Batch (None, 24, 24, 128)       512
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 6, 6, 128)         0
_________________________________________________________________
dropout (Dropout)            (None, 6, 6, 128)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 4, 4, 256)         295168
_________________________________________________________________
batch_normalization_2 (Batch (None, 4, 4, 256)         1024
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 2, 2, 256)         0
_________________________________________________________________
dropout_1 (Dropout)          (None, 2, 2, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0
_________________________________________________________________
dense (Dense)                (None, 256)               262400
_________________________________________________________________
batch_normalization_3 (Batch (None, 256)               1024
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 744,202
Trainable params: 742,410
Non-trainable params: 1,792
```

Fig.7.2.2. 1 Arhitectură personalizată pe setul de date Fashion-MNIST

Tabel 7.2.2. 1 Acuratețea pe setul de validare pentru setul de date Fashion-MNIST

Funcția de activare	**Tipuri de architecturi**		
	LeNet-5	***Arhitectură personalizată fără mărirea datelor***	***Arhitectură personalizată cu mărirea datelor***
LReLU	88.81%	93.58%	92.22%
PReLU	89.21%	93.60%	92.76%
Softplus	88.92%	93.68%	92.41%
ELU	89.33%	93.75%	92.13%
SELU	89.34%	93.63%	92.70%
GELU	89.66%	**93.99%**	92.95%
ReLU	89.31%	93.90%	92.56%
Swish-1	**90.09%**	94.02%	92.85%
Swish	89.92%	93.89%	**93.16%**
TeLU	**90.09%**	93.93%	**93.16%**

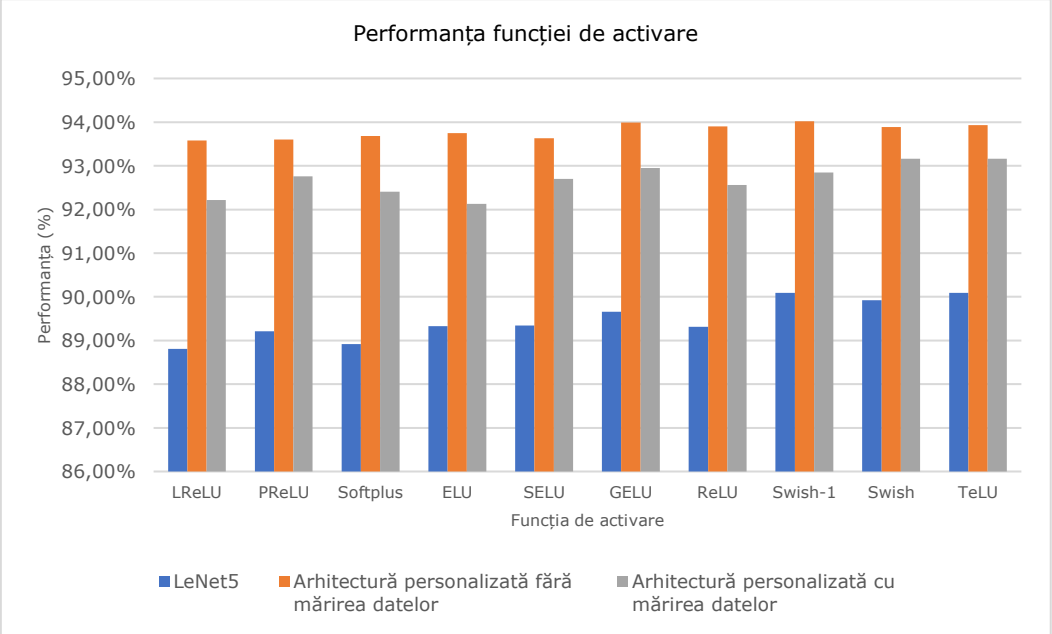


Fig.7.2.2. 2 Variația performanței pe setul de validare pe Fashion-MNIST

După cum se vede în tabelul 7.2.2.2., rezultatele experimentelor cu două arhitecturi diferite: LeNet-5 (dezvoltată de Google) și arhitectură personalizată cu cele două cazuri: când se folosește mărirea datelor și când nu se folosește această tehnică. Se poate observa că arhitectura propusă funcționează bine pe acest set de date și un alt avantaj este dat de numărul mic de parametri de $\approx 744\text{k}$. O altă tehnică de evitare a supraspecializării pe care am folosit-o a fost scăderea succesivă a valorii de *dropout* (de la 0,5, 0,4, 0,3, 0,2). Funcția de activare TeLU pe arhitectura LeNet-5 dă alături de funcția Swish_1 cele mai bune rezultate. Pe arhitectura personalizată fără mărirea datelor, TeLU se clasează pe locul 2, după funcția GELU, iar pe arhitectura personalizată cu mărirea datelor, TeLU se clasează pe primul loc alături de funcția Swish cu parametru învățabil.

Tabel 7.2.2. 2 Funcția de cost pe setul de validare pentru setul de date Fashion-MNIST

Funcția de activare	**Tipuri de architecturi**		
	LeNet-5	***Arhitectură personalizată fără mărirea datelor***	***Arhitectură personalizată cu mărirea datelor***
LReLU	0.3120	0.2173	0.2180
PReLU	0.2987	0.2231	0.2072
Softplus	0.3076	0.2271	0.2193
ELU	0.2832	0.2401	0.2192
SELU	0.3113	0.2365	0.2062
GELU	0.2944	0.2491	0.2028
ReLU	0.3069	0.2577	0.2160
Swish-1	0.2872	0.2566	0.2060
Swish	0.2899	0.2513	0.2009
TeLU	**0.2987**	**0.2642**	**0.1972**

Putem observa în tabelul 7.2.2.2. că funcția TeLU are cea mai mică funcție de cost pe setul de test.

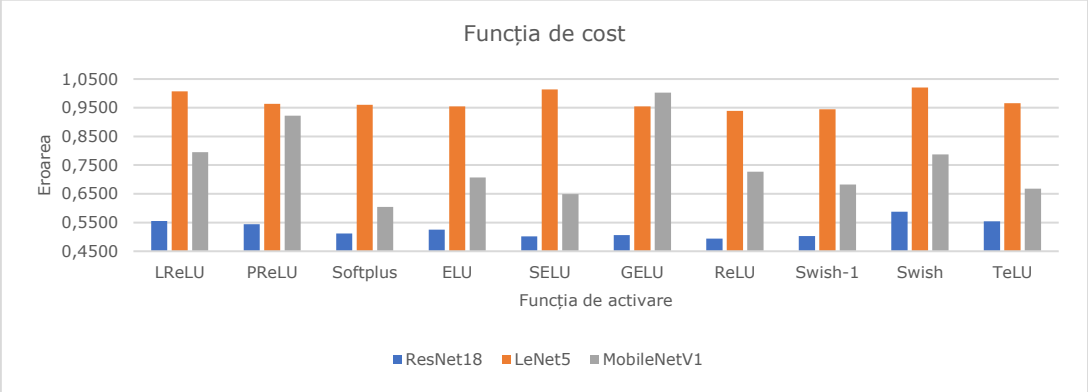


Fig.7.2.2. 3 Variația funcției de cost pe setul de date Fashion-MNIST

Fig.7.2.2. 4 Predicții pe setul de date Fashion-MNIST

În Fig.7.2.2.4. s-au făcut predicții pe setul de date Fashion-MNIST folosind arhitectura LeNet-5, etichetele ilustrate cu albastru sunt etichetele reale, etichetele ilustrate cu roșu sunt etichetele prezise eronat.

Pentru a testa funcția TeLU învățabilă am folosit aceleași setări ca în setul de date MNIST, rețeaua de bază cu 15 straturi, dar de această dată, Fashion-MNIST fiind un set de date mai complex am antrenat pentru 20 de epoci ca în cazul implementării funcției de activare TanhExp.

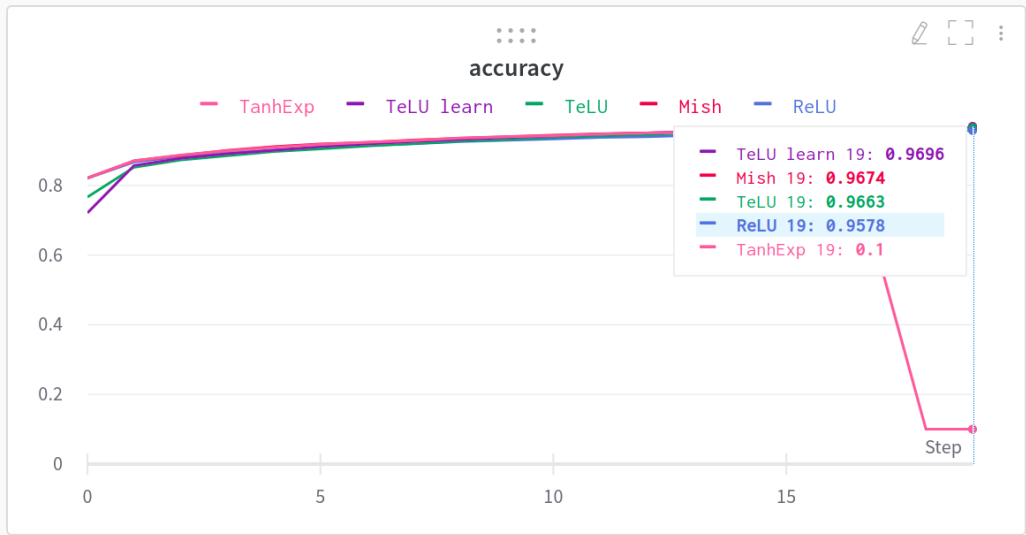


Fig.7.2.2. 5 Acuratețea pe setul de validare în epoca 20 pe setul de date Fashion-MNIST

Din figura de mai sus se poate vedea că funcția TeLU învățabilă a dat cele mai bune rezultate, depășind funcțiile Mish, ReLU și TanhExp.

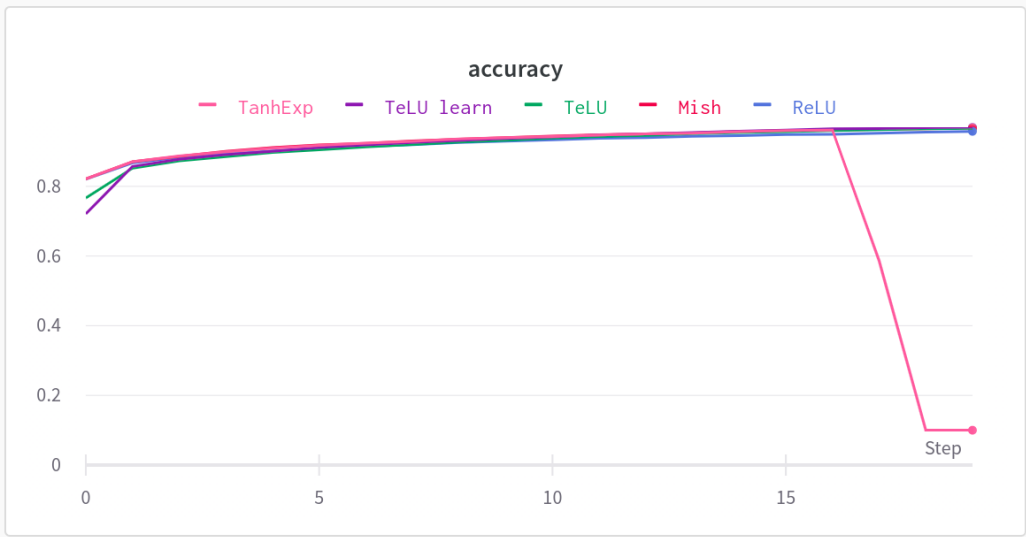


Fig.7.2.2. 6 Acuratețea pe setul de validare pe Fashion-MNIST

Fig.7.2.2. 7 Funcția de cost în epoca 20 pe setul de validare pe Fashion-MNIST

Fig.7.2.2. 8 Funcția de cost în epoca 20 pe setul de validare pe Fashion-MNIST

Name (5 visualized)	Runtime	accuracy	epoch	loss
TeLU learn	5m 27s	0.9696	19	0.08247
Mish	5m 25s	0.9674	19	0.08982
TeLU	5m 23s	0.9663	19	0.09295
ReLU	5m 10s	0.9578	19	0.117
TanhExp	5m 23s	0.1	19	NaN

Fig.7.2.2. 9 Sumarizarea rezultatelor de antrenare pe setul de validare pe Fashion-MNIST

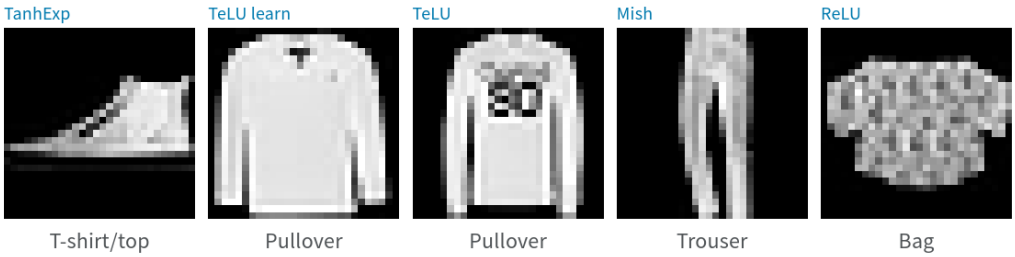


Fig.7.2.2. 10 Predicții pe setul de date Fashion-MNIST pentru fiecare funcție de activare

7.2.3. Experimentul 3: Setul de date CIFAR-10

Setul de date CIFAR-10 este format din 60000 de imagini color 32x32 grupate în 10 clase, cu 6000 de imagini per clasă. Există 50000 de imagini de antrenament și 10000 de imagini de test.

Legat de configurația arhitecturii, am testat ambele funcții folosind mai multe arhitecturi: LeNet-5, AlexNet, MobileNetV1, MobileNetV2 și ResNet18. Am folosit arhitecturi mici și mari pentru a vedea cum se comportă activarea. Am ales MobileNetV1 ca arhitectură pentru numărul ei redus de parametri, și astfel conducând la un timp redus de antrenare. Mi-am orientat atenția către arhitecturi mici și robuste pentru a fi rapide și precise. Abordarea mea a fost să găsesc o funcție de activare adecvată care să depășească funcțiile actuale de activare pentru a îmbunătăți precizia pe setul de date dar cu economie de timp și resurse computaționale. În cazul arhitecturii AlexNet, am folosit straturi de normalizare a lotului și dropout cu o rată egală cu 0,5. În calitate de optimizator, am folosit „rmsprop” și fiind o clasificare de tip multi-clase am folosit-o ca funcție de cost „categorical_crossentropy”. De asemenea, am folosit tehnica de mărire a datelor pentru AlexNet, am folosit întoarcere întâmplătoare a imaginilor pe orizontală (0.1) și pe verticală (0.1), am setat modul pentru umplerea punctelor în afara limitelor de intrare („cel mai apropiat”), am setat și o divizare de validare egală cu 0,2, ceea ce înseamnă că am folosit 20% din date pentru validare.

Am folosit mai multe tipuri de arhitecturi pe setul de date CIFAR-10 și anume: ResNet18, Lenet-5 și MobileNet versiunea 1. În tabelul de mai jos putem vedea cum precizia este influențată de tipul arhitectură și puterea de calcul. Am folosit oprirea timpurie și am antrenat doar 100 de epoci.

Tabel 7.2.3. 1 Acuratețea de validare pe setul de date CIFAR-10 pentru funcția TeLU

Funcția de activare	***ResNet18***	***MobileNetV1***
LReLU	89.53%	81.05%
PReLU	90.13%	81.63%
GELU	90.60%	80.91%
ReLU	90.16%	84.46%
Swish-1	**90.78%**	84.01%
Swish învățabilă	88.53%	81.76%
TeLU	90.07%	**84.69%**

Număr de epoci = 100

Din tabelul 7.2.3.1. se poate vedea că funcția TeLU nu dă cele mai bune rezultate, dar cu toate acestea dă rezultate mai bune ca funcțiile de activare astfel:

a. În cazul arhitecturii ResNet18, TeLU dă rezultate mai bune ca funcțiile Swish cu parametru învățabil și LReLU.
b. În cazul arhitecturii MobileNetV1, TeLU dă cele mai bune rezultate depășind funcțiile LReLU, PReLU, GELU, ReLU, Swish-1 și Swish învățabil, rezultate care confirmă potențialul acestei propuneri.

Tabel 7.2.3. 2 Acuratețea de validare pe setul de date CIFAR-10 pentru funcția TeLU învățabilă

Model	**TeLU**	**TeLU învățabilă**	**TanhExp**	**ReLU**	**Mish**
LeNet-5	**73.49%**	70.93%	70.81%	70.41%	70.24%
AlexNet	**85.18%**	72.29%	85.17%	83.19%	84.76%

LeNet-5 antrenat 50 epoci
AlexNet antrenat 20 epoci

Am efectuat experimentele pe un număr variat de epoci, dar TeLU a obținut cele mai bune performanțe foarte rapid și chiar la începutul fazei de antrenare pe LeNet-5 și AlexNet, proprietăți care o fac o soluție simplă și robustă.
Din tabelul 7.2.3.2. se poate vedea că funcția TeLU învățabilă nu dă cele mai bune rezultate în comparație cu alte funcții de activare dar cu toate acestea dă rezultate competitive astfel:

a. În cazul arhitecturii LeNet-5, TeLU fără parametru învățabil dă cele mai bune rezultate, mai bune ca funcțiile TeLU cu parametru învățabil, TanhExp, ReLU și Mish, aducând o îmbunătățire a performanței cu mai bine de 3%.
b. În cazul arhitecturii AlexNet, TeLU dă cele mai bune rezultate depășind TeLU cu parametru învățabil, TanhExp, ReLU și Mish.

Diversitatea funcțiilor utilizate pentru o comparație echitabilă îmi întărește convingerile că aceste funcții pot fi luate în considerare pentru a crește performanța rețelei neuronale profunde.

7.2.4. Experimentul 4: Setul de date CIFAR-100

Setul de date CIFAR-100 este similar cu setul de date CIFAR-10, cu excepția faptului că diferă numărul de clase, pentru acest set de date fiind 100 de clase care conțin 600 de imagini fiecare clasă. Cele 100 de clase din CIFAR-100 sunt grupate în 20 de superclase. Aceste seturi de date au fost colectate de Alex Krizhevsky, Vinod Nair și Geoffrey Hinton.

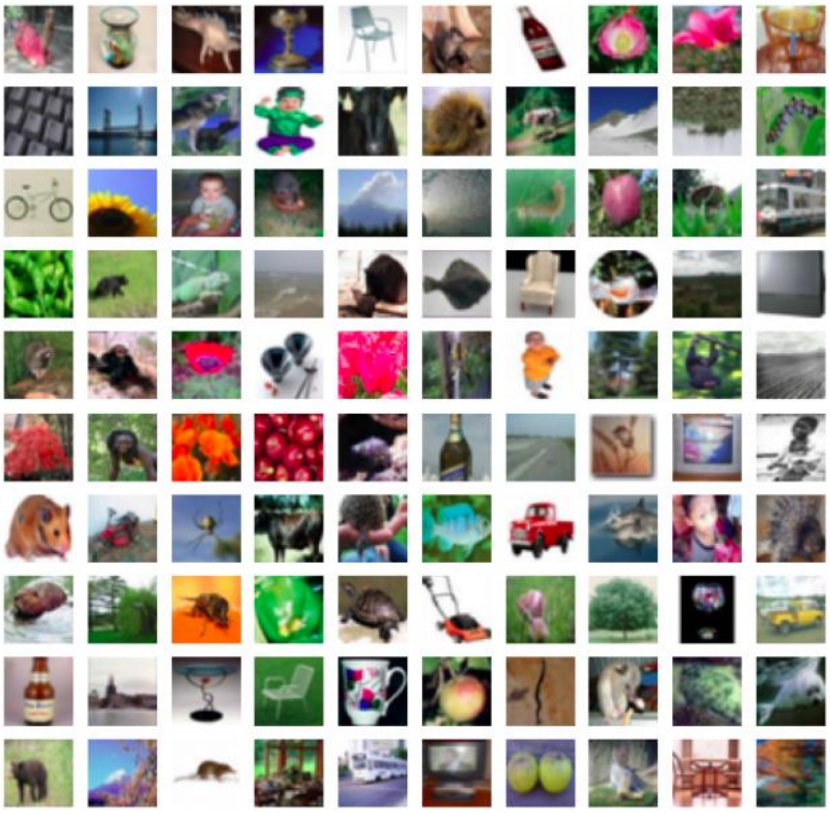


Fig.7.2.4. 1 Setul de date CIFAR-100

Am folosit mai multe tipuri de arhitecturi pe setul de date CIFAR-100 și anume: MobileNetV2 și ResNet18. În tabelul 7.2.4.1. putem vedea cum precizia este influențată de tipul arhitectură și puterea de calcul. Am folosit oprirea timpurie și am antrenat doar 200 de epoci. Pentru un set de date atât de mare precum Cifar-100, se obțin rezultate mai bune antrenând mai multe epoci.

Tabel 7.2.4. 1 Acuratețea de validare pe setul de date CIFAR-100 pentru TeLU

Funcție de activare	***MobileNetV2***	***ResNet18***
LReLU	32.73%	43.86%
PReLU	18.39%	44.57%
Softplus	37.81%	45.00%
ELU	24.80%	45.82%
SELU	27.54%	43.44%
GELU	25.71%	41.80%
ReLU	11.57%	48.96%
Swish-1	41.09%	**48.97%**
TeLU	**41.86%**	47.70%

Din Tabelul 7.2.4.1. se poate vedea că funcția TeLU dă cele mai bune rezultate astfel:

a. În cazul arhitecturii MobileNetV2, TeLU dă cele mai bune rezultate decât funcțiile LReLU, PReLU, Softplus, ELU, SELU, GELU, ReLU și Swish-1.
b. În cazul arhitecturii ResNet18, TeLU nu dă cele mai bune rezultate dar cu toate acestea depășește funcțiile LReLU, PReLU, Softplus, ELU, SELU și GELU.

Tabel 7.2.4. 2 Acuratețea de validare pe setul de date CIFAR-100 pentru TeLU învățabilă

Model	**TeLU**	**TeLU învățabil**	**TanhExp**	**ReLU**	**Mish**
LeNet-5	38.92%	**40.56%**	40.44%	40.31%	39.26%
AlexNet	**62.70%**	48.80%	61.50%	61.31%	61.52%

LeNet-5 și AlexNet antrenate pentru 50 epoci

Din tabelul 7.2.4. 2 se poate vedea că funcțiile TeLU și TeLU învățabilă dau cele mai bune rezultate astfel:

a. În cazul arhitecturii LeNet-5, TeLU învățabilă dă rezultate mai bune ca funcțiile TeLU, TanhExp, ReLU și Mish.
b. În cazul arhitecturii AlexNet, TeLU dă rezultate mai bune ca funcțiile TeLU învățabilă, TanhExp, ReLU și Mish.

În seturile anterioare de experimente, am putut vedea diferite intervale pentru acuratețe, aceste diferențe provin de la faptul că am antrenat câteva epoci și cu arhitecturi pe care mi le-a permis resursele computaționale reduse. În experimente am văzut că cele mai bune rezultate au fost obținute pentru MobileNetV1, MobileNetV2, LeNet-5 și AlexNet pe CIFAR-10 și LeNet-5 și MobileNetV2 pe CIFAR-100. De asemenea, cea mai bună funcție de cost de validare a fost pe Fashion-MNIST pentru arhitectura personalizată cu mărirea datelor. În definirea modelelor am folosit ca optimizator Adam [280] (learning rate LR - rată de învățare egală cu 0,0001), de asemenea și *categorical_crossentropy* pentru funcția de cost.

7.3. Îmbunătățirea preciziei rețelelor neuronale profunde prin dezvoltarea de noi funcții de activare – Partea experimentală

După cum s-a observat în subcapitolul 6.4., am propus 4 funcții de activare, atât cu parametri predefiniți cât și învățabili. Aceste funcții pot fi vizualizate în figura de mai jos:

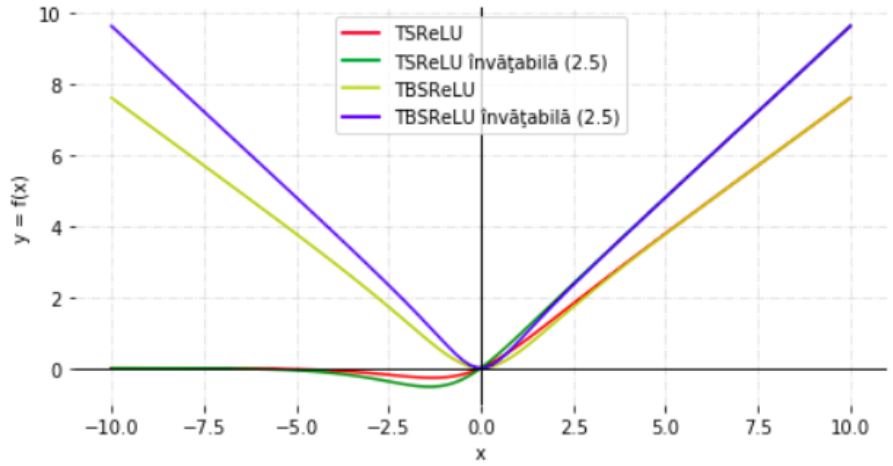


Fig.7.3. 1 Funcții de activare propuse

În Fig.7.3.2. se poate vedea totalitatea funcțiilor folosite la antrenarea rețelelor în cadrul acestui subcapitol, funcții precum sigmoida, tanh, ReLU, Swish, TSReLU, TSReLU învățabilă, TBSReLU și TBSReLU învățabilă.

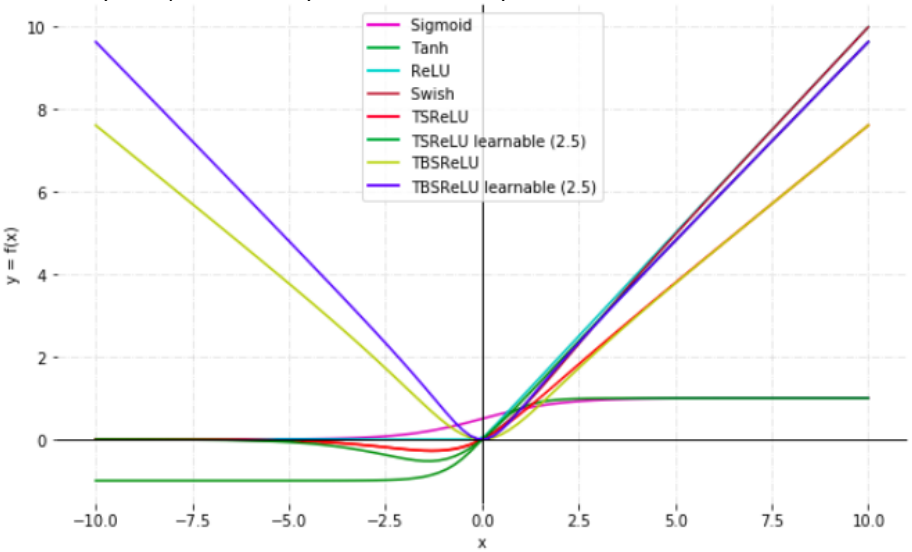


Fig.7.3. 2 Funcții de activare pentru antrenare

7.3.1. Experimentul 1: Setul de date CIFAR-10

Pentru primul experiment, am folosit setul de date CIFAR-10, unul dintre cele mai populare seturi de date din domeniul Computer Vision. Pentru a evalua TSReLU și TSReLU învățabilă, am folosit ResNet18 versiunea 1 cu normalizarea lotului (*batch normalization*) [281] și am antrenat 20 de epoci. Ca și procedură, pur și simplu am înlocuit toate straturile cu activarea „relu" cu propunerile mele. Mai multe informații suplimentare sunt dimensiunea lotului care a fost egală cu 32, rata de învățare egală cu 0,001. Pentru acest set de date, avem 10 clase, deci în arhitectură pe ultimul strat dens voi avea o funcție de activare de tip Softmax, inițializatorul kernelului este „he_normal" și am folosit și mărirea datelor (data augmentation cu atributul flip orizontal) și am amestecat aleatoriu datele. Tabelele de mai jos prezintă precizia validării și funcția de cost de validare pentru funcțiile propuse, în contrast cu alte patru funcții de activare: Sigmoida, Tanh, ReLU, Swish. Având șase funcții de activare pentru evaluare am decis să antrenăm doar 20 de epoci, dar acest număr nu înseamnă nicio constrângere, ci dimpotrivă cu creșterea numărului de epoci va crește și performanța modelului. Pentru a arăta efectul creșterii numărului de epoci adăugăm Tabelul 7.3.1.2. În acest caz, am antrenat modelul pentru 50 de epoci și am folosit o dimensiune a lotului egală cu 64 și o rată de învățare egală cu 0,0001.

Tabel 7.3.1. 1 Rezultate experimentale pentru ResNet18 pe setul de date CIFAR-10

Funcția de activare	**Acuratețea pe setul de validare**	**Funcția de cost pe setul de validare**
Sigmoida	35.95%	2.7083
Tanh	67.10%	1.311
ReLU	79.51%	0.8105
Swish	81.64%	0.7610
TSReLU	**83.72%**	**0.6745**
TSReLU învățabilă	79.21%	0.8205

epoci = 20

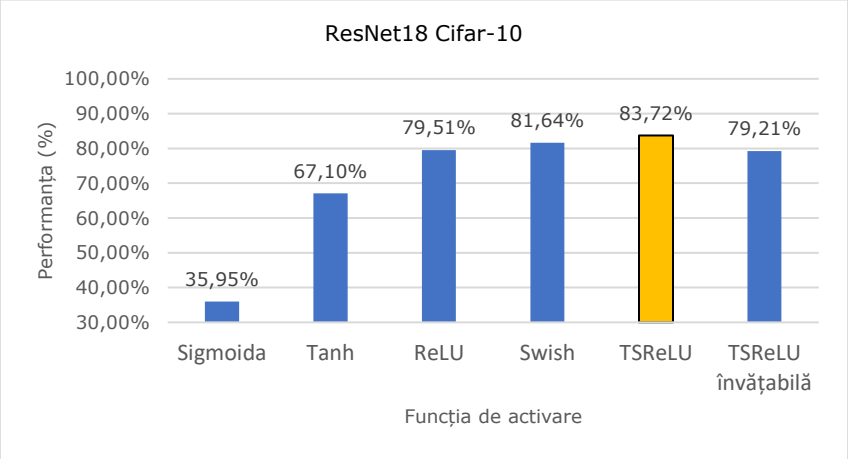


Fig.7.3.1. 1 Rezultate experimentale pe arhitectura ResNet18 pentru setul de date CIFAR-10

Tabel 7.3.1. 2 Rezultate experimentale pentru ResNet18 pe setul de date CIFAR-10

Funcția de activare	**Acuratețea pe setul de validare**	**Funcția de cost pe setul de validare**
Sigmoida	79.05%	0.8700
Tanh	81.34%	0.8277
ReLU	85.29%	0.7201
Swish	86.00%	0.7003
TSReLU	86.20%	0.6894
TSReLU învățabilă	**86.51%**	**0.6738**

epoci = 50

Se poate observa din Fig.7.3.1.1. că pe arhitectura ResNet18 funcția de activare TSReLU are cea mai bună performanță de 83.72% și cea mai mică funcție de cost 0.6745.

Pentru a evalua TSReLU și TSReLU învățabilă, am folosit *ResNet34* versiunea 1 cu normalizarea lotului și am antrenat 20 de epoci. Ca și procedură, pur și simplu am înlocuit toate straturile cu activarea „relu” cu propunerile mele. Mai multe informații suplimentare privind dimensiunea lotului care a fost egală cu 128 și rata de învățare egală egală cu 0,001.

Tabel 7.3.1. 3 Rezultate experimentale pentru Resnet34 pe setul de date CIFAR-10

Funcția de activare	**Acuratețea pe setul de validare**	**Funcția de cost pe setul de validare**
Sigmoida	12.66%	5.7740
Tanh	66.50%	1.1983
ReLU	78.49%	0.8440
Swish	83.81%	0.6747
TSReLU	81.80%	0.7130
TSReLU învățabilă	**83.83%**	**0.6525**

epoci = 20

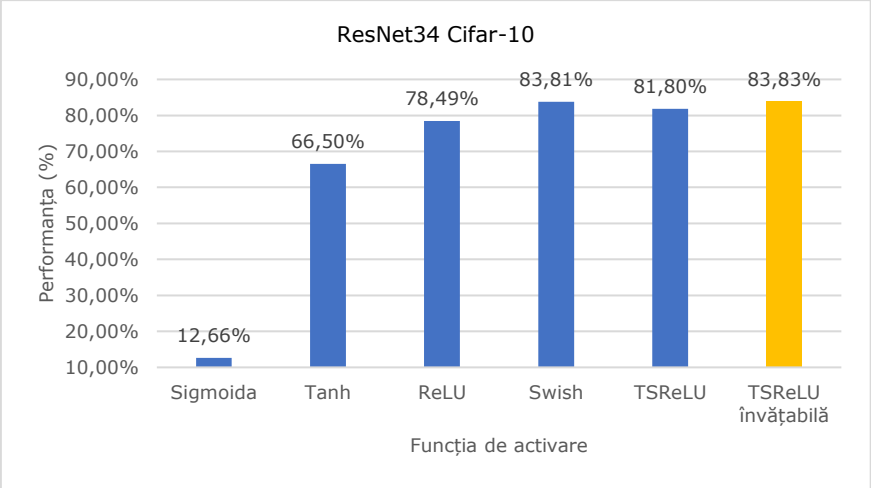


Fig.7.3.1. 2 Rezultate experimentale pe arhitectura ResNet34 pentru setul de date CIFAR-10

Se poate observa din Fig.7.3.1.2. că pe arhitectura ResNet34 funcția de activare TSReLU învățabilă are cea mai bună performanță de 83.83% și cea mai mică funcție de cost 0.6525.

Pentru a-mi adăuga mai multă consecvență evaluării asupra funcțiilor de activare care pot fi TSReLU și TSReLU învățabilă, am folosit ResNet34 versiunea 1 cu normalizarea lotului și am instruit modelul pentru mai multe epoci, un număr egal cu 50. Doar am înlocuit toate straturile de activare „relu” cu propunerea mea. Informații suplimentare sunt dimensiunea lotului egală cu 128 și rata de învățare egală cu 0,001. Aceste rezultate pot fi văzute în tabelul 7.3.1.4.

Tabel 7.3.1. 4 Rezultate experimentale pentru ResNet34 pe setul de date CIFAR-10

Funcția de activare	**Acuratețea pe setul de validare**	**Funcția de cost pe setul de validare**
Sigmoida	72.62%	0.9826
Tanh	82.07%	0.7842
ReLU	86.62%	0.6496
Swish	87.95%	0.6206
TSReLU	88.03%	0.6261
TSReLU învățabilă	**88.10%**	**0.6183**

epoci = 50

7.3.2. Experimentul 2: Setul de date CIFAR-100

În cazul acestui set de date am folosit atât arhitectura ResNet18 cât și ResNet34 versiunea 1 pentru a valida funcționalitatea funcțiilor de activare propuse. Singurele diferențe față de cazul setului de date CIFAR-10, au fost legate de faptul că am antrenat 50 de epoci și dimensiunea lotului a fost egală cu 128. Tabelele următoare prezintă precizia de validare și funcția de cost a funcțiilor de activare în cazul arhitecturilor ResNet18 și ResNet34.

Tabel 7.3.2. 1 Rezultate experimentale pentru ResNet18 pe setul de date CIFAR-100

Funcția de activare	**Acuratețea pe setul de validare**	**Funcția de cost pe setul de validare**
Sigmoida	17.23%	4.4247
Tanh	39.65%	2.6248
ReLU	52.58%	2.0501
Swish	56.93%	1.8810
TSReLU	57.13%	1.9090
TSReLU învățabilă	**57.20%**	**1.8500**

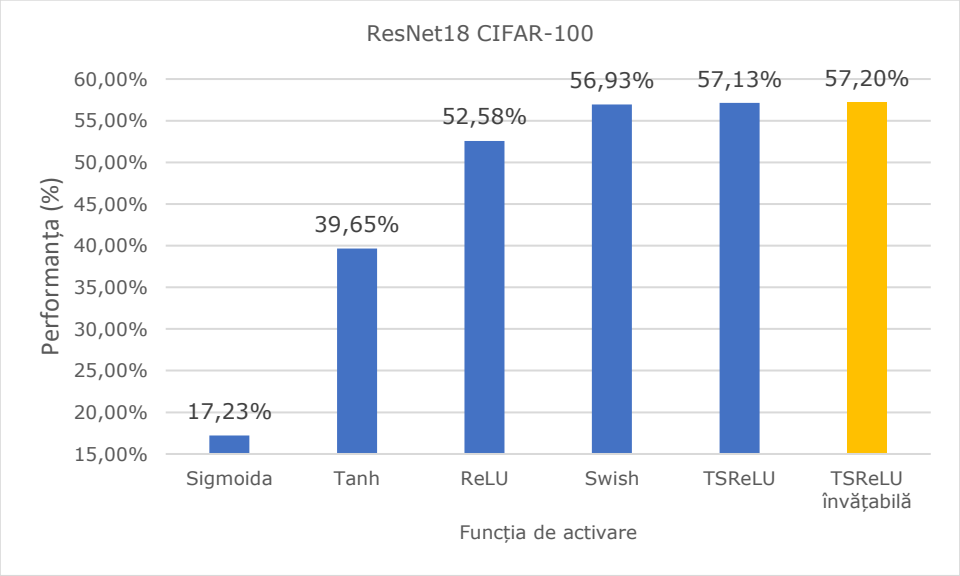


Fig.7.3.2. 1 Rezultate experimentale pe arhitectura ResNet18 pentru setul de date CIFAR-100

Se poate observa din Fig.7.3.2.1. că pe arhitectura ResNet18 funcția de activare TSReLU învățabilă are cea mai bună performanță de 57.20% și cea mai mică funcție de cost 1.8500.

Pentru a-mi adăuga mai multă robustețe evaluării, am antrenat 70 de epoci în cazul arhitecturii ResNet18 pentru setul de date CIFAR-100, aceste rezultate pot fi văzute în tabelul 7.3.2.2.

Tabel 7.3.2. 2 Rezultate experimentale pentru Resnet18 pe setul de date CIFAR-100

Funcția de activare	**Acuratețea pe setul de validare**	**Funcția de cost pe setul de validare**
Sigmoida	39.46%	2.7586
Tanh	47.39%	2.4111
ReLU	57.96%	2.0874
Swish	**59.80%**	**2.0063**
TSReLU	59.56%	2.0120
TSReLU învățabilă	58.61%	2.0348

Tabel 7.3.2. 3 Rezultate experimentale pentru Resnet34 pe setul de date CIFAR-100

Funcția de activare	Acuratețea pe setul de validare	Funcția de cost pe setul de validare
Sigmoida	21.27%	3.4736
Tanh	39.59%	2.8341
ReLU	56.08%	2.0294
Swish	58.72%	1.9275
TSReLU	56.77%	2.0572
TSReLU învățabilă	**59.17%**	**1.8555**

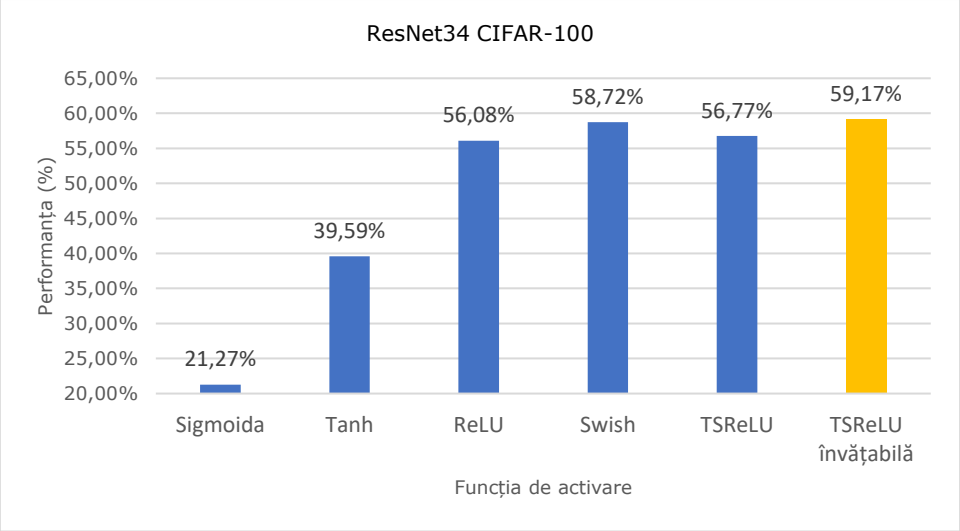


Fig.7.3.2. 2 Rezultate experimentale pe arhitectura ResNet34 pentru setul de date CIFAR-100

Se poate observa din Fig.7.3.2.2. că pe arhitectura ResNet34, funcția de activare TSReLU învățabilă are cea mai bună performanță de 59.17% și cea mai mică funcție de cost 1.8555.

7.3.3. Experimentul 3: Setul de date Câini și Pisici

În cazul acestui set de date am folosit un model pre-antrenat VGG16 cu mărirea datelor (*data augmentation*), mai exact am aplicat modul mărire, rotire, schimbarea lățimii, schimbarea înălțimii și întoarcere orizontală. Funcția de activare propusă, de această dată, am aplicat-o doar pe ultimele două straturi dense, pur și simplu înlocuind funcția de activare „relu” cu funcția de activare propusă. De asemenea, am aplicat *dropout* după fiecare strat dens. Fiind o sarcină de clasificare binară, pe ultimul strat dens aplicăm funcția sigmoidă. Ca funcție de cost, am folosit entropia binară încrucișată, optimizatorul RMSprop, 20 de epoci, dimensiunea lotului

de 30, dimensiunea imaginii fiind 150x150,3. Unde dimensiunea imaginii este definită astfel: (lățime, înălțime, număr de canale).

Tabelul 7.3.3.1 prezintă precizia de validare și funcția de cost de validare pentru funcțiile de activare în cazul unei arhitecturi pre-antrenate VGG.

Tabel 7.3.3. 1 Rezultate experimentale pentru VGG16 pre-antrenat pe setul de date Câini și Pisici

Funcția de activare	**Acuratețea pe setul de validare**	**Funcția de cost pe setul de validare**
Sigmoida	89.81%	0.2246
Tanh	91.67%	0.2665
ReLU	91.45%	0.2835
Swish	91.25%	**0.1899**
TSReLU	**91.80%**	0.2305
TSReLU învățabilă	91.58%	0.2286
TBSRelu	91.32%	0.1992
TBSRelu învățabilă	90.81%	0.2190

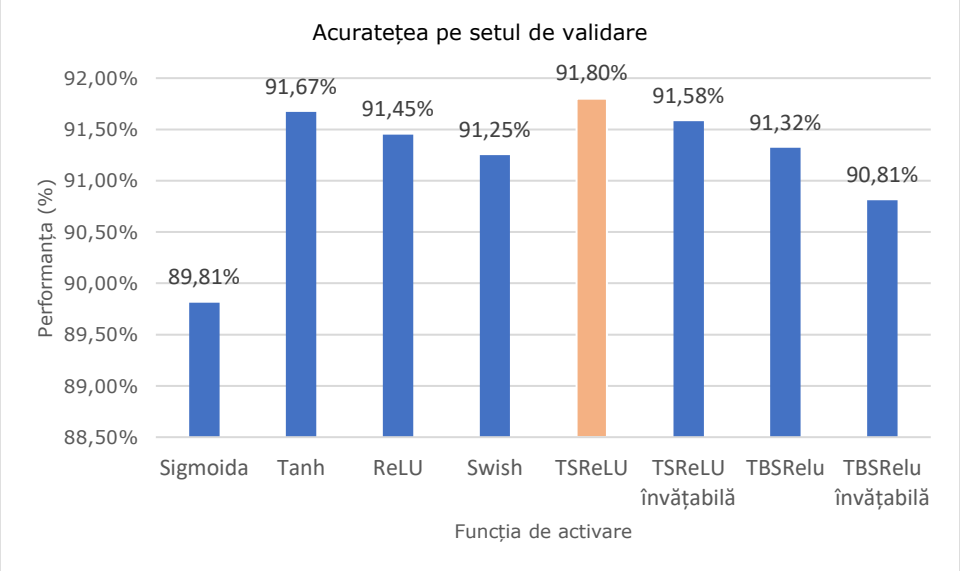


Fig.7.3.3. 1 Rezultate experimentale pe arhitectura VGG16 pre-antrenată pentru setul de date Câini și Pisici

Se poate observa din Fig. 7.3.3.1. că pe arhitectura VGG16 pre-antrenată funcția de activare TSReLU are cea mai bună performanță de 91.80%, iar cea mai mică funcție de cost o are funcția Swish 0.1899.

7.4. P-Swish: Funcție de activare cu parametri învățabili bazată pe funcția Swish în Învățarea Profundă - Partea experimentală

În cadrul acestui subcapitol prezint rezultatele experimentale privind o nouă funcție de activare numită P-Swish atât pe sarcini de Computer Vision cât și NLP. Din punct de vedere al arhitecturii am folosit:

- LeNet-5 [21]
- Rețea în Rețea (Network in Network) [123]
- ResNet34 [33]
- O arhitectură Xception [159] pre-antrenată
- O arhitectură MobileNetV1 pre-antrenată [158]
- O arhitectură MobileNetV2 [158] pre-antrenată pe sarcini de Computer Vision
- Arhitecturi personlizate pentru NLP
- O arhitectură MobileNetV2 pre-antrenată pe o sarcină de segmentare semantică [282].

7.4.1. Experimentul 1: Arhitectura LeNet-5 pe setul de date CIFAR-10

Am folosit arhitectura LeNet-5 cu regularizarea lotului (batch normalization) pe setul de date CIFAR-10, am antrenat 50 epoci și am salvat cel mai bun model (*Model Checkpoint*). Rata de învățare a fost 0.0001.

Tabel 7.4.1. 1 Rezultate experimentale - LeNet-5 pe setul de date CIFAR-10

Funcția de activare	**Acuratețea pe setul de validare**	**Funcția de cost pe setul de validare**
ReLU	69,63%	0,9421
Sigmoida	56,41%	1,2931
Swish	71,38%	0,8863
P-swish învățabilă α și β învățabili și δ = 1	70,3%	0,8955
P-swish constantă α = 1, β = 0 și δ=1	70,85%	0,9014
P-swish învățabilă parțial α învățabil (1) și β = 0, δ =1	**71,52%**	**0,8695**
P-swish învățabilă parțial α învățabil (1), β = 0 și δ învățabil (1)	69,17%	0,9535

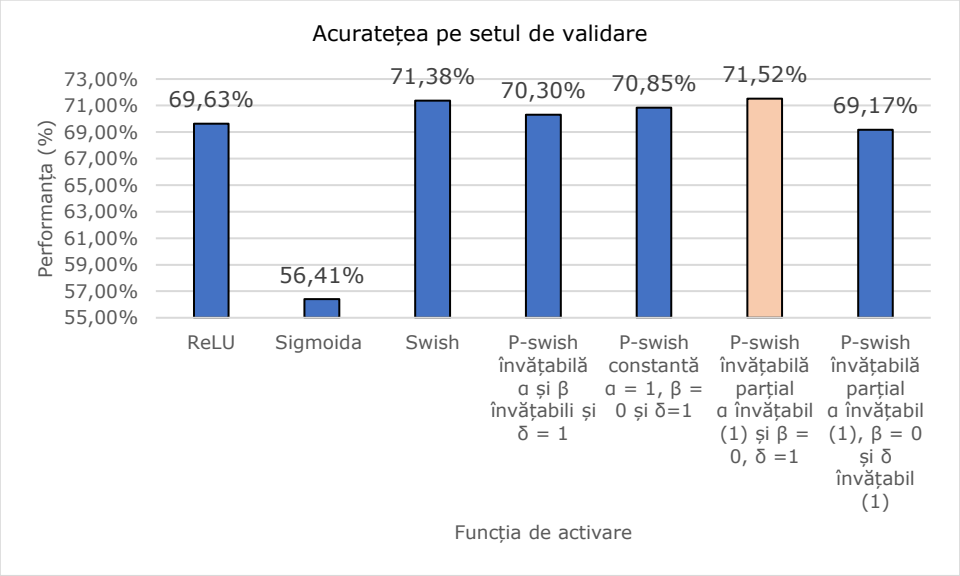


Fig.7.4.1. 1 Rezultate experimentale cu arhitectura LeNet-5 pe setul de date CIFAR-10

Se poate observa din Fig.7.4.1.1. că folosind arhitectura LeNet-5 cu regularizarea lotului pe setul de date CIFAR-10, P-Swish învățabilă parțial are cea mai bună performanță de 71.52% și cea mai mică funcție de cost 0.8695.

7.4.2. Experimentul 2: Arhitectura LeNet-5 pe setul de date CIFAR-100

Am folosit arhitectura LeNet-5 cu regularizarea lotului pe setul de date CIFAR-10, am antrenat 100 epoci și am salvat cel mai bun model (*Model Checkpoint*). Rata de învățare a fost de 0.01.

Tabel 7.4.2. 1 Rezultate experimentale - LeNet-5 pe setul de date CIFAR-100

Funcția de activare	**Acuratețea pe setul de validare**	**Funcția de cost pe setul de validare**
ReLU	35,66%	2,6106
Sigmoida	21,83%	3,3513
Swish	37,75%	2,5694
P-swish învățabilă α și β învățabili și δ = 1	36,81%	2,5651
P-swish constantă α = 1, β = 0 și δ=1	**38,29%**	**2,4980**
P-swish învățabilă parțial α învățabil (1) și β = 0, δ =1	36,71%	2,5866
P-swish învățabilă parțial α învățabil (1), β = 0 și δ învățabil (1)	35,91%	2,6435

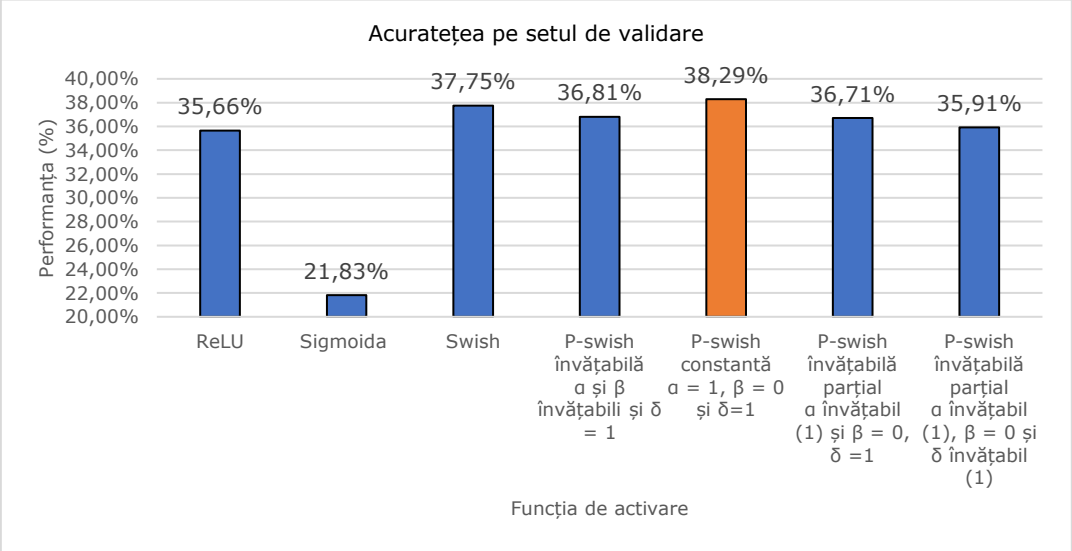


Fig.7.4.2. 1 Rezultate experimentale cu arhitectura LeNet-5 pe setul de date CIFAR-100

Se poate observa din Fig.7.4.2.1. că folosind arhitectura LeNet-5 cu regularizarea lotului pe setul de date CIFAR-100, P-Swish constantă are cea mai bună performanță de 38.29% și cea mai mică funcție de cost 2.4980.

7.4.3. Experimentul 3: Arhitectura NiN pe setul de date CIFAR-10

Am folosit arhitectura NiN cu regularizarea L_2 pe setul de date CIFAR-10, am antrenat 50 epoci și am salvat cel mai bun model prin metoda *Model Checkpoint*. Rata de învățare a fost 0.04.

Tabel 7.4.3. 1 Rezultate experimentale - NiN pe setul de date CIFAR-10

Funcția de activare	**Acuratețea pe setul de validare**	**Funcția de cost pe setul de validare**
ReLU	85,73%	0,6810
Sigmoida	87,24%	0,6115
Swish	**87,74%**	0,5845
P-swish învățabilă α și β învățabili și δ = 1	87,52%	**0,5124**
P-swish constantă α = 1, β = 0 și δ=1	87,06%	0,6262
P-swish învățabilă parțial α învățabil (1) și β = 0, δ =1	87,23%	0,6141
P-swish învățabilă parțial α învățabil (1), β = 0 și δ învățabil (1)	86,52%	0,6427

Se poate observa din Tabelul 7.4.3.1. că folosind arhitectura NiN cu regularizarea lotului pe setul de date CIFAR-10, cea mai bună performanță de 87.74% este dată de Swish dar totuși funcția P-Swish nu dă rezultate cu mult mai slabe și spre deosebire de restul funcțiilor P-Swish învățabilă α și β învățabili și δ = 1 dă cea mai mică funcție de cost de 0,5124.

7.4.4. Experimentul 4: Arhitectura NiN pe setul de date CIFAR-100

Am folosit arhitectura NiN cu regularizarea lotului L_2 pe setul de date CIFAR-100, am antrenat 100 epoci și am salvat cel mai bun model. Rata de învățare a fost 0.01.

Tabel 7.4.4. 1 Rezultate experimentale - NiN pe setul de date CIFAR-100

Funcția de activare	**Acuratețea pe setul de validare**	**Funcția de cost pe setul de validare**
ReLU	65,79%	1,7330
Sigmoida	28,00%	4,0609
Swish	67,30%	1,7316
P-swish învățabilă α și β învățabili și δ = 1	66,88%	1,7674
P-swish constantă α = 1, β = 0 și δ=1	**67,53%**	**1,7166**
P-swish învățabilă parțial α învățabil (1) și β = 0, δ =1	66,61%	1,7628
P-swish învățabilă parțial α învățabil (1), β = 0 și δ învățabil (1)	67,34%	1,7229

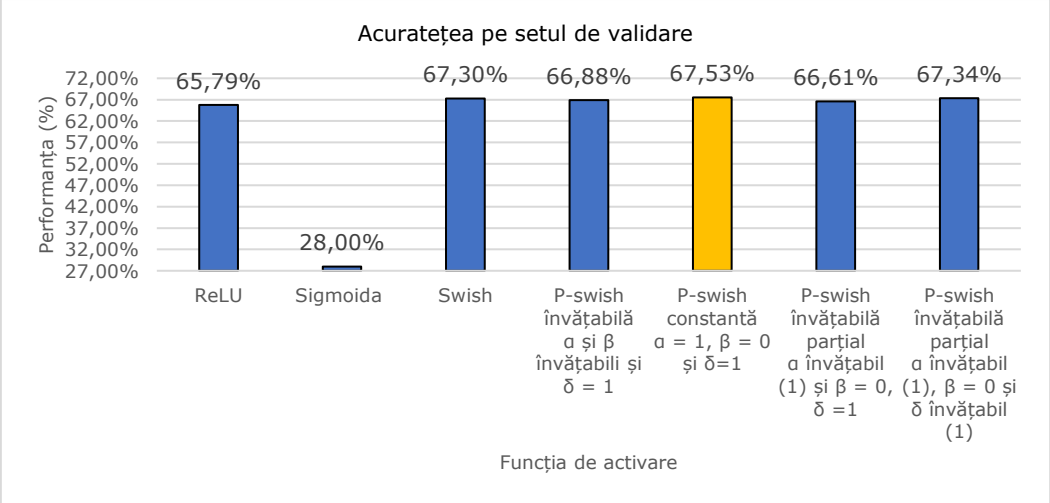


Fig.7.4.4. 1 Rezultate experimentale cu arhitectura NiN pe setul de date CIFAR-100

Se poate observa din Fig.7.4.4.1. că folosind arhitectura NiN cu regularizarea lotului pe setul de date CIFAR-100, cea mai bună performanță de 67,53% este dată de P-swish constantă α = 1, β = 0 și δ=1, urmată de P-swish învățabilă parțial α învățabil (inițializat cu 1), β = 0 și δ învățabil (inițializat cu 1) care nu dă rezultate cu mult mai slabe decât P-swish constantă.

7.4.5. Experimentul 5: Arhitectura ResNet34 pe setul de date CIFAR-10

Am folosit arhitectura ResNet34 cu regularizarea lotului, dimensiunea lotului de 128, pe setul de date CIFAR-10, am antrenat 50 epoci și am salvat cel mai bun model.

Tabel 7.4.5. 1 Rezultate experimentale - ResNet34 pe setul de date CIFAR-10

Funcția de activare	**Acuratețea pe setul de validare**	**Funcția de cost pe setul de validare**
ReLU	86,35%	0,6730
Sigmoida	75,73%	0,9428
Swish	88,16%	0,6077
P-swish învățabilă α și β învățabili și δ = 1	87,97%	0,5943
P-swish constantă α = 1, β = 0 și δ=1	88,28%	0,6284
P-swish învățabilă parțial α învățabil (1) și β = 0, δ =1	88,40%	0,5961
P-swish învățabilă parțial α învățabil (1), β = 0 și δ învățabil (1)	**88,67%**	**0,5715**

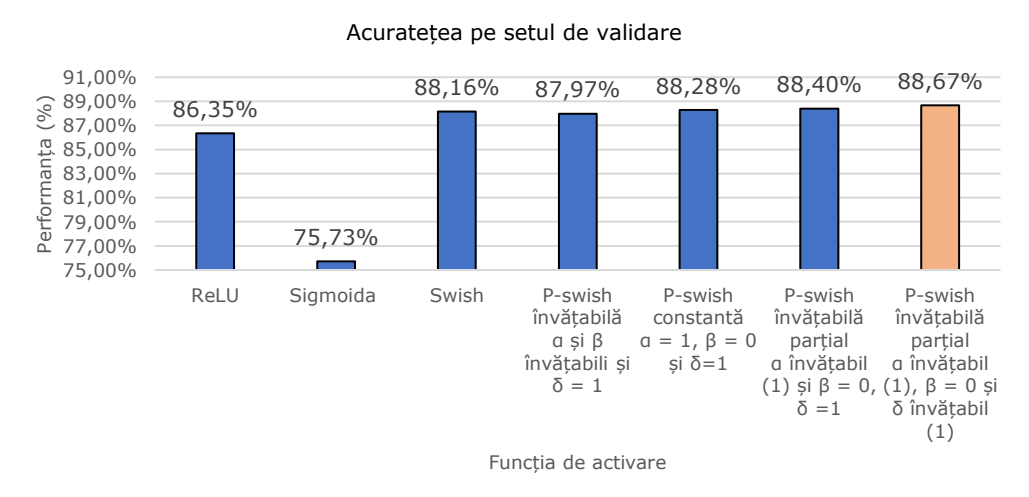


Fig.7.4.5. 1 Rezultate experimentale cu arhitectura ResNet34 pe setul de date CIFAR-10

Se poate observa din Fig.7.4.5.1 că folosind arhitectura RESNET34 cu regularizarea lotului pe setul de date CIFAR-10, cea mai bună performanță de 88,67% este dată de P-swish învățabilă parțial α învățabil (inițializat cu 1), β = 0 și δ învățabil (inițializat cu 1) și tot această funcție dă și cea mai mică funcție de cost de 0,5715.

7.4.6. Experimentul 6: Arhitectura ResNet34 pe setul de date CIFAR-100

Am folosit arhitectura ResNet34 cu regularizarea lotului, dimensiunea lotului de 128, pe setul de date CIFAR-100, am antrenat 100 de epoci și am salvat cel mai bun model.

Tabel 7.4.6. 1 Rezultate experimentale - ResNet34 pe setul de date CIFAR-100

Funcția de activare	**Acuratețea pe setul de validare**	**Funcția de cost pe setul de validare**
ReLU	68,69%	**1,7626**
Sigmoida	59,04%	1,919
Swish	69,38%	1,814
P-swish învățabilă α și β învățabili și δ = 1	**69,49%**	1,7765
P-swish constantă α = 1, β = 0 și δ=1	69,36%	1,7718
P-swish învățabilă parțial α învățabil (1) și β = 0, δ =1	69,16%	1,8212
P-swish învățabilă parțial α învățabil (1), β = 0 și δ învățabil (1)	68,95%	1,802

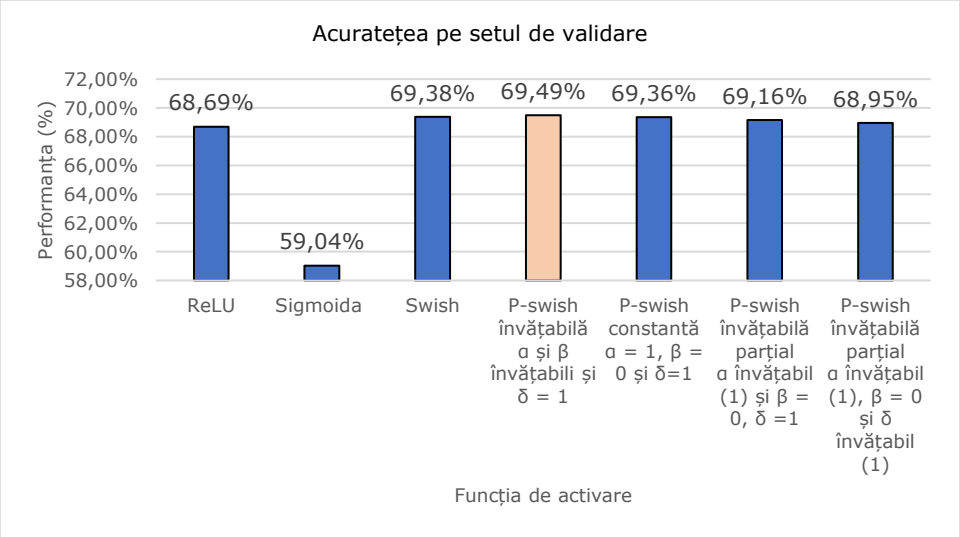


Fig.7.4.6. 1 Rezultate experimentale cu arhitectura ResNet34 pe setul de date CIFAR-100

Se poate observa din Fig.7.4.6.1. că folosind arhitectura RESNET34 cu regularizarea lotului pe setul de date CIFAR-100, cea mai bună performanță de **69,49%** este dată de P-swish învățabilă α și β învățabili și δ = 1 și funcția ReLU dă cea mai mică funcție de cost de **1,7626**.

În subcapitolele anterioare am putut vedea că propunerea mea a oferit cea mai bună precizie. Asta o face să fie o soluție puternică și robustă în comparație cu funcțiile ReLU, sigmoida și Swish.

7.4.7. Experimentul 7: Arhitectura Xception pre-antrenată pe setul de date CIFAR-10

Am folosit arhitectura Xception pre-antrenată cu regularizarea lotului, dimensiunea lotului de 128, pe setul de date CIFAR-10, am antrenat 50 epoci și am salvat cel mai bun model.

Tabel 7.4.7. 1 Rezultate experimentale - Xception pre-antrenată pe setul de date CIFAR-10

Funcția de activare	**Acuratețea pe setul de validare**	**Funcția de cost pe setul de validare**
ReLU	87,96%	0,4696
Sigmoida	88,19%	0,4621
Swish	88,01%	0,5152
P-swish învățabilă α și β învățabili și δ = 1	87,95%	0,4999
P-swish constantă α = 1, β = 0 și δ=1	87,77%	0,4777
P-swish învățabilă parțial α învățabil (1) și β = 0, δ =1	**88,44%**	**0,4362**
P-swish învățabilă parțial α învățabil (1), β = 0 și δ învățabil (1)	88,22%	0,4924

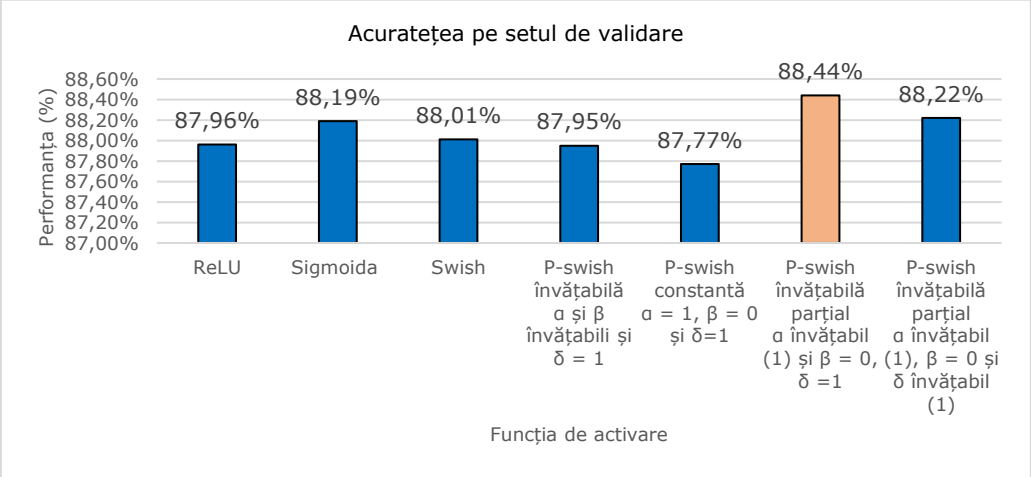


Fig.7.4.7. 1 Rezultate experimentale - arhitectura Xception pre-antrenată pe setul de date CIFAR-10

Se poate observa din Fig.7.4.7.1. că folosind arhitectura Xception pre-antrenată cu regularizarea lotului pe setul de date CIFAR-10, cea mai bună performanță de **88,22%** este dată de P-swish învățabilă parțial α învățabil (inițializat cu 1), β = 0 și δ învățabil (inițializat cu 1) și că funcția P-swish învățabilă parțial α învățabil (inițializat cu 1) și β = 0, δ =1 dă cea mai mică funcție de cost de **0,4362.**

7.4.8. Experimentul 8: Arhitectura Xception pre-antrenată pe setul de date CIFAR-100

Am folosit arhitectura Xception pre-antrenată cu regularizarea lotului, dimensiunea lotului fiind 128, pe setul de date CIFAR-100, am antrenat 100 de epoci și am salvat cel mai bun model.

Tabel 7.4.8. 1 Rezultate experimentale - Xception pre-antrenată pe setul de date CIFAR-100

Funcția de activare	**Acuratețea pe setul de validare**	**Funcția de cost pe setul de validare**
ReLU	61,07%	1,9328
Sigmoida	60,51%	2,0312
Swish	61,90%	**1,8177**
P-swish învățabilă α și β învățabili și δ = 1	59,67%	1,9147
P-swish constantă α = 1, β = 0 și δ=1	**62,05%**	1,9425
P-swish învățabilă parțial α învățabil (1) și β = 0, δ =1	61,90%	1,8351
P-swish învățabilă parțial α învățabil (1), β = 0 și δ învățabil (1)	56,52%	1,8343

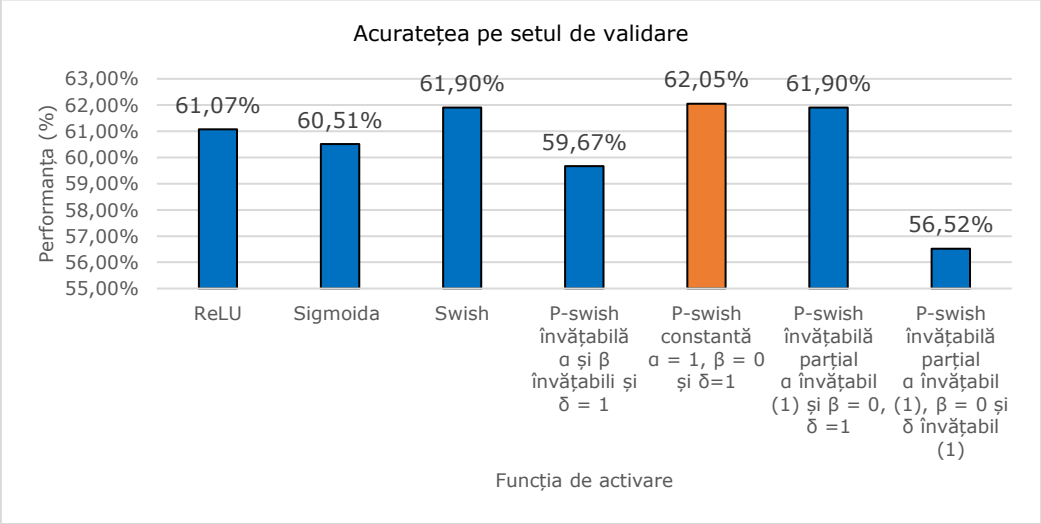


Fig.7.4.8. 1 Rezultate experimentale cu arhitectura Xception pre-antrenată pe setul de date CIFAR-100

Se poate observa din Fig.7.4.8.1. că folosind arhitectura Xception pre-antrenată cu regularizarea lotului pe setul de date CIFAR-100, cea mai bună performanță de **62,05%** este dată de P-swish constantă α = 1, β = 0 și δ=1 iar funcția Swish dă cea mai mică funcție de cost de **1,8177**.

7.4.9. Experimentul 9: Arhitectura MobileNetV1 pre-antrenată pe setul de date CIFAR-10

Am folosit arhitectura MobileNetV1 pre-antrenată cu regularizarea lotului, dimensiunea lotului de 128, pe setul de date CIFAR-10, am antrenat 50 epoci și am salvat cel mai bun model.

Tabel 7.4.9. 1 Rezultate experimentale - MobileNetV1 pre-antrenată pe setul de date CIFAR-10

Funcția de activare	**Acuratețea pe setul de validare**	**Funcția de cost pe setul de validare**
ReLU	79,35%	0,6366
Sigmoida	**84,53%**	**0,4835**
Swish	75,69%	0,7175
P-swish învățabilă α și β învățabili și δ = 1	72,35%	0,7985
P-swish constantă α = 1, β = 0 și δ=1	83,03%	0,5221
P-swish învățabilă parțial α învățabil (1) și β = 0, δ =1	74,52%	0,7854
P-swish învățabilă parțial α învățabil (1), β = 0 și δ învățabil (1)	80,57%	0,6285

Se poate observa din Tabelul 7.4.9.1. că folosind arhitectura MobileNetV1 pre-antrenată cu regularizarea lotului pe setul de date CIFAR-10 cea mai bună performanță de **84,53%** este dată de Sigmoidă și tot funcția Sigmoidă dă cea mai mică funcție de cost de **0,4835.** Dar și funcțiile P-swish învățabilă parțial α învățabil (inițializat cu 1), β = 0 și δ învățabil (inițializat cu 1) și P-swish constantă α = 1, β = 0 și δ=1 dau rezultate bune.

7.4.10. Experimentul 10: Arhitectura MobileNetV2 pre-antrenată pe setul de date CIFAR-10

Am folosit arhitectura MobileNetV2 pre-antrenată cu regularizarea lotului, dimensiunea lotului de 128, pe setul de date CIFAR-10, am antrenat 50 epoci și am salvat cel mai bun model.

Tabel 7.4.10. 1 Rezultate experimentale - MobileNetV2 pre-antrenată pe setul de date CIFAR-10

Funcția de activare	**Acuratețea pe setul de validare**	**Funcția de cost pe setul de validare**
ReLU	71,48%	0,8745
Sigmoida	**79,27%**	**0,6759**
Swish	67,36%	0,9688
P-swish învățabilă α și β învățabili și δ = 1	69,01%	0,9042
P-swish constantă α = 1, β = 0 și δ=1	61,77%	1,1106
P-swish învățabilă parțial α învățabil (1) și β = 0, δ =1	59,98%	1,1616
P-swish învățabilă parțial α învățabil (1), β = 0 și δ învățabil (1)	63,98%	1,0278

Se poate observa din Tabelul 7.4.10.1. că folosind arhitectura MobileNetV2 pre-antrenată cu regularizarea lotului pe setul de date CIFAR-10 cea mai bună performanță de **79,27%** este dată de Sigmoidă și tot funcția Sigmoidă dă cea mai mică funcție de cost de **0,6759.** Dar cu toate acestea și funcția P-swish învățabilă α și β învățabili și δ = 1 se află în top 3 cele mai bune rezultate.

7.4.11. Experimentul 11: Arhitectură personalizată pentru o sarcină de NLP pe setul de date Reuters

Setul de date Reuters [192] este un set de date cu 11.228 de știri de la Reuters publicate în 1986, etichetate peste 46 de subiecte. Este un set de date simplu, folosit la scară largă pentru clasificarea textului. Printre cele 46 de topicuri distincte, unele au o pondere mai mare decât altele, dar fiecare topic are cel puțin 10 exemple în setul de antrenare. Are 90 de clase, 7769 documente de antrenare și 3019 documente de test.

Ca arhitectură, am definit în Tensorflow 2.0 un model secvențial, cu 3 straturi dense care au avut ca dimensiuni (128, 128 și 64 unități) iar ultimul strat a fost un dens egal cu numărul de clase 46 de unități cu funcția de activare Softmax deoarece avem o clasificare de mai multe clase. Am folosit tehnica Dropout succesiv și regulizările $l_1 = 0.01$ și $l_2 = 0.01$. Am folosit oprire timpurie (timp de așteptare 5 epoci), am antrenat 50 epoci cu o dimensiune a lotului de 256.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 128)               1280128
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 46)                2990
=================================================================
Total params: 1,307,886
Trainable params: 1,307,886
Non-trainable params: 0
```

Fig.7.4.11. 1 Arhitectură personalizată pe setul de date Reuters

Tabel 7.4.11. 1 Rezultate experimentale pe setul de date REUTERS

Funcția de activare	**Acuratețea pe setul de validare**	**Funcția de cost pe setul de validare**	**Acuratețea pe setul de test**	**Funcția de cost pe setul de test**
ReLU	79,80%	**1,2200**	75,60%	1,4856
Sigmoida	56,80%	2,1309	55,30%	2,2352
Swish	77,20%	1,3605	75,60%	1,4856
P-Swish învățabilă α și β învățabili δ = 1	**80,90%**	1,2630	76,44%	**1,4624**
P-Swish constantă α = 1, β = 0 și δ=1	80,70%	1,2904	74,93%	1,5098
P-Swish învățabilă parțial α învățabil (1) și β = 0, δ=1	80,80%	1,2600	75,56%	1,4882
P-Swish învățabilă parțial α învățabil (1), β = 0 și δ învățabil (1)	80,00%	1,2796	**77,33%**	1,517

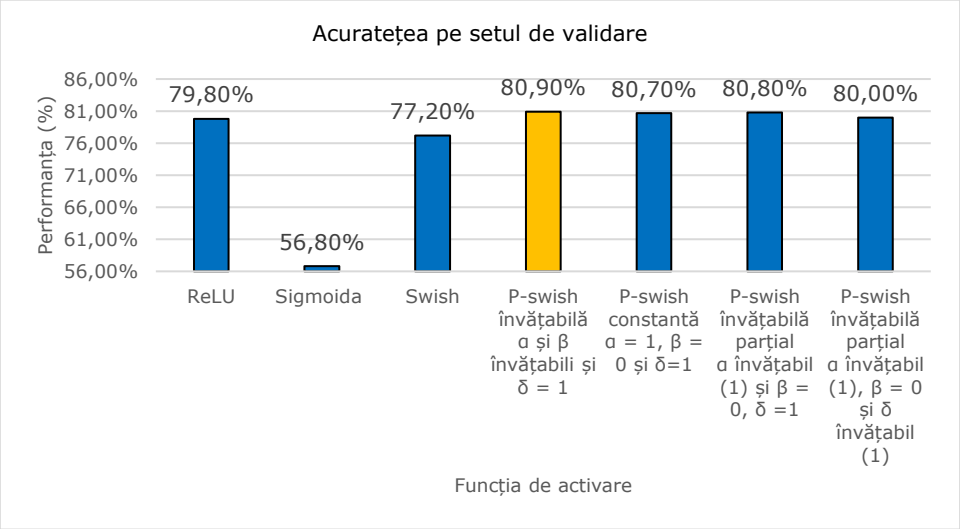


Fig.7.4.11. 2 Rezultate experimentale pe setul de validare Reuters

Se poate observa din Fig.7.4.11.2. că cea mai bună performanță pe setul de validare a fost atinsă de funcția de activare P-Swish învățabilă α și β învățabili și δ constant, δ = 1 de **80,90%**, iar funcția de cost cea mai mică **1,2200** a fost înregistrată de funcția de activare ReLU. Cu toate acestea, în figura Fig.7.4.11.3. se poate vedea că funcția de activare P-Swish învățabilă parțial α învățabil (inițializat cu 1), β = 0 și δ învățabil (inițializat cu 1) are cea mai bună performanță **77,33%** pe setul de test, iar cea mai mică funcție de cost a fost a funcției de activare P-Swish învățabilă α și β învățabili δ = 1 egală **cu 1,4624.** Se poate remarca faptul că noua funcție de activare se comportă foarte bine chiar și pe setul de test.

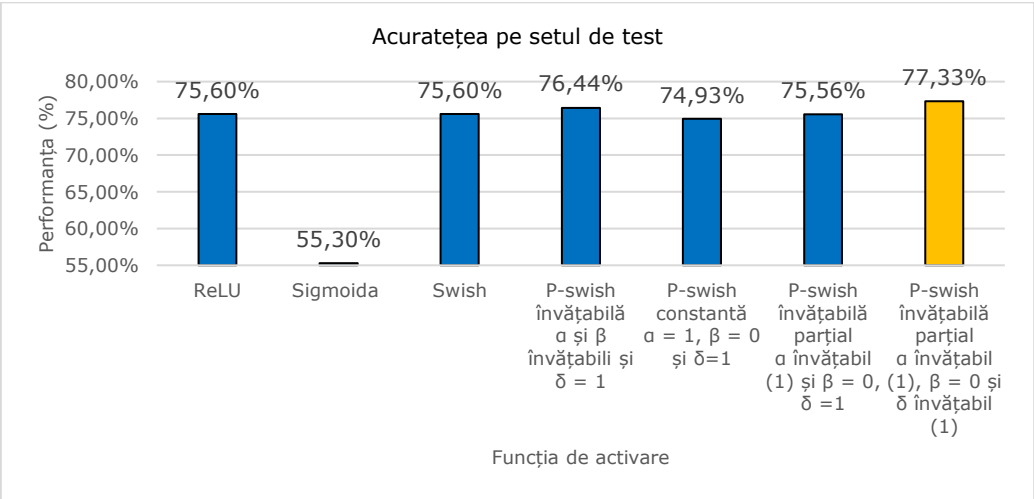


Fig.7.4.11. 3 Rezultate experimentale pe setul de test Reuters

7.4.12. Experimentul 12: Arhitectură personalizată pentru o sarcină de NLP pe setul de date IMDB

Setul de date IMDB [193] este un set de date cu 50000 de păreri (reviews) dintr-o Bază de date pentru filme de pe Internet, acest set este împărțit în 25000 date pentru antrenare și 25000 date pentru test, fiecare set conține 50% păreri negative și 50% păreri pozitive.

Ca arhitectură, am definit în Tensorflow 2.0 un model secvențial, cu 3 straturi dense care au avut ca dimensiuni (64, 32 și 16 unități) iar ultimul strat a fost un dens egal cu numărul de clase, în acest caz 1 unitate cu funcția de activare sigmoidă fiind cazul unei clasificări binare. Am folosit tehnicile dropout succesiv (0.4, 0.3, 0.2). Am antrenat 20 epoci cu o dimensiune a lotului de 512. Ca și optimizator am folosit SGD, cu o rată de învățare lr=0.1, momentum = 0.9, nesterov = *Adevărat*, (*Nesterov accelerated gradient*) [283].

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 64)                640064
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                2080
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 16)                528
_________________________________________________________________
dropout_2 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 17
=================================================================
Total params: 642,689
Trainable params: 642,689
Non-trainable params: 0
_________________________________________________________________
```

Fig.7.4.12. 1 Arhitectură personalizată pe setul de date IMDB

Tabel 7.4.12. 1 Rezultate experimentale pe setul de date IMDB

Funcția de activare	**Acuratețea pe setul de validare**	**Funcția de cost pe setul de validare**
ReLU	86,57%	0,554
Sigmoida	50,02%	0,6906
Swish	86,33%	0,666
P-Swish învățabilă α și β învățabili δ = 1	86,54%	0,633
P-Swish constantă α = 1, β = 0 și δ=1	86,60%	0,6327
P-Swish învățabilă parțial α învățabil (1) și β = 0, δ =1	**86,66%**	**0,3792**
P-Swish învățabilă parțial α învățabil (1), β = 0 și δ învățabil (1)	80,68%	0,9783

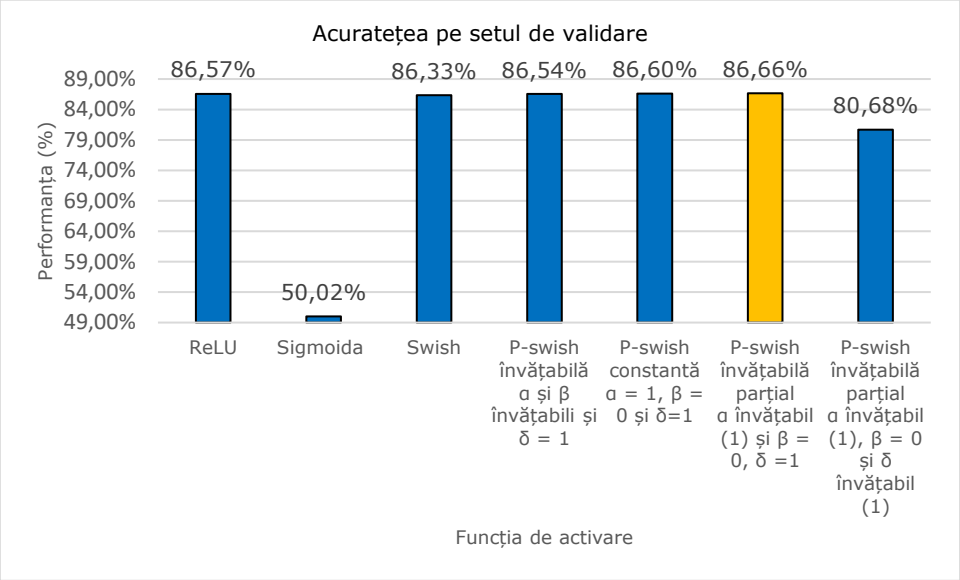


Fig.7.4.12. 2 Rezultate experimentale pe setul de validare IMDB

Se poate observa din Fig.7.4.12.2 că cea mai bună performanță pe setul de validare a fost atinsă de funcția de activare P-Swish învățabilă parțial α învățabil (inițializat cu 1) și β = 0, δ =1 de **86,66%**, iar funcția de cost cea mai mică **0,3792** a fost înregistrată de aceiași funcție de activare.

7.4.13. Experimentul 13: Arhitectură personalizată folosind LSTM pentru o sarcină de NLP pe setul de date FastText crawl 300d 2M

Setul de date FastText crawl 300d 2M [194] lansat de Facebook (**300-dimensional pretrained FastText English word vectors)** este un set de date cu vectori de cuvinte în limba engleză FastText pre-antrenat. Setul de date conține 5331 de date.

```
[13]: (0                          simplistic , silly and tedious .
       1        it's so laddish and juvenile , only teenage bo...
       2        exploitative and largely devoid of the depth o...
       3        [garbus] discards the potential for pathologic...
       4        a visually flashy but narratively opaque and e...
                                       ...
       10657    both exuberantly romantic and serenely melanch...
       10658    mazel tov to a film about a family's joyous li...
       10659    standing in the shadows of motown is the best ...
       10660    it's nice to see piscopo again after all these...
       10661    provides a porthole into that noble , tremblin...
       Name: text, Length: 10662, dtype: object,
       array([0, 0, 0, ..., 1, 1, 1]))
```

Fig.7.4.13. 1 Date pozitive și negative din setul de date FastText crawl 300d 2M

Ca arhitectură, am definit în Tensorflow 2.0 un model de tip functional API, care avea un strat Embedding, un strat de SpatialDropout = 0.2, 2 straturi de Batch Normalization, un strat bidirectional tip LSTM cu 64 unități, un strat de Dropout = 0.5, un strat dens de 128 și în final un strat dens de 2 cu o funcție de activare Softmax. Ca și funcție de cost am folosit "sparse categorical crossentropy", optimizator Adam și o rată de învățare lr = 0.0004, dimensiunea lotului de 256.

```
Model: "model"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input_1 (InputLayer)         [(None, 64)]              0
_________________________________________________________________________________________________
embedding (Embedding)        (None, 64, 600)           11672400    input_1[0][0]
_________________________________________________________________________________________________
spatial_dropout1d (SpatialDropo (None, 64, 600)        0           embedding[0][0]
_________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 64, 600)        2400        spatial_dropout1d[0][0]
_________________________________________________________________________________________________
bidirectional (Bidirectional)  (None, 128)             340480      batch_normalization[0][0]
_________________________________________________________________________________________________
dropout (Dropout)            (None, 128)               0           bidirectional[0][0]
_________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 128)            512         dropout[0][0]
_________________________________________________________________________________________________
dense (Dense)                (None, 128)               16512       batch_normalization_1[0][0]
_________________________________________________________________________________________________
add (Add)                    (None, 128)               0           batch_normalization_1[0][0]
                                                                   dense[0][0]
_________________________________________________________________________________________________
dense_1 (Dense)              (None, 2)                 258         add[0][0]
=================================================================================================
Total params: 12,032,562
Trainable params: 12,031,106
Non-trainable params: 1,456
```

Fig.7.4.13. 2 Arhitectură personalizată pe setul de date FastText crawl 300d 2M

Tabel 7.4.13. 1 Rezultate experimentale pe setul de date FastText crawl 300d 2M

Funcția de activare	**Acuratețea pe setul de validare**	**Funcția de cost pe setul de validare**
ReLU	77,04%	0,9496
Sigmoida	77,23%	**0,4833**
Swish	77,30%	0,5472
P-Swish învățabilă α și β învățabili δ = 1	**78,69%**	0,7264
P-Swish constantă α = 1, β = 0 și δ=1	77,36%	0,5416
P-Swish învățabilă parțial α învățabil (1) și β = 0, δ =1	77,06%	0,5195
P-Swish învățabilă parțial α învățabil (1), β = 0 și δ învățabil (1)	77,73%	0,9282

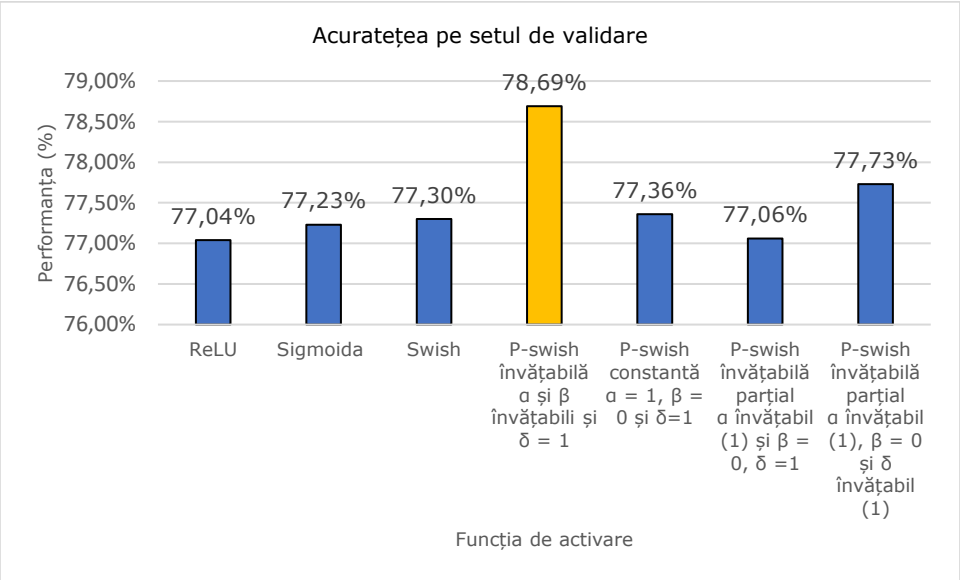


Fig.7.4.13. 3 Rezultate experimentale pe setul de validare FastText crawl 300d 2M

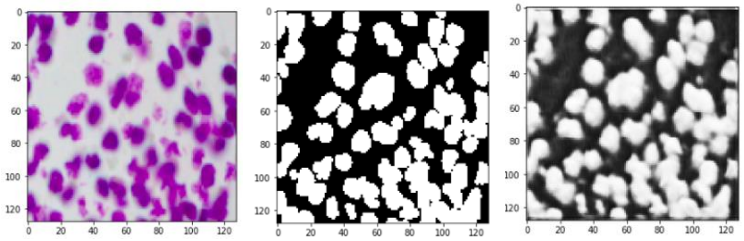
Se poate observa din Fig.7.4.13.3. că cea mai bună performanță pe setul de validare a fost atinsă de funcția de activare P-Swish învățabilă α și β învățabili δ = 1 de **78,69%**, iar funcția de cost cea mai mică **0,4833** a fost înregistrată de funcția de activare sigmoidă, dar următoarea funcție care are cea mai mică funcție de cost după sigmoidă este P-Swish învățabilă parțial α învățabil (1) și β = 0, δ =1 cu **0,5195.**

7.4.14. Experimentul 14: O arhitectură U-Net pe o sarcină de segmentare semantică

Setul de date 2018 Data Science Bowl [189] conține un număr mare de imagini nuclee segmentate. Imaginile au fost achiziționate într-o varietate de condiții și variază în funcție de tipul de celulă, de mărime și de caracteristicile imaginii (luminos versus fluorescență).

Acest set de date conține:

- imagini pentru fiecare nucleu
- măștile conțin măștile segmentate ale fiecărui nucleu. Măștile sunt incluse doar în setul de antrenare. Fiecare mască conține un nucleu. Măștile nu au voie să se suprapună (niciun pixel nu aparține la două măști).



a) b) c)

Fig.7.4.14. 1 Imagine originală (a) și masca corespondentă(b) masca generată (c) pe setul de date 2018 Data Science Bowl

Am definit în Tensorflow 2.0 un model de arhitectură de rețea complet conectată tip U-Net (*Fully Connected Convolutional Network - U-Net*) [284], pentru a genera masca corespondentă fiecărui nucleu. Are aproximativ 1,9 M parametri antrenabili. Am folosit o dimensiune a lotului egală cu 16, lr = 0,00001, împărțirea setului de date pentru validare de 10% și Adam ca optimizator. Modelul a fost antrenat 20 de epoci.

Pe lângă acuratețea pe setul de validare și funcția de cost pe setul de validare am luat în considerare și metrica IOU (*Intersection Over Union*) pe setul de validare. IoU, cunoscută și sub denumirea de *indicele Jaccard*, este cea mai utilizată metrică pentru compararea asemănării dintre două forme arbitrare. IoU codifică proprietățile de formă ale obiectelor analizate, de exemplu lățimile, înălțimile și pozițiile a două pătrate de delimitare a celor două regiuni și apoi calculează o metrică normalizată care se concentrează asupra suprapunerii lor. [285] Am înlocuit funcția „relu” cu propunerile mele, fiecare propunere în parte, pe toate straturile cu activare.

Tabel 7.4.14. 1 Rezultate experimentale pe setul de date 2018 Data Science Bowl

Funcția de activare	**Acuratețea pe setul de validare**	**Funcția de cost pe setul de validare**	**IOU pe setul de validare**
ReLU	96,88%	0,0757	55,42%
Sigmoida	81,02%	0,4435	0%
Swish	95,76%	0,1025	50,89%
P-Swish învățabilă α și β învățabili δ = 1	97,08%	**0,0721**	55,48%
P-Swish constantă α = 1, β = 0 și δ=1	**97,11%**	0,0757	55,52%
P-Swish învățabilă parțial α învățabil (1) și β = 0, δ =1	97,02%	0,0754	**56,14%**
P-Swish învățabilă parțial α învățabil (1), β = 0 și δ învățabil (1)	96,53%	0,0860	51,81%

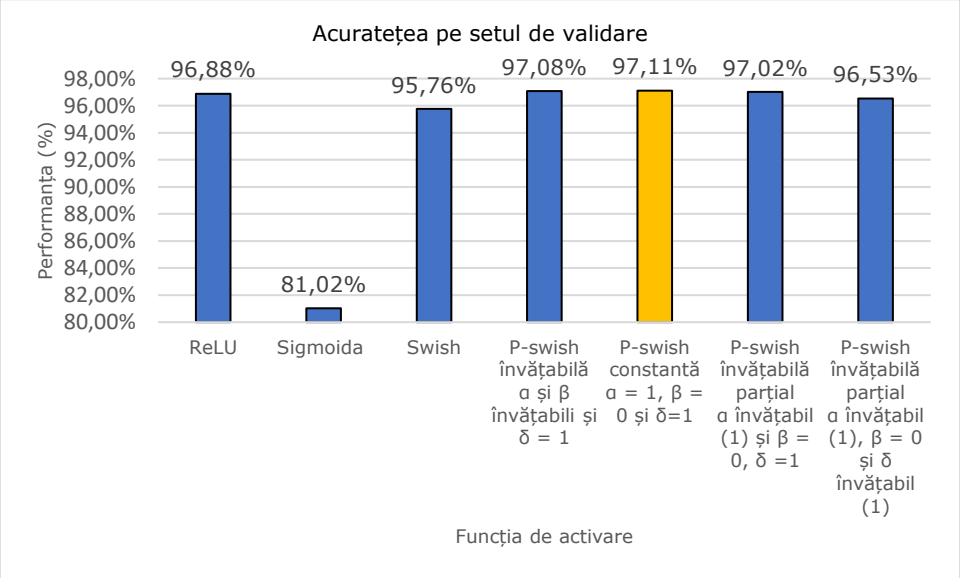


Fig.7.4.14. 2 Rezultate experimentale pe setul de validare 2018 Data Science Bowl

Se poate observa din Fig.7.4.14.2. că cea mai bună performanță pe setul de validare a fost atinsă de funcția de activare P-Swish constantă α = 1, β = 0 și δ=1 de **97,11%**, iar funcția de cost cea mai mică **0,0721** a fost dată de funcția P-Swish învățabilă α și β învățabili δ = 1 și cel mai bun IOU **56,14%** a fost înregistrat de funcția de activare P-Swish învățabilă parțial α învățabil (inițializat cu 1) și β = 0, δ =1.

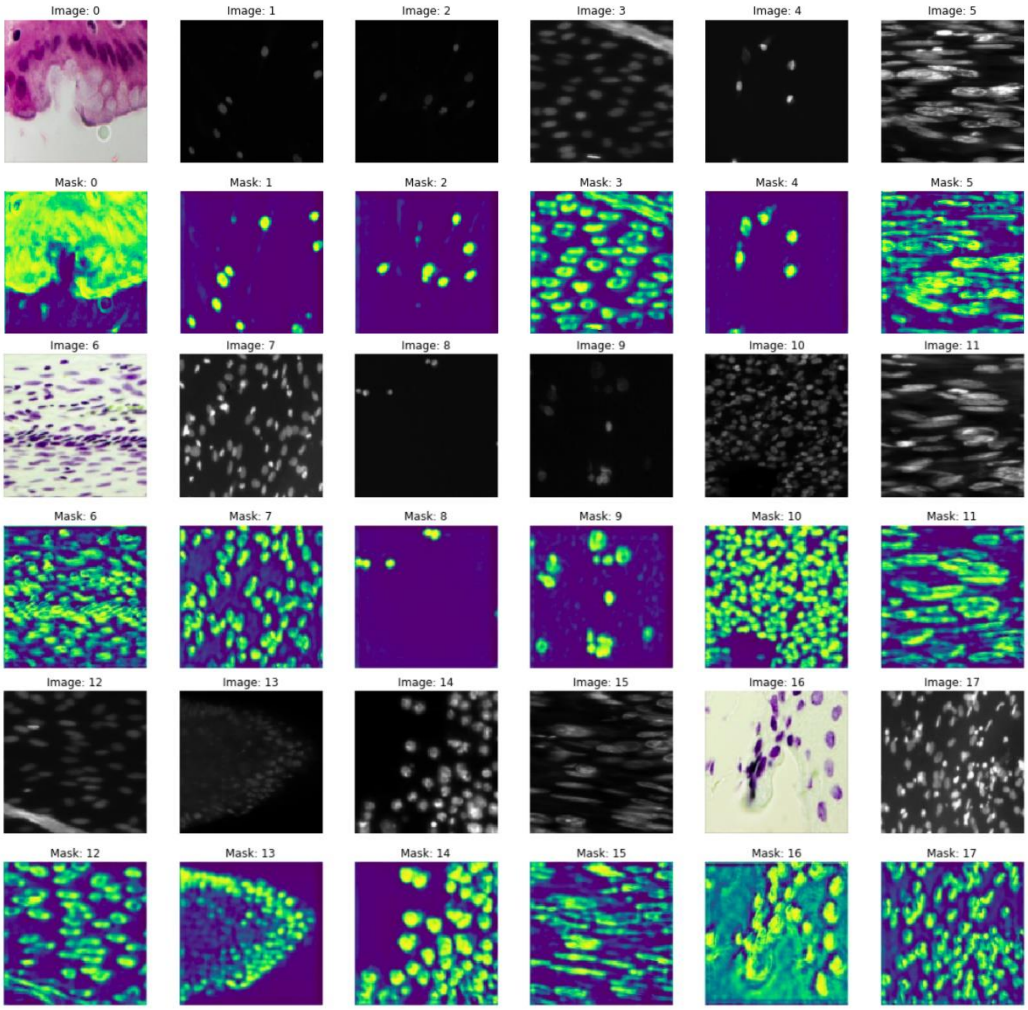


Fig.7.4.14. 3 Predicții pe setul de validare 2018 Data Science Bowl

În acest capitol, am propus o nouă funcție de activare numită **P-Swish** care aduce îmbunătățiri ale performanței pe seturi de date variate, variind tipul de sarcină în Învățarea Profundă. Caracterul său de funcție pe porțiuni este semnificativ aducând cea mai bună precizie în sarcinile NLP și în sarcina de segmentare semantică. De asemenea, P-Swish s-a comportat mult mai bine decât alte funcții în majoritatea cazurilor în sarcini de clasificare pentru recunoașterea obiectelor în Computer Vision.

7.5. Noi funcții de activare bazate pe funcția de activare TanhExp în Învățarea Profundă - Partea experimentală

În cadrul acestui subcapitol voi compara propunerile mele cu funcțiile de activare tanh, ReLU și TanhExp. Voi folosi arhitectura LeNet-5 [21], AlexNet [26] și arhitecturi personalizate.

7.5.1. Experimentul 1: Setul de date MNIST

Ca arhitectură, am folosit aceiași arhitectură ca în cazul funcției de activare TanhExp, adică o rețea de bază cu 15 straturi, pe care am antrenat-o pe setul de date MNIST [183] timp de 10 epoci. Alte informații, menționăm că am folosit o împărțire a setului de date, am folosit 20% din date pentru validare, iar dimensiunea lotului a fost 64.

Tabel 7.5.1. 1 Rezultate experimentale pe setul MNIST

Funcție de activare	***Acuratețe de validare***	***Funcție de cost de validare***
ReLU	98.61%	0.0566
tanh	98.33%	0.0590
TanhExp	98.60%	0.0609
TanhExp învățabilă	98.47%	**0.0547**
a_TanhExp	98.18%	0.0647
a_TanhExp învățabilă	**98.62%**	0.0593

În Tabel 7.5.1. 2 putem vedea că propunerea mea a_TanhExp învățabilă oferă cea mai bună precizie pe setul de date MNIST.

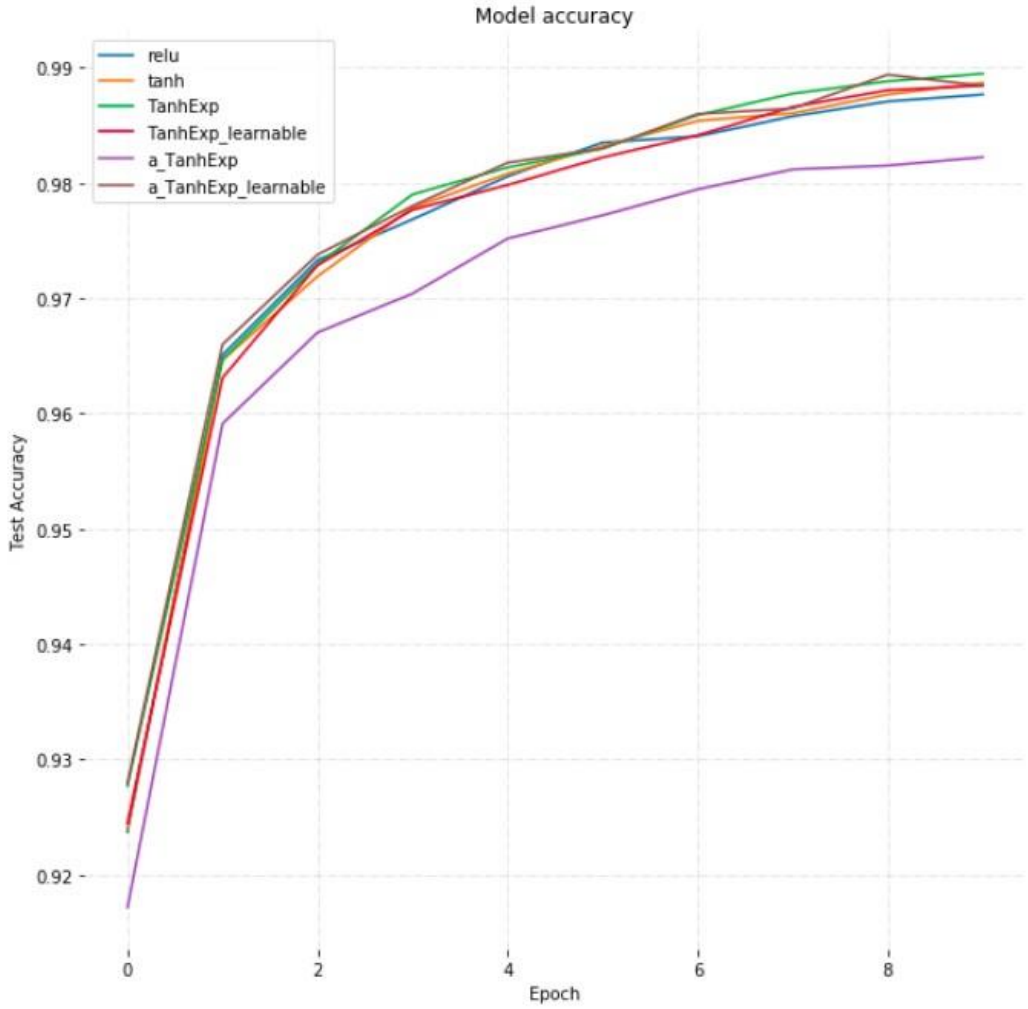


Fig.7.5.1. 1 Acuratețea modelului pe setul de validare pe MNIST

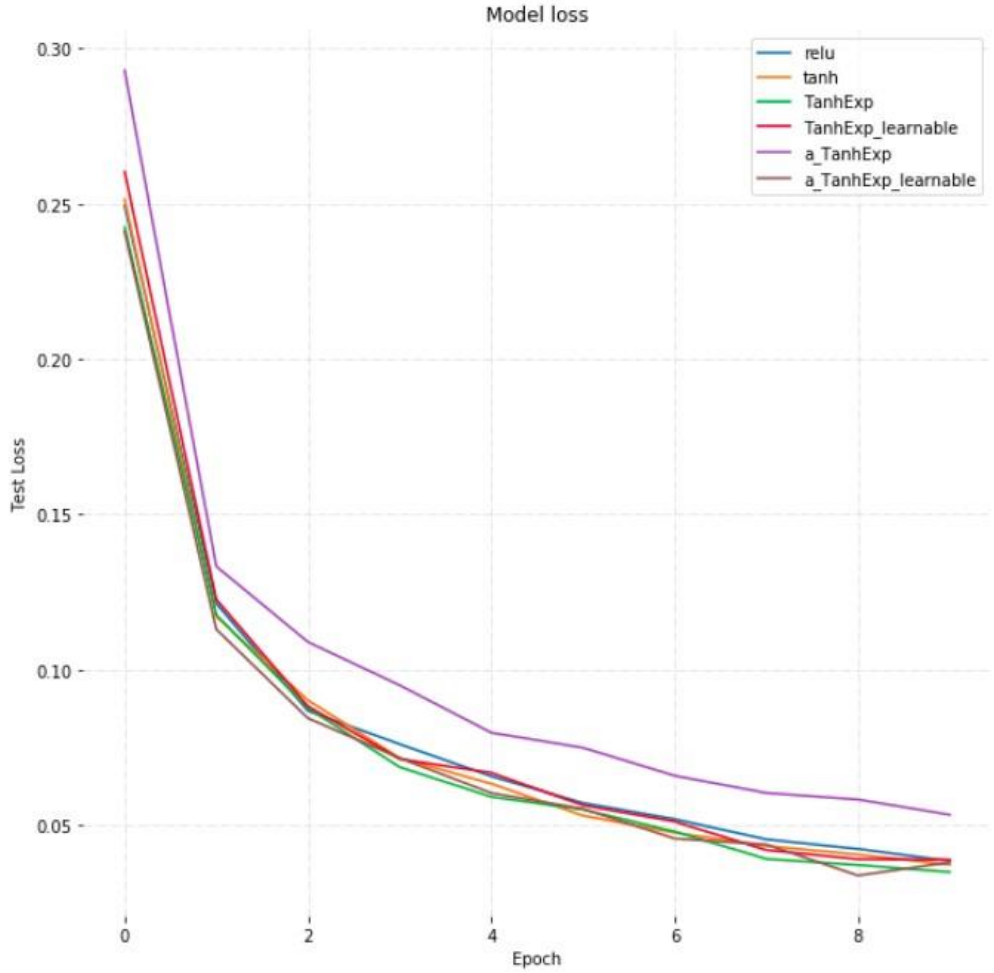


Fig.7.5.1. 2 Funcția de cost a modelului pe setul de validare pe MNIST

7.5.2. Experimentul 2: Setul de date Fashion-MNIST

Ca arhitectură, am folosit ca în cazul setului de date MNIST, rețeaua de bază cu 15 straturi, am antrenat pe setul de date Fashion-MNIST [279] pentru 20 de epoci fiind un set de date mai complex ca MNIST, am avut aceiași dimensiune a lotului și divizare de validare ca în cazul experimentelor pe MNIST.

Tabel 7.5.2. 1 Rezultate experimentale pe setul Fashion-MNIST

Funcție de activare	***Acuratețe de validare***	***Funcție de cost de validare***
ReLU	91.59%	0.2736
tanh	91.33%	0.3287
TanhExp	91.44%	0.3392
TanhExp învățabilă	82.50%	0.5243
a_TanhExp	91.53%	0.3353
a_TanhExp învățabilă	**91.63%**	0.3154

Și în acest caz din Tabel 7.5.2. 3 putem vedea că propunerea mea a_TanhExp învățabilă oferă cea mai bună precizie pe setul de date Fashion-MNIST.

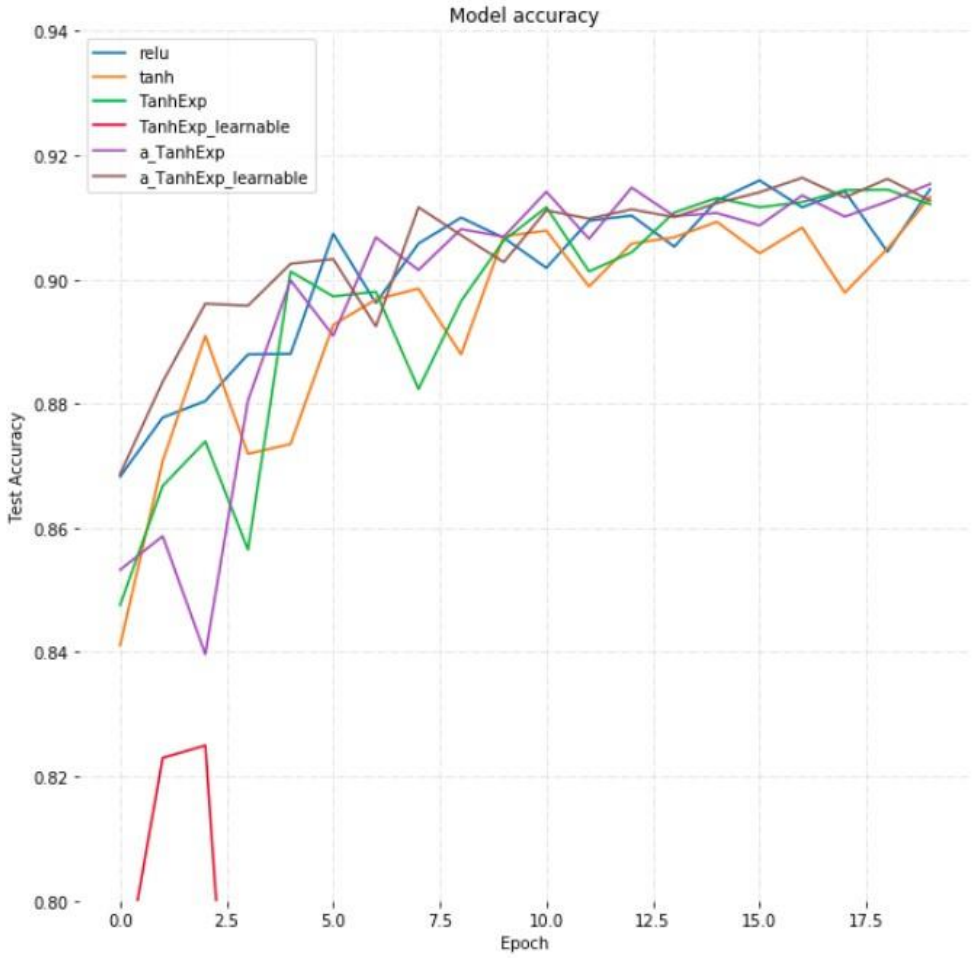


Fig.7.5.2. 1 Acuratețea modelului pe setul de validare pe Fashion-MNIST

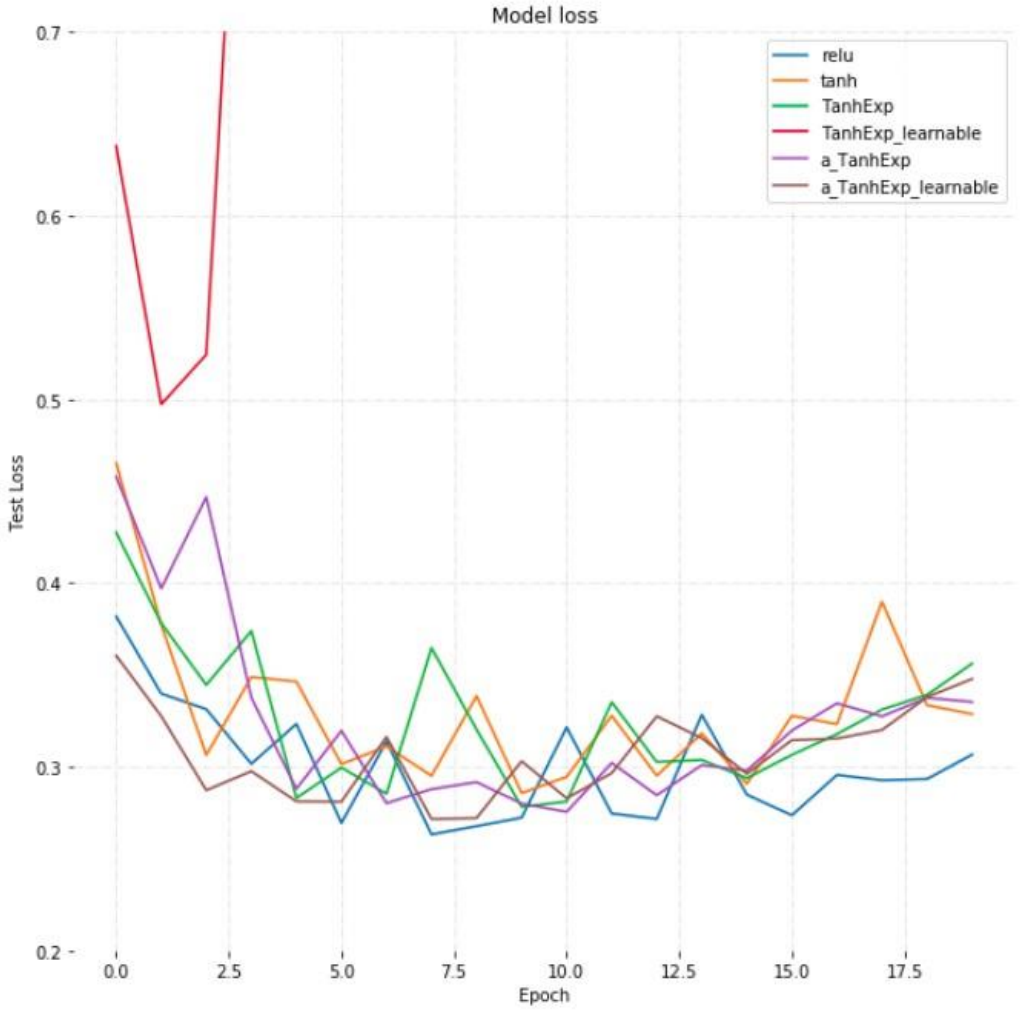


Fig.7.5.2. 2 Funcția de cost a modelului pe setul de validare pe Fashion-MNIST

7.5.3. Experimentul 3: Seturile de date CIFAR-10 și CIFAR-100

Ca arhitecturi, am folosit LeNet-5 și AlexNet, am antrenat pentru 20 de epoci pentru fiecare arhitectură în parte. Alte informații, menționăm că, în cazul arhitecturii LeNet-5, am folosit tehnica de normalizare a lotului [281], dimensiunea lotului fiind 128 și tehnica de augmentare a datelor (mărirea datelor) [286], am aplicat pe imagini deplasare orizontală aleatorie (*0,1*), pe verticală (*0.1*), setăm modul pentru umplerea punctelor în afara limitelor de intrare cu „cel mai apropiate”, setăm imaginile pe modul rotire pe orizontală (*Adevărat*) și 20% din date păstrăm pentru setul de validare. În cazul arhitecturii AlexNet, am folosit tehnica dropout [175]- [178] (rata dropout =

0,5) pe staturile 6 și 7, ca optimizator am folosit „rmsprop”, ca funcție de cost „sparse_categorical_crossentropy” și în cazul acestei arhitecturi am folosit tehnica de normalizare a lotului, dimensiunea lotului fiind 128 și tehnica de augmentare a datelor la fel ca în arhitectura LeNet-5.

Tabel 7.5.3. 1 Acuratețea de validare pe setul de date CIFAR-10

Funcție de activare	***LeNet-5***	***AlexNet***
ReLU	70.03%	83.32%
tanh	64.91%	75.43%
TanhExp	69.05%	83.98%
TanhExp învățabilă	44.21%	79.28%
a_TanhExp	69.74%	**84.97%**
a_TanhExp învățabilă	**70.28%**	83.19%

În Tabel 7.5.3.1 putem vedea că propunerile mele oferă cele mai bune valori pentru acurateţe pe setul de validare, a_TanhExp învățabilă pe LeNet-5 și a_TanhExp pe AlexNet. Deci, aceste rezultate le fac să fie o soluție puternică și robustă în comparație cu funcțiile ReLU, tanh și TanhExp.

Tabel 7.5.3. 2 Acuratețea de validare pe setul de date CIFAR-100

Funcție de activare	***LeNet-5***	***AlexNet***
ReLU	39.81%	60.62%
tanh	37.49%	53.57%
TanhExp	40.13%	60.99%
TanhExp învățabilă	**40.58%**	56.47%
a_TanhExp	37.70%	61.07%
a_TanhExp învățabilă	40.55%	**61.36%**

În Tabel 7.5.3.2 putem vedea că propunerea mea a_TanhExp învățabilă oferă cele mai bune valori pentru acurateţe pe setul de validare, de această dată pe ambele arhitecturi.

7.5.4. Experimentul 4: Detecția anomaliilor în seriile de timp

În acest subcapitol, am folosit setul de date Numenta Anomaly Benchmark (NAB) [287]. Acest set de date oferă date artificiale de serii temporale care conțin perioade cu comportament anormal etichetate. Seriile de timp sunt date care sunt măsurători ordonate, marcate în timp, cu o singură valoare. Am folosit fișierul `*art_daily_small_noise.csv*` pentru antrenare și fișierul `*art_daily_jumpsdown.csv*` pentru testare. Acest set de date este simplu și mi-a permis să rezolv în mod eficient detecția anomaliilor.

Așa cum am spus mai devreme, anomaliile se întâlnesc atunci când punctele din date au valori extreme foarte îndepărtate de restul valorilor sau când s-a întâmplat un eveniment excepțional. Seriile de timp sunt datele înregistrate cronologic pe o durată de timp fixă atunci și atunci când analizăm aceste date par a fi o tendință sau o sezonalitate. Identificarea anomaliilor în seriile temporale este dificilă, în funcție de tipul de date de intrare, de tipul de anomalie și de natura metodei folosite. [234] Am

normalizat valorile datelor din seria de timp din setul de antrenament. Avem date înregistrate la fiecare 5 minute timp de 14 zile, deci avem 288 de pași pe zi. Am folosit un model personalizat de tip autoencoder de convoluție pentru reconstrucție [2] [288] [289] (model deja existent, focusul nostru fiind funcția de activare).
Modelul are o intrare de forma (dimensiunea lotului, lungimea secvenței, numărul de caracteristici) și returnează o ieșire de aceeași formă. În cazul nostru, lungimea secvenței este 288, iar numărul de caracteristici (avem doar prețul) este egal cu 1.

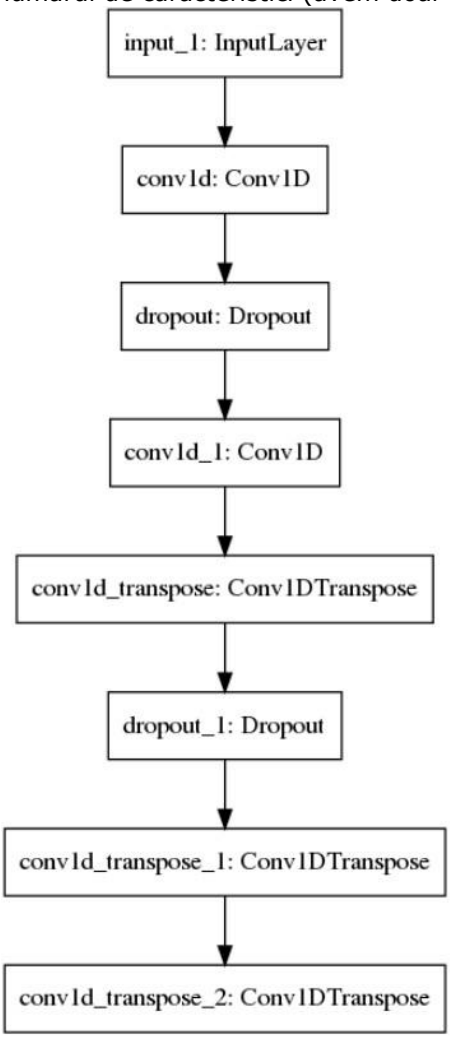
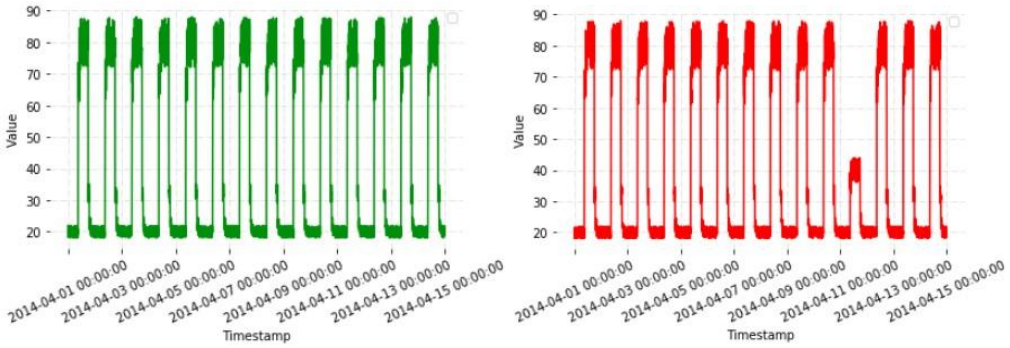


Fig.7.5.4. 1 Straturile modelului autoencoder

În vederea definirii arhitecturii pentru autoencoderul convoluțional de reconstrucție m-am inspirit dintr-o arhitectură existentă [289], acesta este compus din două straturi de convoluție Conv1D (20 și 16 filtre) și strat de dropout (rata de dropout = 0,25) aplicat după fiecare strat Conv1D. Pentru reconstrucție, avem trei straturi Conv1DTranspose (16, 20 și 1 filtre).



a) b)

Fig.7.5.4. 2 a) Serii de timp fără anomalii b) Serii de timp cu anomalii

Pentru detecția anomaliilor din seriile de timp, nu am făcut altceva decât să văd cum modul în care funcționează modelul la reconstrucția datelor de intrare. În acest sens, am încercat să reduc la minimum eroarea dintre datele prezise și datele de intrare pentru antrenament. Am calculat eroarea medie absolută MAE (Mean Absolute Error – eroare medie absolută) [289] pe datele de antrenament.

$$MAE = \frac{\sum_{i=1}^{n} | y_i - x_i |}{n} \tag{7.5.4.1}$$

Unde y_i este predicția (data reconstruită) și x_i corespunde la data originală.

După aceea, am luat valoarea maximă a valorii medii absolute MAE. Această valoare corespunde celui mai rău caz al modului, în care modelul nostru a efectuat reconstrucția unui eșantion. Am considerat această valoare pragul pentru anomalii. Cu alte cuvinte, în cazul unei valori de reconstrucție pentru un eșantion care este mai mare decât această valoare stabilită, atunci putem spune că acest eșantion este considerat o anomalie. Am antrenat 50 de epoci pentru fiecare funcție de activare în parte. Am folosit ca optimizator Adam (rate de învățare = 0,001) și funcția de cost „mse” (Eroare medie pătratică - mean squared error), dimensiunea lotului a fost 128 și am alocat 20% din date pentru setul de validare.

Tabel 7.5.4. 1 Funcția de cost

Funcția de activare	***Funcția de cost pe setul de validare***
ReLU	0.0054
tanh	0.0024
TanhExp	0.0026
TanhExp învățabilă	0.0054
a_TanhExp	**0.0017**
a_TanhExp învățabilă	0.0023

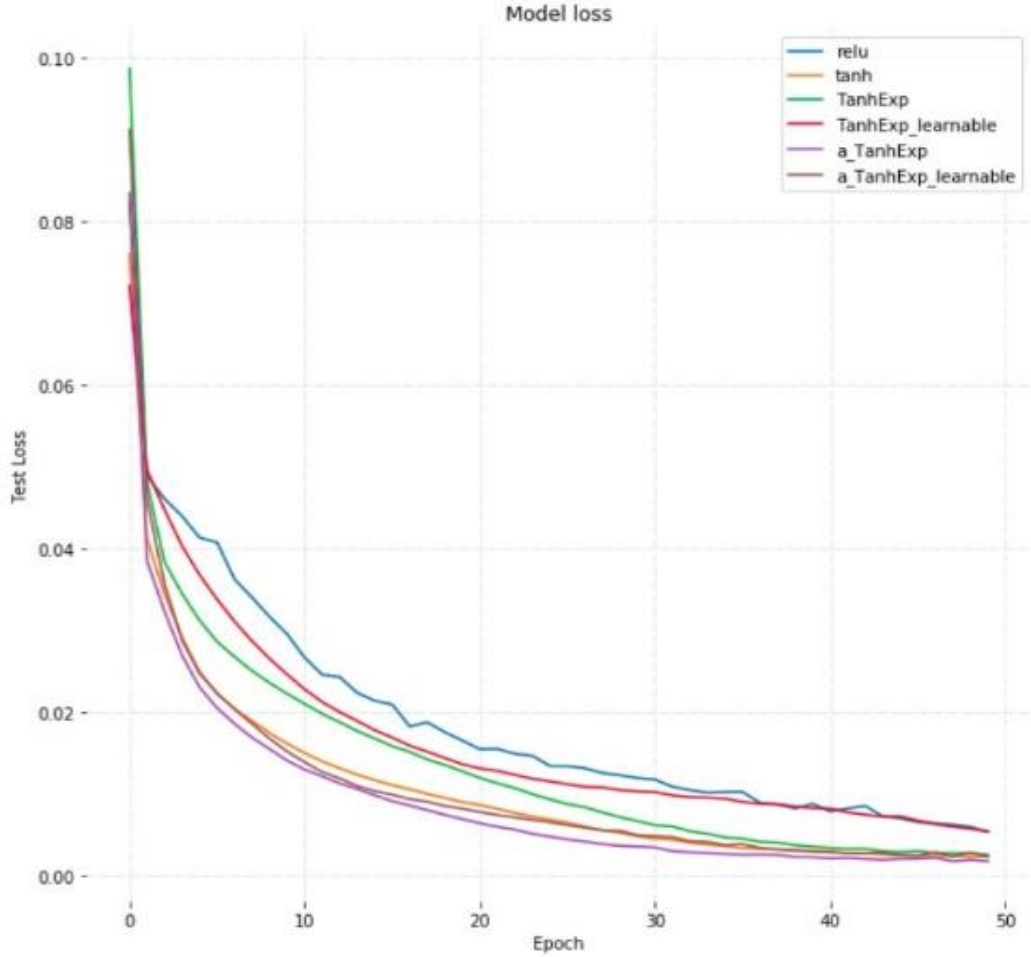


Fig.7.5.4. 3 Funcția de cost pe setul de validare a seriilor de timp

Din figura de mai sus se poate vedea că funcțiile propuse se comportă foarte bine pe acest tip de sarcină și respectiv set de date.
În figura de mai jos putem vedea datele reconstruite din seria de timp pentru a treia secvență de antrenament. Pragul de eroare de reconstrucție este de 0,0665.

Fig.7.5.4. 4 Seria de timp originală vs seria de timp prezisă

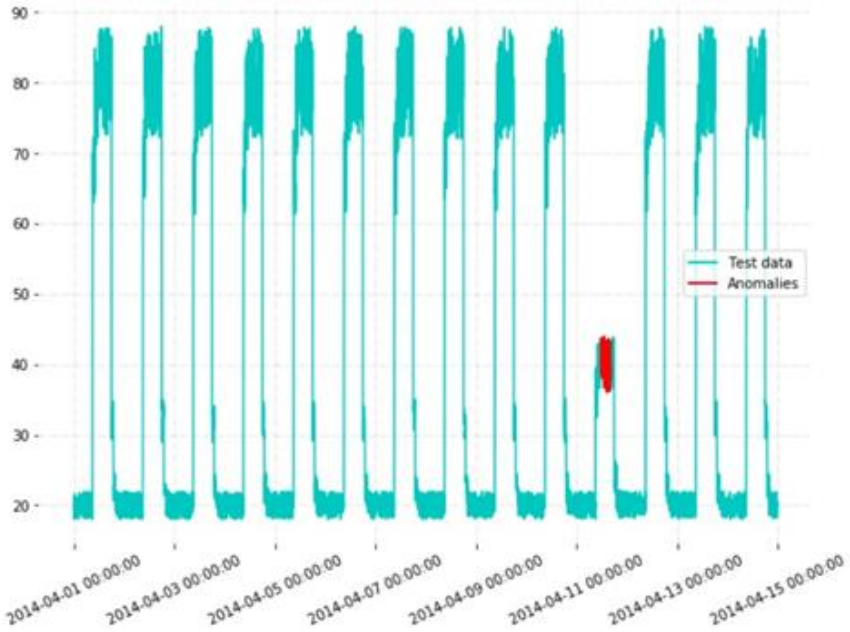


Fig.7.5.4. 5 Anomaliile suprapuse peste datele de test originale

7.6. Dezvoltarea de noi funcții de activare în detecția anomaliilor în serii de timp cu autoencoder LSTM – Partea experimentală

În cadrul acestui subcapitol, compar propunerile mele prezentate în subcapitolul 6.7 cu funcțiile de activare tanh, ReLU și TaLu. Voi folosi o arhitectură personalizată de tip autoencoder LSTM [290].
Am antrenat pe setul de date index S&P 500. Acest set de date conține prețurile de închidere zilnice pentru indicele S&P 500 în perioada 2 ianuarie 1986 - 29 iunie 2018, pentru o perioadă totală de 32 de ani. S&P 500 [291], „este un indice bursier care măsoară performanța bursieră a 500 de companii mari listate la bursele din Statele Unite (SUA)". Este unul dintre cei mai utilizați indici de capitaluri proprii și este considerat a fi una dintre cele mai bune reprezentări ale pieței bursiere din SUA. Acest set de date este disponibil în format CSV cu o coloană care conține timpul zilnic și a doua coloană corespunzătoare prețurilor de închidere pentru fiecare zi. Acest set de date financiare oferă la nivel granular mai multe date de tip serii temporale. Acest set de date este univariat din cauza faptului că are o singură caracteristică (variabila preț). Datele totale sunt 8192 înregistrări. În experimente, am folosit 70% din datele totale, ceea ce înseamnă 5734 pentru antrenare și 2458 (30%) pentru faza de testare.

date	close
1986-01-02	209.59
1986-01-03	210.88
1986-01-06	210.65
1986-01-07	213.80
1986-01-08	207.97
1986-01-09	206.11
1986-01-10	205.96
1986-01-13	206.72
1986-01-14	206.64
1986-01-15	208.26

date	close
2018-06-18	2773.75
2018-06-19	2762.59
2018-06-20	2767.32
2018-06-21	2749.76
2018-06-22	2754.88
2018-06-25	2717.07
2018-06-26	2723.06
2018-06-27	2699.63
2018-06-28	2716.31
2018-06-29	2718.37

Fig.7.6. 1 Primele 10 valori și ultimele 10 valori pe setul de date S&P 500 Index

În figura de mai jos am descris câteva detalii statistice despre S&P 500 Index, valoarea totală a prețului închis, abaterea standard, media, valoarea minimă și valorea maximă. Am scalat datele folosind datele de antrenare și am aplicat aceleași transformări și pe datele de test. Am creat secvențe lunare (30 de zile) de date istorice.

```
count    8192.000000
mean     1070.897411
std       616.714534
min       203.490000
25%       458.632500
50%      1106.435000
75%      1388.132500
max      2872.870000
Name: close, dtype: float64
```

Fig.7.6. 2 Setul de date S&P 500 Index

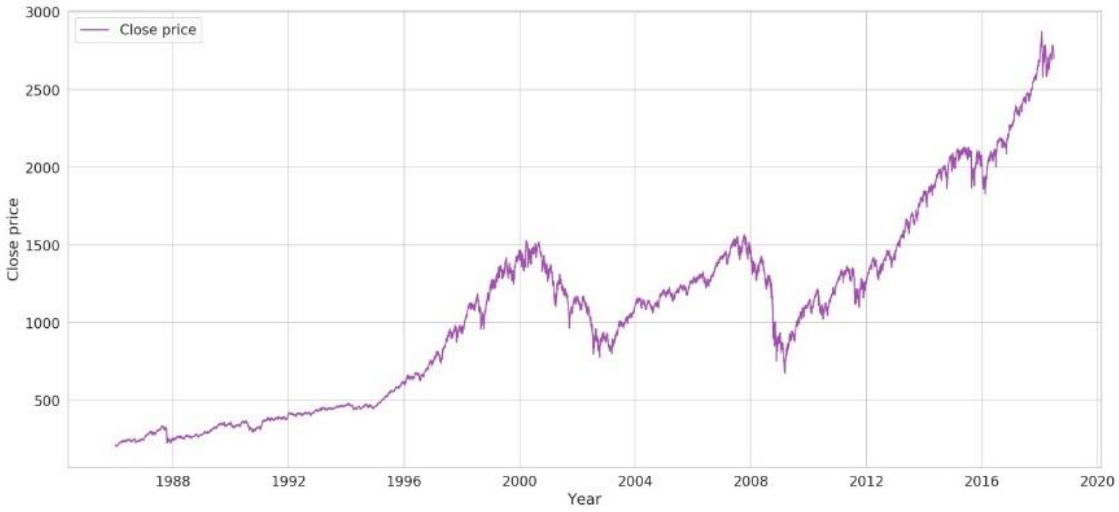


Fig.7.6. 3 Setul de date S&P 500 Index în timp

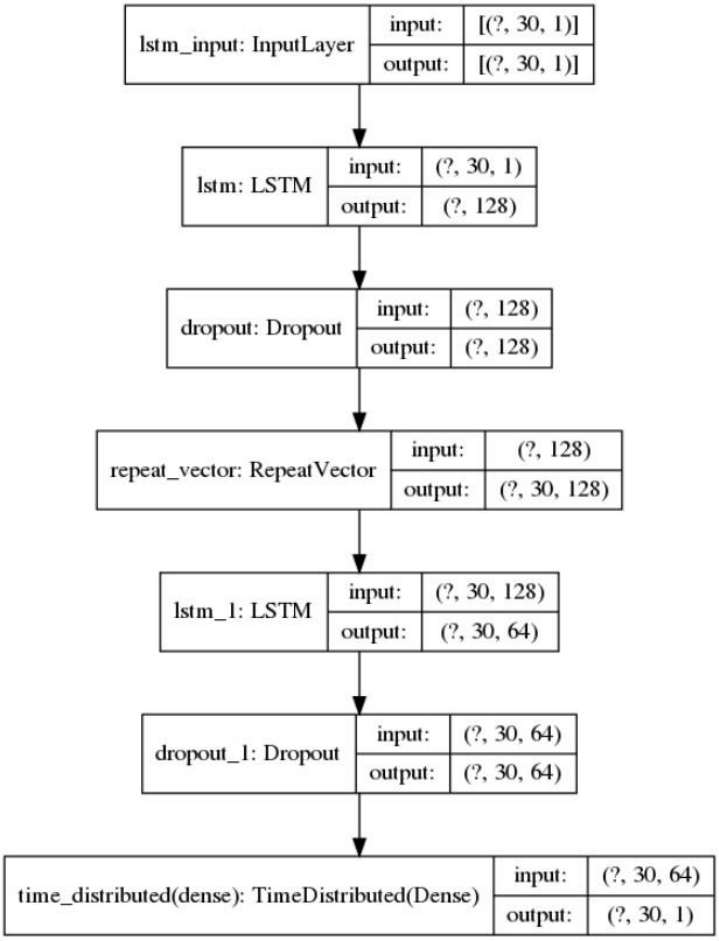


Fig.7.6. 4 Arhitectura autoencoder LSTM

Așa cum se vede în Fig.7.6.4 pentru definirea arhitecturii am pornit de la o arhitectură LSTM Autoencoder existentă [292] și are un strat de intrare, urmat de un strat LSTM cu 128 de unități după care am aplicat un strat de dropout (rată de dropout egală cu 0,25). După aceea, am aplicat un strat de RepeatVector, apoi un alt strat LSTM cu 64 de unități și alt strat de dropout (rata egală cu 0,25). La final, am avut un strat TimeDistributed și un strat dens (are 1 unitate). Deoarece focusul meu în această teză este să identific direcții în zona funcțiilor de activare, care să ne conducă spre o performanță mai bună, în acest studiu performanța este dată de o funcție de cost minimă, am ales arhitecturi existente pe care le-am adaptat în funcție de cerințele sarcinii.
Am folosit ca funcție de cost „mae” (eroare absolută medie) și ca optimizator „Adam”. Stratul RepeatVector repetă pur și simplu intrarea de *n* ori, este responsabil pentru adăugarea tututor citirilor într-un singur vector pe care îl folosesc la reconstrucția secvenței originale. TimeDistributed creează, de asemenea, un vector cu o dimensiune egală cu numărul de ieșiri din stratul anterior.

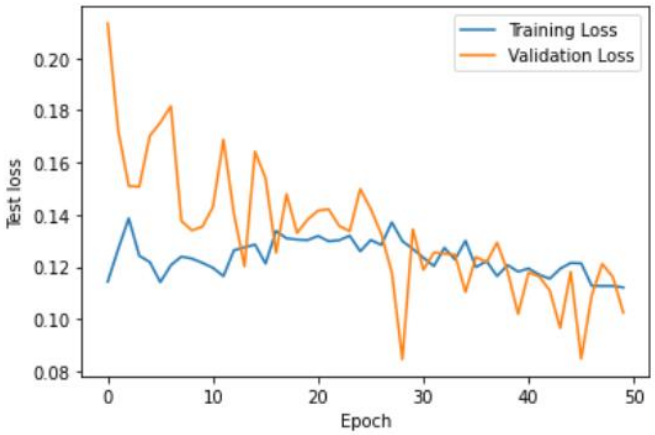


Fig.7.6. 5 Evaluare modelului

Evaluarea autoencoder-ului LSTM poate fi văzută în Fig.7.6.5, curba funcției de cost pe setul de test (*validation loss*) urmează curba funcției de cost pe setul de antrenament (*training loss*) fiind destul de aproape. Are câteva vârfuri de creștere a erorilor în detecție, dar, în general, se comportă bine.

Tabel 7.6. 1 Funcția de cost pe setul de validare

Funcția de activare	***Funcția de cost pe setul de validare 50 epoci 64 dimensiunea lotului***	***Funcția de cost pe setul de validare 50 epoci 32 dimensiunea lotului***
ReLU	0.0846	0.0907
tanh	0.0245	0.0298
TaLu	0.0270	0.0972
Talu learnable	0.0243	0.0269
P_Talu	**0.0224**	**0.0247**

Prin introducerea unor noi funcții de activare am reușit să reduc eroarea funcției de cost pe setul de validare ca în Tabelul 7.6.1, obținând cea mai mică funcție de cost pe setul de validare cu funcția P_Talu învățabilă. Dar, nu numai P_Talu învățabilă a obținut un rezultat bun, ci și Talu învățabilă care a făcut ca modelul să obțină o funcție de cost mică, o valoare mai mică decât funcțiile de activare ReLU, tanh și TaLu. Aceste experimente ne arată potențialul lor în astfel de sarcini. Am antrenat pentru 50 de epoci, am stabilit o dimensiune a lotului egală cu 64, de asemenea am făcut o împărțire a setului de date alocând 20% pentru validare, nu am amestecat datele și am folosit metoda de salvare a celui mai bun model, în care a fost înregistrată funcția de cost minimă pe setul de validare. Am modificat acești hiperparametri pentru a vedea cum se comportă funcțiile propuse în noile condiții de testare, considerând și cazul în care dimensiunea lotului este 32. În Fig. 7.6.6., putem vedea curba de eroare pentru fiecare funcție de activare în parte, putem vedea că tanh, Talu care poate fi învățat și P_Talu care poate fi învățat sunt curbele care sunt cele mai apropiate de pierderea minimă de validare în comparație cu ReLU și TaLu. Tanh fiind o funcție recomandată în arhitectura LSTM, motivul pentru care obține rezultate atât de bune, dar propunerile mele oferă, de asemenea, funcții de cost pe

setul de validare competitive cu tanh. Acest comportament întărește potențialul funcțiilor propuse în cadrul acestei teze.

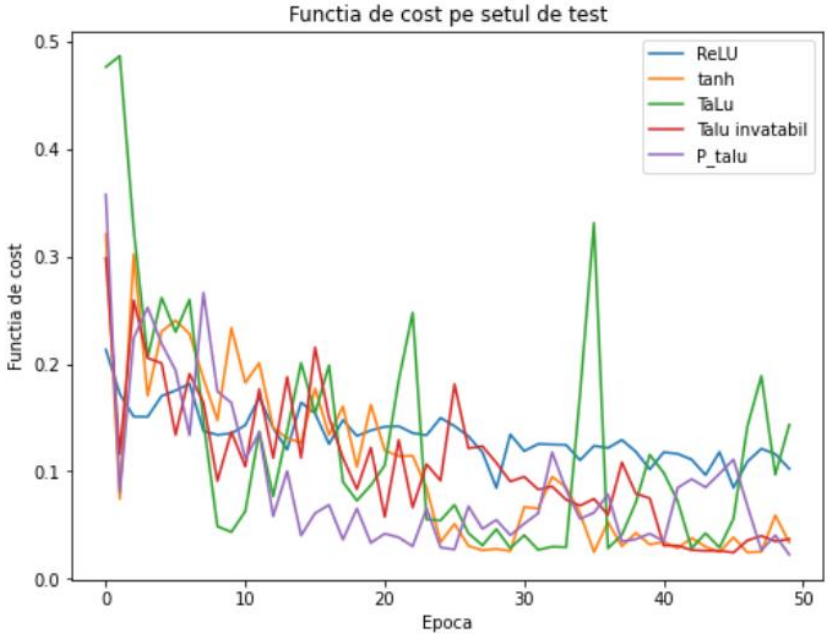


Fig.7.6. 6 **a)** Curba de eroare pentru funcțiile de activare - dimensiunea lotului 64

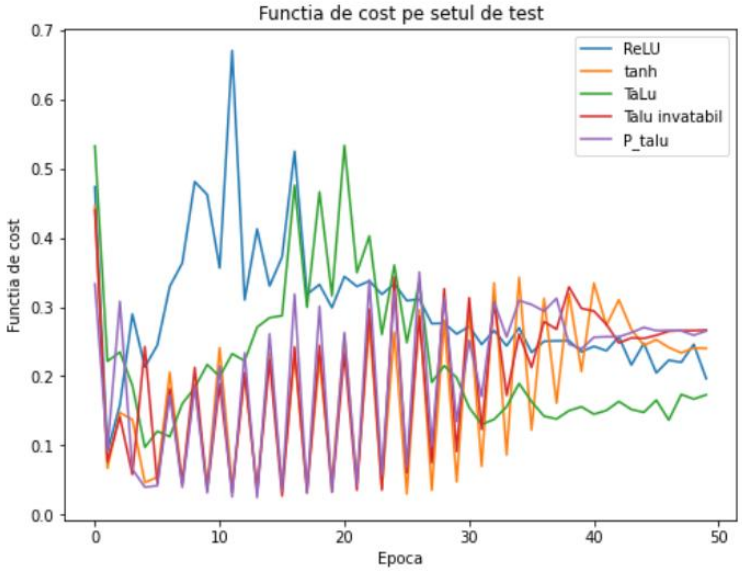


Fig.7.6. 7 **b)** Curba de eroare pentru funcțiile de activare - dimensiunea lotului 32

Pentru detecția anomaliilor din datele din seria de timp, am văzut cum funcționează modelul în reconstrucția datelor de intrare. Pornind de la această ipoteză, am încercat să reduc la minimum eroarea dintre datele prezise și datele de intrare ale antrenamentului. Am calculat metrica MAE (eroare absolută medie) pe eșantioane de antrenament (Fig.7.6.7).

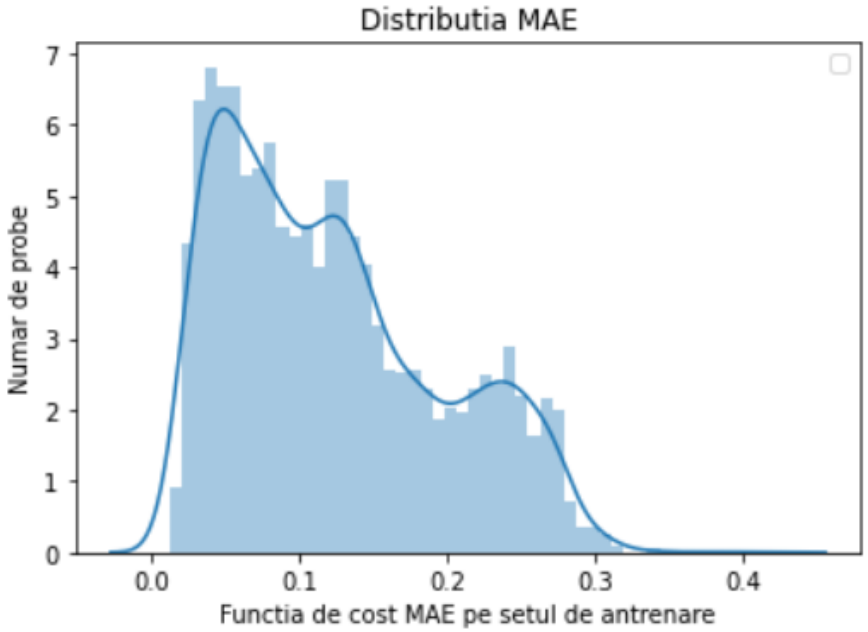


Fig.7.6. 8 Distribuția MAE pe setul de antrenament

Având pragul (valoarea maximă din distribuția MAE), în acest caz acest prag este 0.41 pentru dimensiunea lotului egală cu 64, putem stabili anomaliile din date, toate valorile de deasupra pragului reprezintă o anomalie.

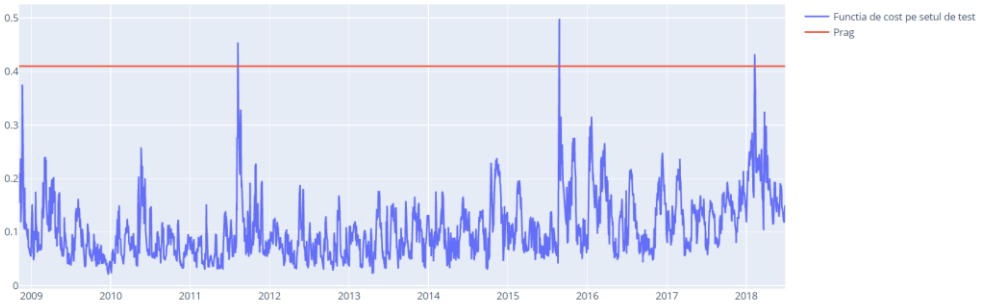


Fig.7.6. 9 Limita pragului pentru detecția anomaliilor prag = 0.41

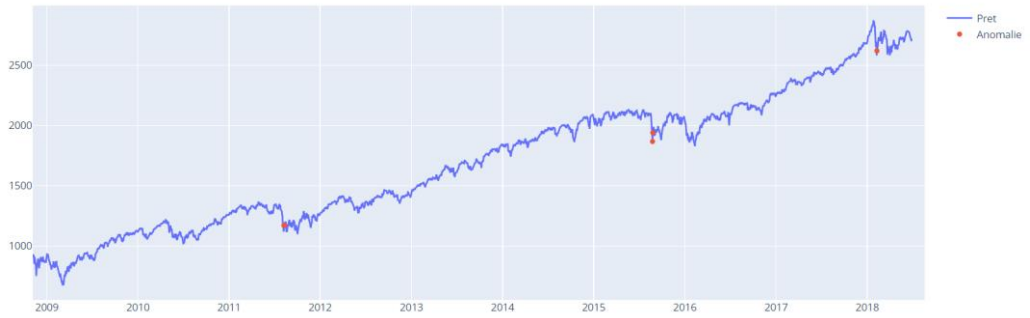


Fig.7.6. 10 Detecția anomaliilor prag = 0.41

Tabel 7.6. 2 Anomalii în prețurile de închidere zilnice în setul de date S&P 500

Data	Funcția de cost test	Prag	Anomalie	Preț
2011-08-09	0.4544	0.41	Adevărat	0.8557
2011-08-11	0.4164	0.41	Adevărat	0.8560
2015-08-25	0.4544	0.41	Adevărat	2.4747
2015-08-26	0.4988	0.41	Adevărat	2.6445
2018-02-09	0.4323	0.41	Adevărat	4.2261

dimensiunea lotului = 64
prag = 0.41

În Fig. 7.6.9 anomaliile sunt marcate prin puncte roșii și putem vedea că apar mai degrabă pe punctele cu modificări abrupte ale prețului de închidere. În tabelul 7.6.2 putem vedea data la care apar anomaliile și valorile lor testând cazul cu dimensiunea lotului de 64 și durata de antrenament egală cu 50 de epoci.

În cazul testării cazului în care avem dimensiunea lotului egală cu 32, valoarea pragului devine 0.62.

Tabel 7.6. 3 Anomalii în prețurile de închidere zilnice în setul de date S&P 500 – primele 5 înregistrări

Data	Funcția de cost test	Prag	Anomalie	Preț
2013-05-30	0.6345	0.62	Adevărat	1.9781
2013-05-31	0.6338	0.62	Adevărat	1.9229
2013-06-03	0.6683	0.62	Adevărat	1.9455
2013-06-04	0.6591	0.62	Adevărat	1.9244
2013-06-05	0.6751	0.62	Adevărat	1.8721

dimensiunea lotului = 32
prag = 0.62

Tabel 7.6. 4 Anomalii în prețurile de închidere zilnice în setul de date S&P 500 – ultimele 5 înregistrări

Data	Funcția de cost test	Prag	Anomalie	Preț
2018-06-25	2.6557	0.62	Adevărat	4.4532
2018-06-26	2.6714	0.62	Adevărat	4.4672
2018-06-27	2.6652	0.62	Adevărat	4.4126
2018-06-28	2.6736	0.62	Adevărat	4.4514
2018-06-29	2.6636	0.62	Adevărat	4.4562

dimensiunea lotului = 32
prag = 0.62

Fig.7.6. 10 Limita pragului pentru detecția anomaliilor prag = 0.62

Fig.7.6. 11 Detecția anomaliilor prag = 0.62

În cazul în care dimensiunea lotului este egală cu 32, am obținut un prag cu o valoare mai mare decât în cazul lotului de 64. Dacă în cazul lotului de 64 cu valoarea pragului de 0.41 am identificat 5 puncte care au fost anomalii, în cazul pragului de 0.62 am identificat un total de 1282 de puncte care sunt considerate anomalii. Primele 5 valori corespondente anomaliilor, respectiv ultimele 5 valori de anomalie pot fi văzute în tabelele Tabel 7.6. 3 și Tabel 7.6. 4.

7.7. Soft Clipping Mish - o nouă funcție de activare - Partea experimentală

În vederea validării propunerilor mele, am ales arhitectura LeNet-5, după fiecare strat de convoluție am aplicat un strat de normalizare a lotului. Am folosit 5 seturi de date și anume: MNIST, Fashion-MNIST, CIFAR-10, CIFAR-100 pentru o sarcină de clasificare și Beijing PM2.5 [293] pentru o sarcină de predicție a poluării aerului. Primele 4 seturi le-am mai folosit și în cadrul altor studii în capitolele anterioare. În toate experimentele, am inițializat **Soft Clipping Mish** (**SC Mish**) învățabilă (**Soft Clipping Mish Learnable – SCL Mish**) cu 0.25, urmând ca parametrul α să fie învățat în timpul antrenamentului.

Setul de date Beijing PM2.5. conține datele PM2.5 ale Ambasadei SUA la Beijing. Ulterior au fost incluse și datele meteorologice de la Aeroportul Internațional din capitala Beijing.

7.7.1. Experimentul 1: Setul de date MNIST

În cadrul acestui experiment am folosit setul de date MNIST pentru a vedea cum se comportă cele două funcții propuse pe o sarcină de clasificare. Ca arhitectură, am folosit LeNet-5, la care am adăugat straturi de normalizare a lotului după fiecare strat de convoluție. În partea de preprocesare am normalizat datele. În cadrul modelului pe ultimul strat am avut 10 neuroni, egal cu numărul de clase (fiind cifre scrise de mână de la 0 la 9), tot pe acest strat am avut ca funcție de activare Softmax. Alte detalii sunt:

- funcția de cost pe care am folosit-o a fost 'categorical_crossentropy',
- optimizator am folosit Adam cu o rată de învățare lr=0.001,
- 50 de epoci,
- dimensiunea lotului a fost 128.

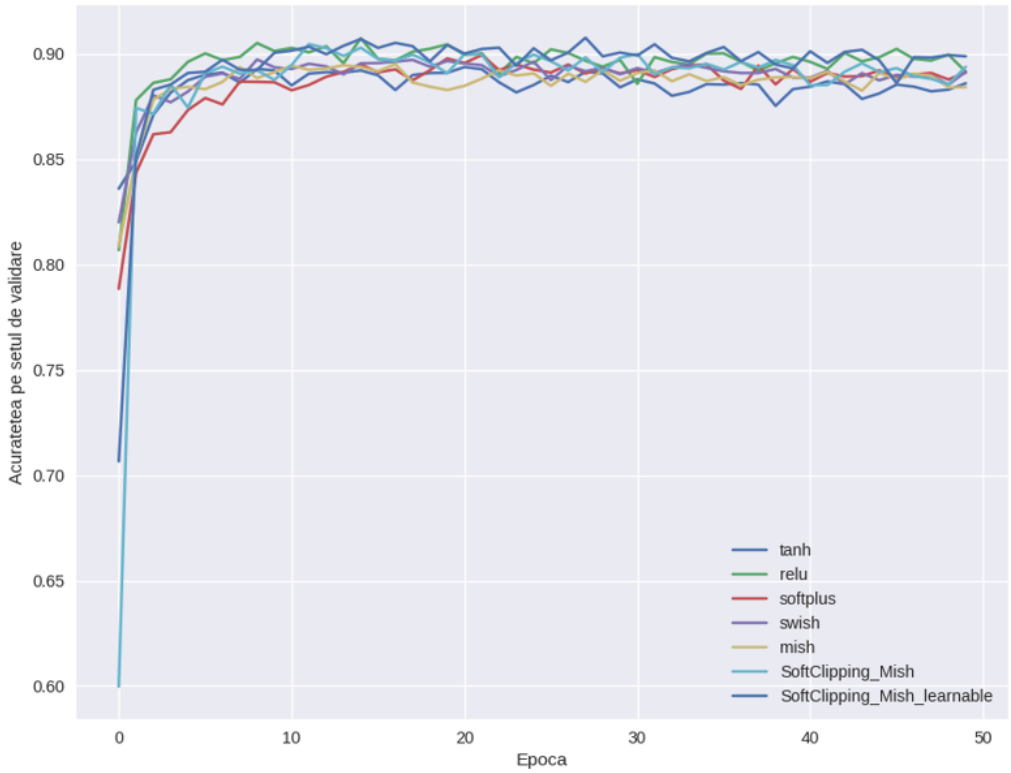


Fig.7.7.1. 1 Acuratețea pe setul de validare pentru setul de date MNIST

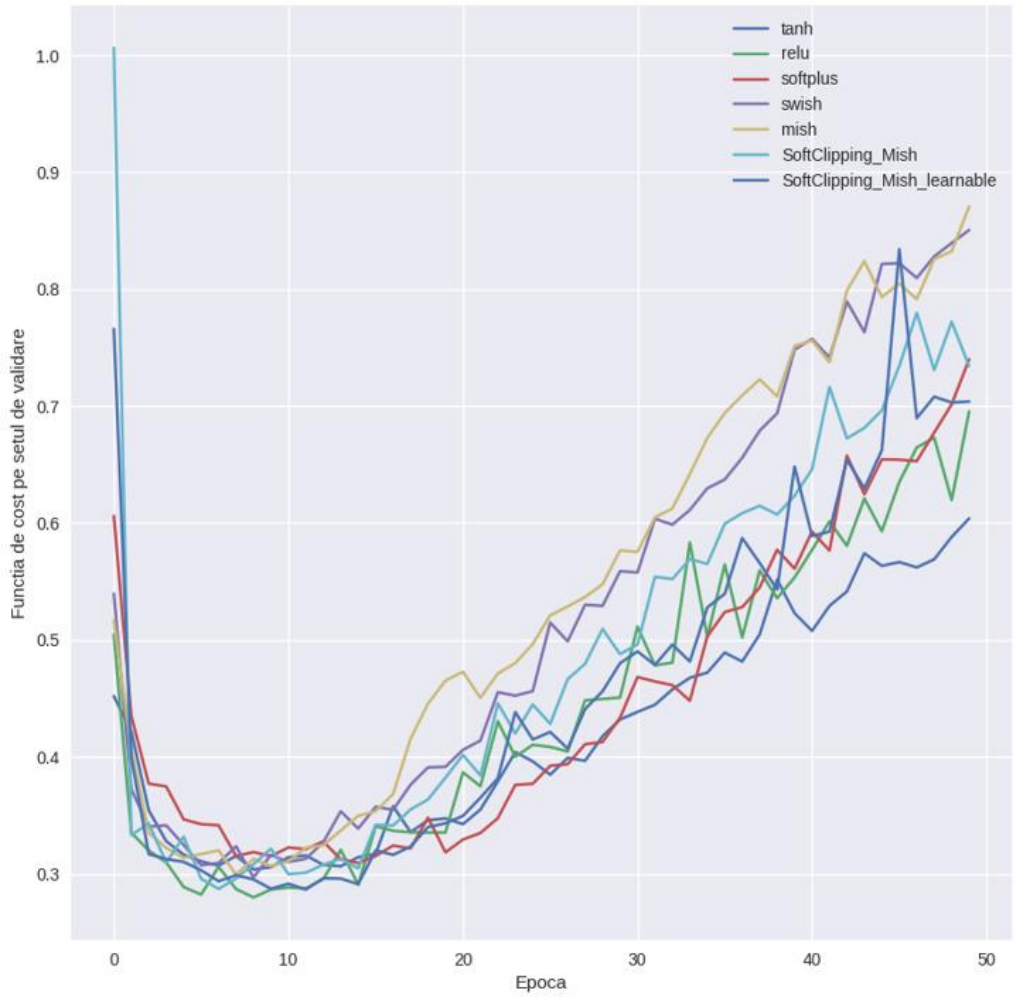


Fig.7.7.1. 2 Funcția de cost pe setul de validare pentru setul de date MNIST

Din figura Fig.7.7.1.1 se poate vedea ca funcția propusă ***Soft Clipping Mish*** learnable (învățabilă) încă din primele 10 epoci depășește celalalte funcții de activare: sigmoida, tanh, ReLU, Softplus, Swish, Mish. Iar din Fig.7.7.1.2 se poate vedea din punct de vedere a funcției de cost funcția ***Soft Clipping Mish learnable*** are cea mai mică funcție de cost începând cu epoca 16, dând cea mai mică funcție până în epoca 50. Ambele grafice ne arată că atât Soft Clipping Mish cu parametru predefinit cât și cu parametru învățabil dau rezultate competitive cu restul funcțiilor.

Tabel 7.7.1. 1Rezultate experimentale pe setul de date MNIST

Arhitectura /Funcția de activare	Sigmoida	tanh	ReLU	Softplus	Swish	Mish	**SC Mish**	**SCL Mish**
LeNet-5	99.14%	99.10%	99.21%	99.03%	99.25%	99.08%	**99.27%**	**99.27%**

50 epoci

7.7.2. Experimentul 2: Setul de date Fashion-MNIST

În cadrul acestui experiment am folosit setul de date Fashion-MNIST pentru a vedea cum se comportă cele două funcții propuse. Ca arhitectură, am folosit ca în cazul setului de date MNIST, păstrând aceleași condiții privind arhitectura și informații suplimentare privind numărul de epoci, dimensiunea lotului, optimizator, funcție de cost și așa mai departe. Am antrenat 50 epoci, salvând cel mai bun model pentru fiecare funcție în parte. Am antrenat pe 60000 de imagini și validat pe 10000.

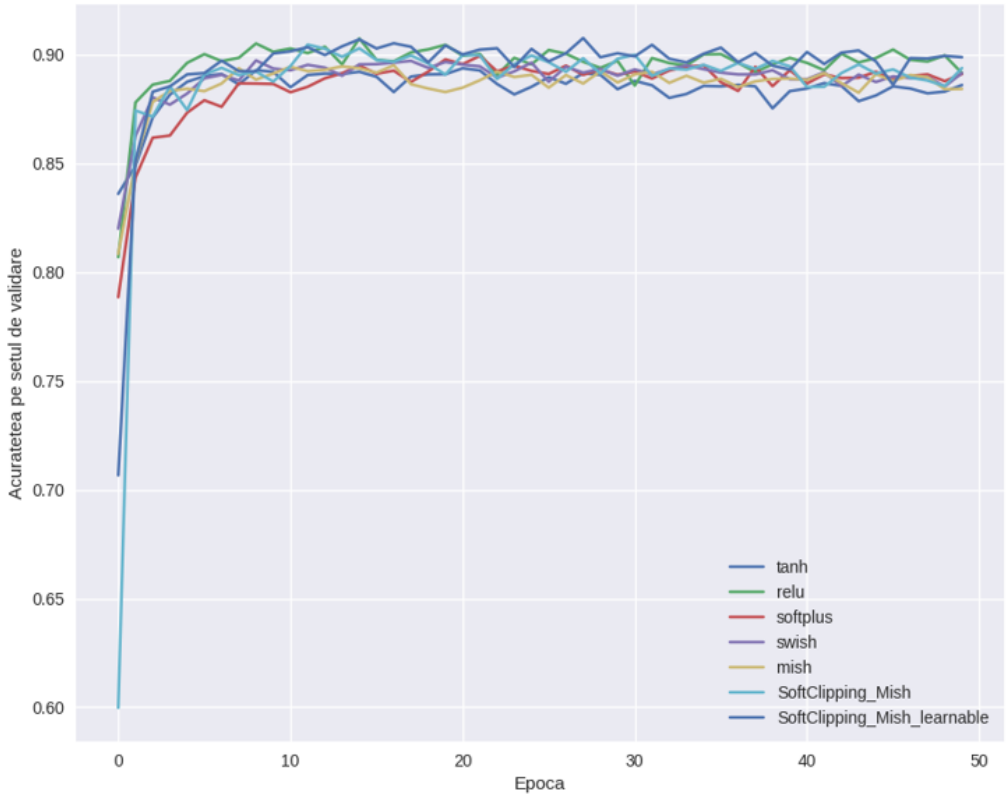


Fig.7.7.2. 1 Acuratețea pe setul de validare Fashion-MNIST

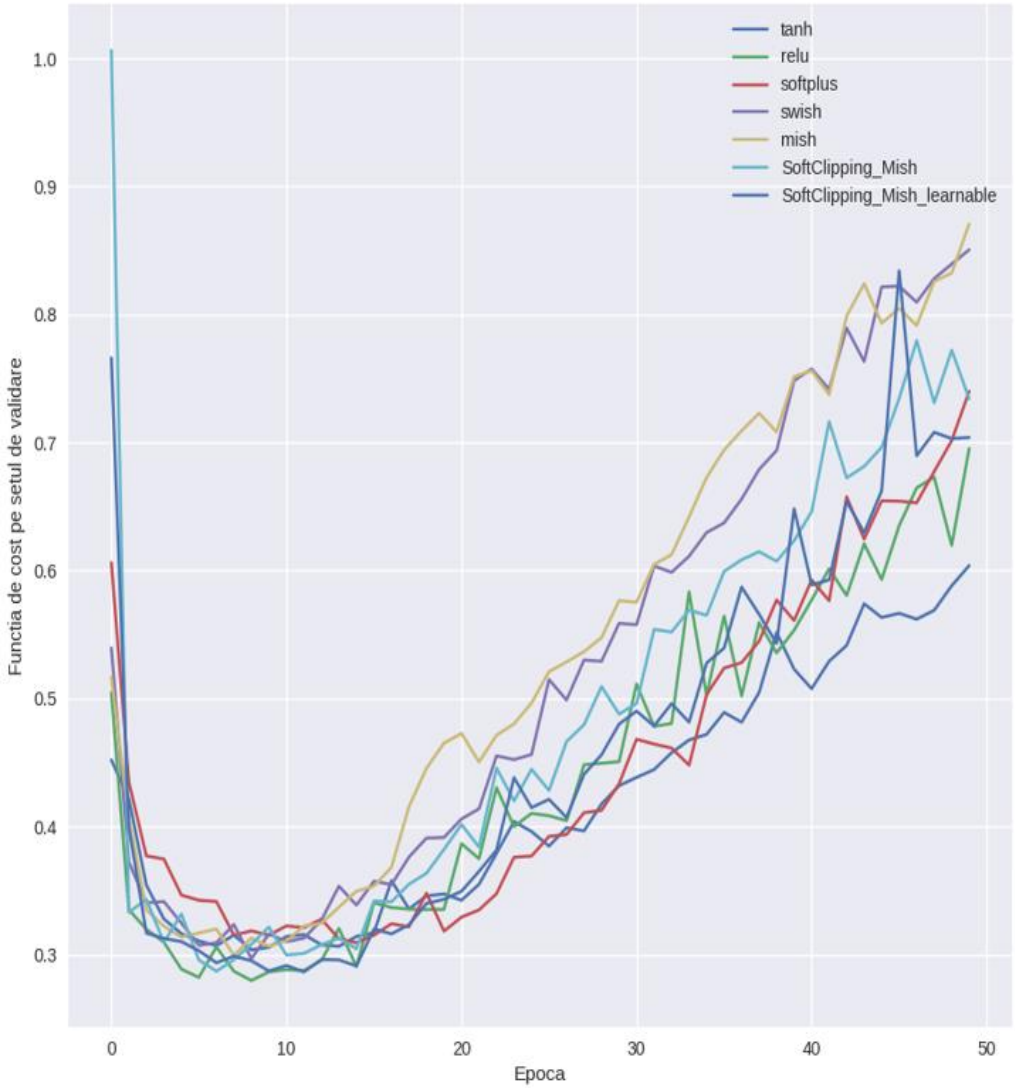


Fig.7.7.2. 2 Funcția de cost pe setul de validare Fashion-MNIST

Tabel 7.7.2. 1 Rezultate experimentale pe setul de date Fashion-MNIST

Arhitectura /Funcția de activare	Sigmoida	tanh	ReLU	Softplus	Swish	Mish	**SC Mish**	**SCL Mish**
LeNet-5	89.59%	89.37%	90.74%	89.94%	89.73%	89.51%	90.46%	**90.76%**

50 epoci

7.7.3. Experimentul 3: Setul de date CIFAR-10

În cadrul acestui experiment am folosit setul de date CIFAR-10 pentru a extinde studiul impactului a celor două funcții propuse. Ca arhitectură, am folosit ca în cazul experimentelor anterioare, LeNet-5, păstrând aceleași condiții privind arhitectura și informații suplimentare privind numărul de epoci, dimensiunea lotului, optimizator, funcție de cost și așa mai departe. Am antrenat 50 epoci, dimensiunea lotului de 128 (se recomandă ca această dimensiune să fie 32 sau 64 pentru creșterea performanței, dar pentru a antrena mai rapid am folosit 128), salvând cel mai bun model pentru fiecare funcție în parte. Am antrenat pe 50000 de imagini și validat pe 10000.

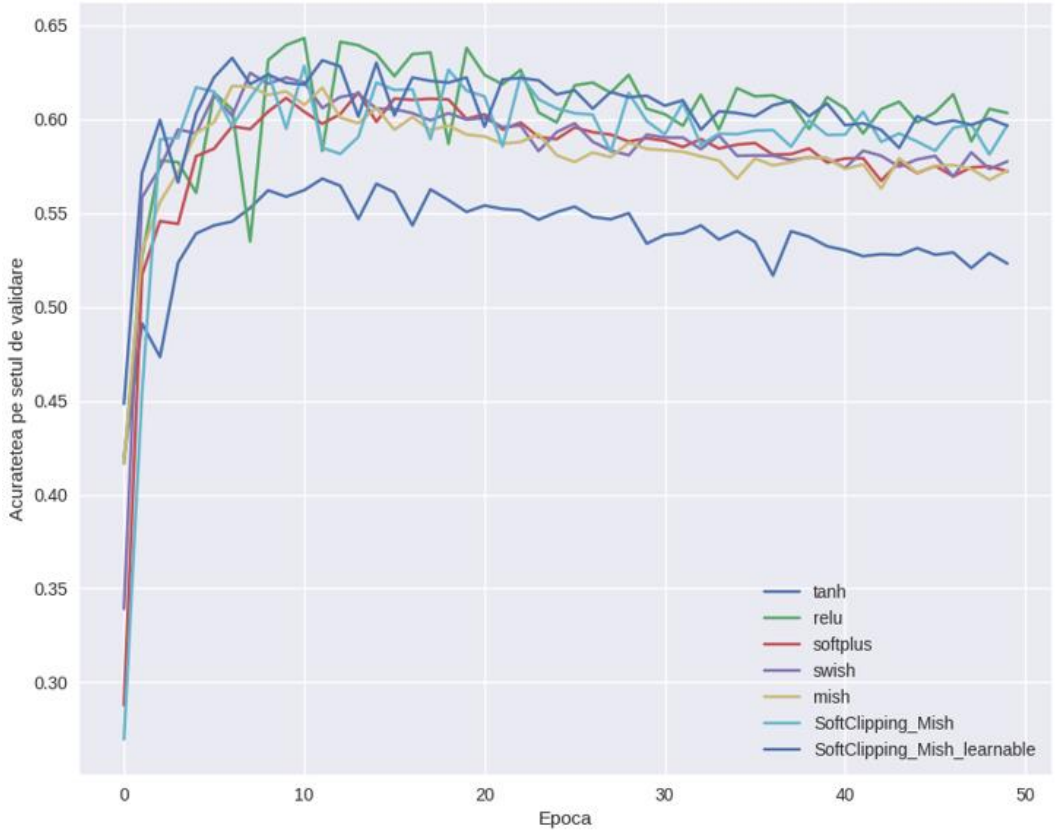


Fig.7.7.3. 1 Acuratețea pe setul de validare pentru setul de date CIFAR-10

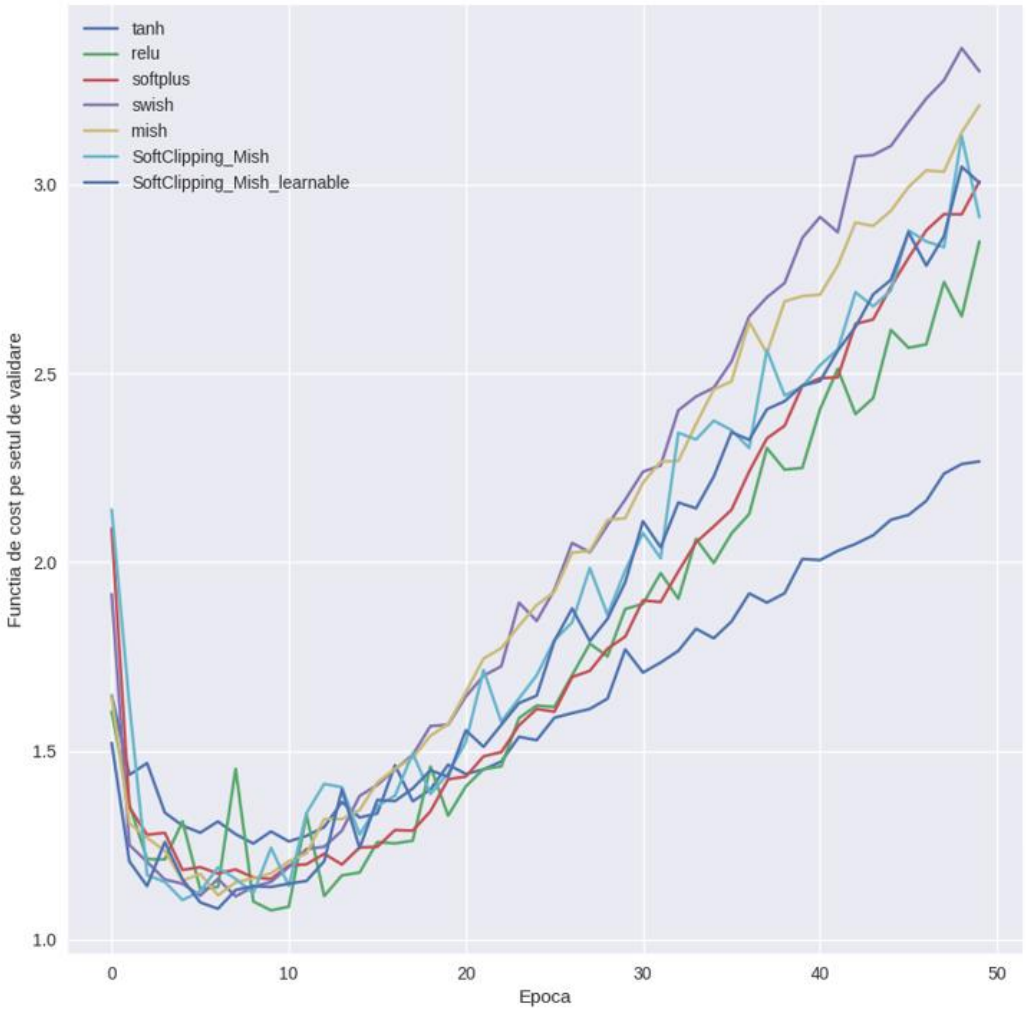


Fig.7.7.3. 2 Funcția de cost pe setul de validare pentru setul de date CIFAR-10

Tabel 7.7.3. 1 Rezultate experimentale pe setul de date CIFAR-10

Arhitectura /Funcția de activare	Sigmoida	tanh	ReLU	Softplus	Swish	Mish	**SC Mish**	**SCL Mish**
LeNet-5	58.12%	56.83%	**64.32%**	61.43%	62.47%	61.77%	62.84%	63.26%

50 epoci

În tabelul 7.7.3.1 s-a putut vedea că pe CIFAR-10, cea mai bună valoare pentru acurateţe pe setul de validare a fost obținută de funcția ReLU, dar în top 3

funcții de activare se află și cele două funcții propuse, lucru care îmi întărește convingerea că sunt niște propuneri valide, cu atât mai mult cu cât au obținut precizii mai mari ca funcțiile Swish și Mish, care sunt cunoscute în literatura de specialitate ca două funcții care oferă rezultate competitive.

7.7.4. Experimentul 4: Setul de date CIFAR-100

În cadrul acestui experiment am folosit setul de date CIFAR-100 pentru a testa cele două funcții propuse. Ca arhitectură, am folosit ca în cazul experimentelor anterioare, LeNet-5, păstrând aceleași condiții privind arhitectura și ca informații suplimentare am antrenat 100 epoci, dimensiunea lotului 128 și 100 de clase. Am salvat cel mai bun model pentru fiecare funcție în parte. Am antrenat pe 50000 de imagini și validat pe 10000.

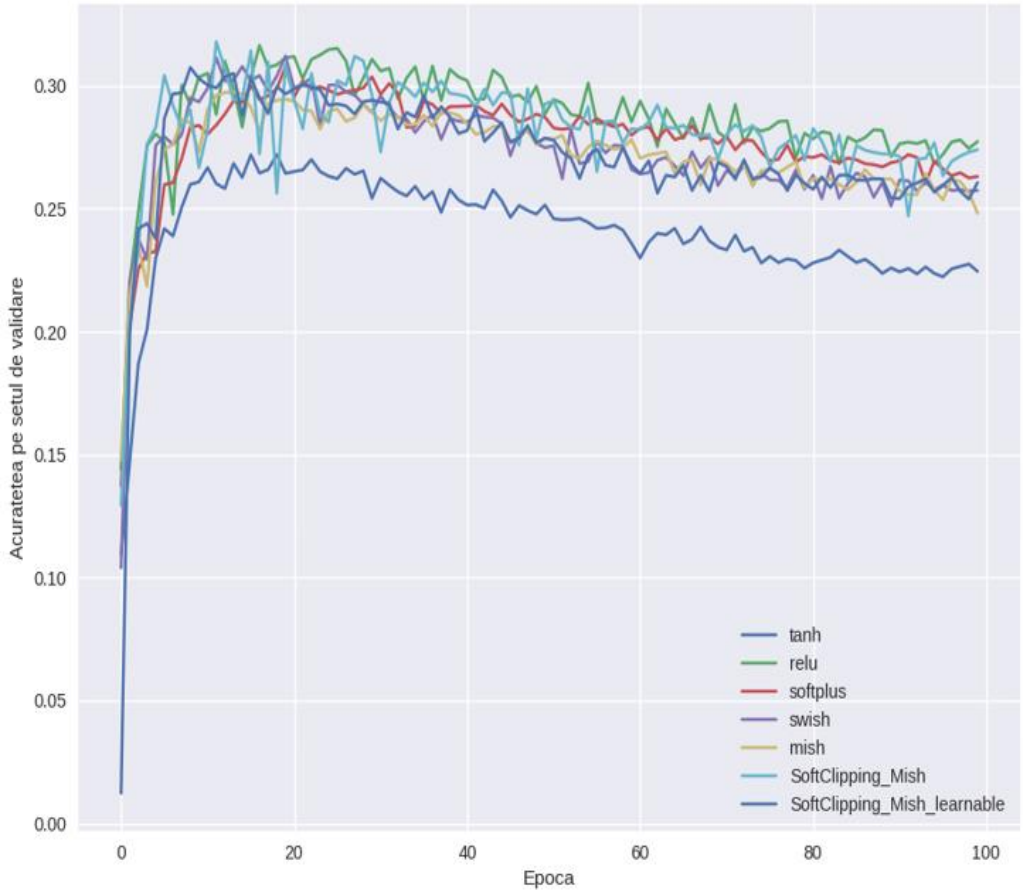


Fig.7.7.4. 1 Acuratețea pe setul de validare pentru setul de date CIFAR-100

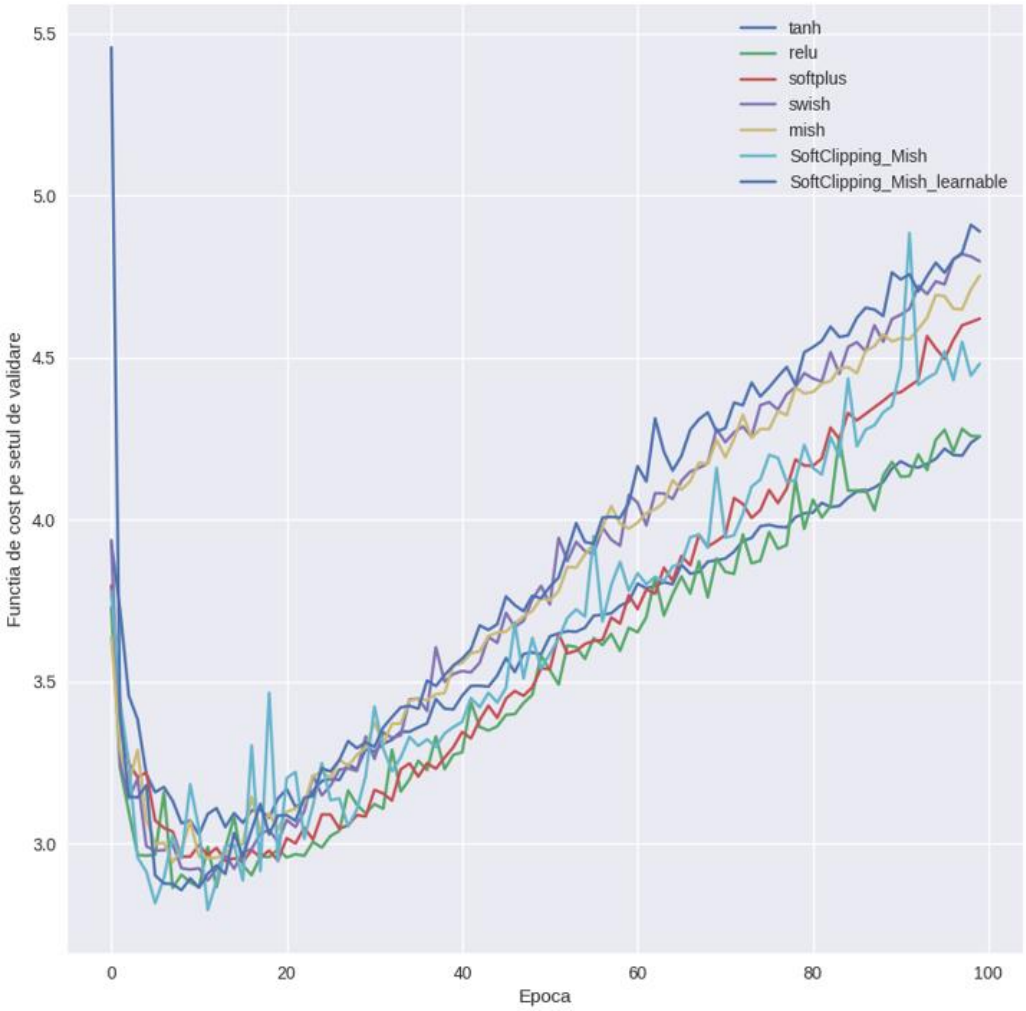


Fig.7.7.4. 2 Acuratețea pe setul de validare pentru setul de date CIFAR-100

Tabel 7.7.4. 1 Rezultate experimentale pe setul de date CIFAR-100

Arhitectura /Funcția de activare	Sigmoida	tanh	ReLU	Softplus	Swish	Mish	**SC Mish**	**SCL Mish**
LeNet-5	26.63%	27.20%	31.64%	30.78%	31.21%	29.75%	**31.79%**	30.74%

100 epoci

7.7.5. Experimentul 5: Predicția poluării aerului folosind LSTM si GRU

În cadrul acestui experiment am folosit un set de date multivariate pentru serii de timp numit Beijing PM2.5. [293], având un total de 43824 de instanțe. Am folosit două tipuri de arhitecturi: LSTM (Long Short-Term Memory) și GRU (Gated Recurrent Unit). Avem date zilnice din oră în oră pe o perioadă de 5 ani (02.01.2010 - 31.12.2014). Ca și intrări avem: temperatura (*temp*), presiunea (*press*), viteza vântului (*wnd_spd*), zăpadă (*snow*), ploaie (*rain*) dar și roua (*dew*) care este un bun indicator al calității aerului aproape de suprafață. Pe baza acestor caracteristici scopul meu este să prezic calitatea aerului dată de gradul de poluare (*pollution*).

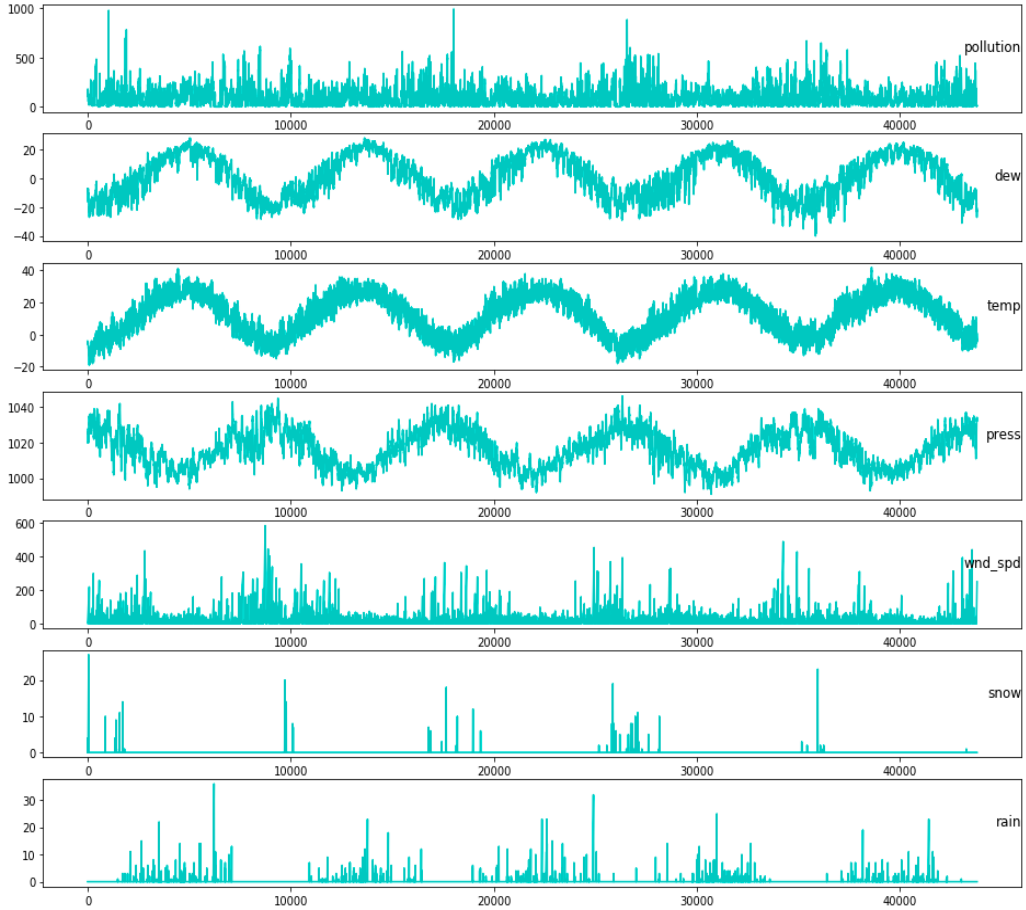


Fig.7.7.5. 1 Setul de date Beijing PM2.5

a) Arhitecturile LSTM și GRU fără straturi ascunse

Pe stratul LSTM am avut 32 de neuroni după care am aplicat o rată dropout de 0.20. Funcția de cost pe care am folosit-o a fost *'mae'* (*mean absolute error*) și pe ultimul strat dens avem o sigmoidă cu 1 neuron. Am antrenat pe 13140 (3 ani) de

eșantioane și am validat pe 30659 de eșantioane timp de 100 de epoci și dimensiunea lotului a fost 128.

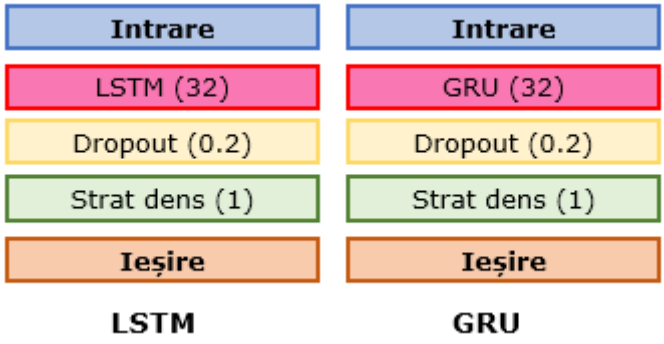


Fig.7.7.5. 2 Arhitectura fără straturi ascunse a) LSTM și b) GRU

În cazul arhitecturii GRU am procedat în mod similar ca în cazul arhitecturii LSTM, înlocuind doar stratul LSTM cu un strat GRU. După modelare și antrenare, am analizat funcția de cost (ideal să fie cât mai mică) și metrica RMSE.

Tabel 7.7.5. 1 Rezultate experimentale pe arhitecturile LSTM și GRU fără straturi ascunse pentru setul de date Beijing PM2.5

	Arhitectura LSTM	**Arhitectura GRU**	**Arhitectura LSTM**	**Arhitectura GRU**
Funcția de activare	***Funcția de cost***		***RMSE***	
Sigmoida	0.0172	0.0163	15.094	15.319
tanh	0.0172	0.0171	17.847	16.834
ReLU	0.0152	0.0163	17.658	18.003
Softplus	0.0159	0.0160	13.562	14.595
Swish	0.0158	0.0164	11.187	13.71
Mish	0.0156	0.0164	**10.681**	13.825
SC Mish	**0.0146**	0.0145	**10.815**	9.304
SCL Mish	**0.0146**	**0.0143**	12.179	**9.185**

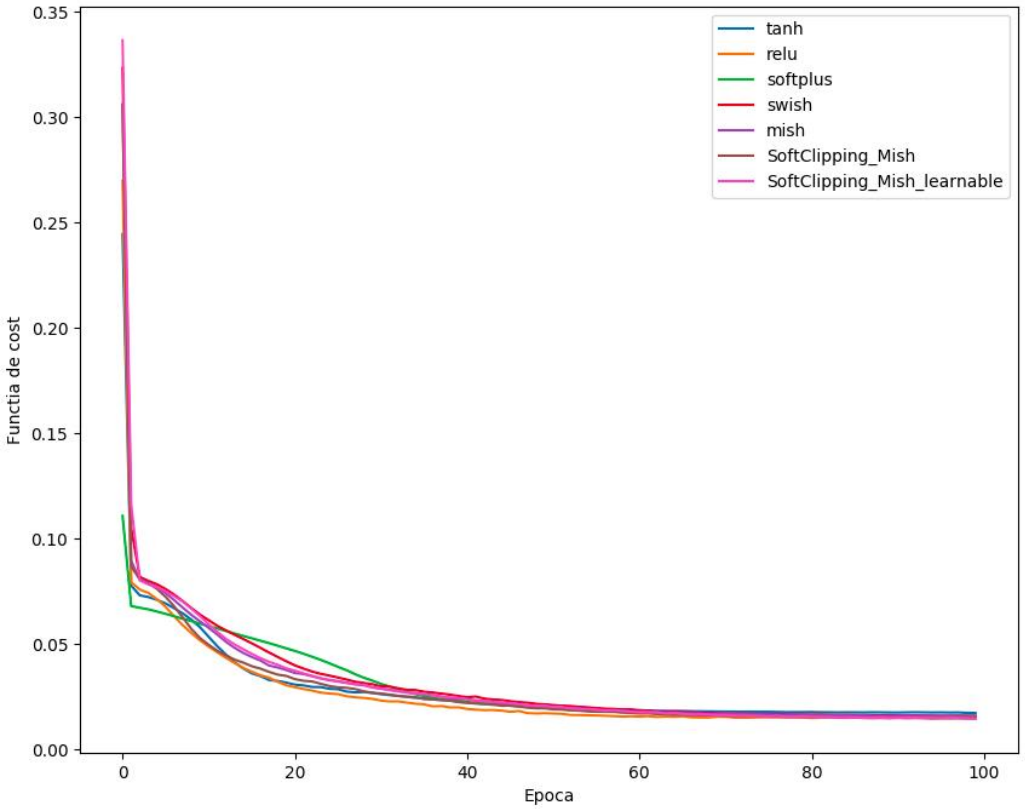


Fig.7.7.5. 3 Funcția de cost pentru arhitectura LSTM

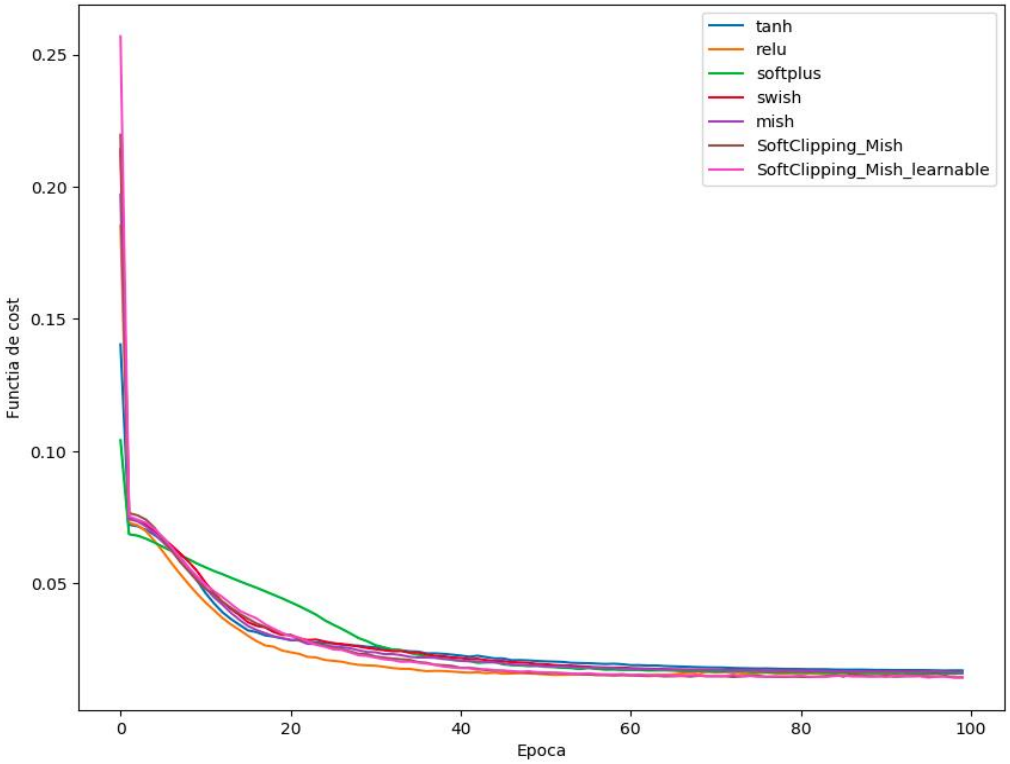


Fig.7.7.5. 4 Funcția de cost pentru arhitectura GRU

b) Arhitecturile LSTM și GRU cu un strat ascuns

În acest experiment am introdus un strat ascuns cu 16 neuroni, după fiecare strat am aplicat dropout (0.20). Am aplicat același raționament în ambele arhitecturi, păstrând aceleași condiții pentru o comparație cât mai exactă.

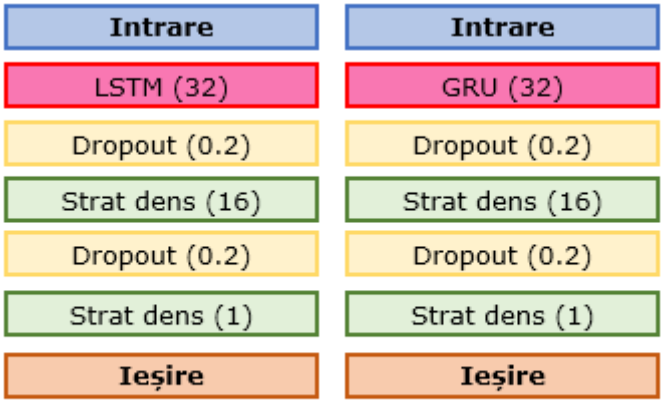


Fig.7.7.5. 5 Arhitecturile cu un strat ascuns a) LSTM și b) GRU

Tabel 7.7.5. 2 Rezultate experimentale pe arhitecturile LSTM și GRU cu un strat ascuns pentru setul de date Beijing PM2.5

Funcția de activare	**Arhitectura LSTM**	**Arhitectura GRU**	**Arhitectura LSTM**	**Arhitectura GRU**
	Functia de cost		***RMSE***	
Sigmoida	0.0178	0.0180	20.662	23.138
tanh	0.0176	0.0178	18.681	18.361
ReLU	0.0170	0.0159	23.170	13.999
Softplus	0.0182	0.0177	17.032	17.500
Swish	0.0153	0.0168	11.249	17.663
Mish	0.0148	**0.0146**	**10.125**	**9.594**
SC Mish	0.0176	0.0159	20.632	15.319
SCL Mish	**0.0144**	0.0158	11.236	19.492

7.8. Concluzii

1. Din studiul *„Modificarea dinamică a funcției de activare folosind algoritmul de propagare înapoi în rețelele neuronale artificiale”,* putem observa că rata de eroare obținută a fost de circa 2.3% pentru Variația erorii per epocă pe setul de date Iris (β=5%, variant=*Adevărat),* cu o descreștere rapidă, s-a putut vedea în Fig.7.1.1.1 că eroarea pe setul de validare a scăzut de la 23% la 2.3% foarte rapid, în primele 50 de epoci, ceea ce arată că sigmoida cu parametrul β modificat dinamic în timpul antrenamentului reprezintă o bună alegere atât în cazul unei

sarcini de clasificare pe setul de date multivariat *Iris* dar și pentru analiza variabilei diagnostic (evaluată binar) pentru un pacient dacă prezintă semne de diabet după anumite criterii sau nu pe setul de *Diabetes*. Acest studiu a fost extins pe încă alte 9 seturi de date pentru a valida în mod corespunzător eficacitatea acestei soluții, arătând că este o soluție robustă și consistentă fiind independentă de setul de date folosit. De asemenea, putem observa că există diferențe semnificative între o activare dinamică a funcției cu β = 10% și o activare dinamică a funcției cu β = 5%. Dar ceea ce putem remarca în ambele cazuri este o îmbunătățire a performanței de clasificare. O altă remarcă este necesitatea evidentă a găsirii inițializării corespunzătoare pentru modificarea dinamică optime a funcției de activare de la o epocă de antrenament la o altă epocă.

2. Din studiul *„Cele mai utilizate funcții de activare: clasic versus curent"*, putem concluziona că funcția ReLU rămâne o soluție robustă, consistentă și greu de depășit datorită aplicării facile și a unui comportament favorabil pe o diversitate cât mai mare de arhitecturi și seturi de date.

3. Din studiul *„TeLU: o nouă funcție de activare pentru Învățarea Profundă"* putem trage concluziile din scenariile testate astfel:
    - Din punct de vedere al acurateței pe setul de validare pentru setul de date *CIFAR-10* folosind arhitectura **MobileNetV1** (Experimentul 3), funcția de activare TeLU a condus la o îmbunătățire de performanță depășind funcțiile în modul următor:
        - cu ***3.64%*** mai bine ca LReLU,
        - cu ***3.06%*** mai bine ca PReLU,
        - cu ***3.78%*** mai bine ca GELU,
        - cu ***0.23%*** mai bine ca ReLU,
        - cu ***0.68%*** mai bine ca Swish-1 și
        - cu ***2.93%*** mai bine ca Swish-1 învățabilă.
    - Din punct de vedere al acurateței pe setul de validare pentru setul de date *CIFAR-10* folosind arhitectura **LeNet-5** (Experimentul 3 - Tabelul 7.2.3.2), funcția de activare TeLU a condus la o îmbunătățire de performanță depășind funcțiile în modul următor:
        - cu ***2.56%*** mai bine ca TeLU învățabilă,
        - cu ***2.68%*** mai bine ca TanhExp,
        - cu ***3.08%*** mai bine ca ReLU și
        - cu ***3.25%*** mai bine ca Mish.
    - Din punct de vedere al acurateței pe setul de validare pentru setul de date *CIFAR-10* folosind arhitectura **AlexNet** ((Experimentul 3 - Tabelul 7.2.3.2), funcția de activare TeLU a condus la o îmbunătățire de performanță depășind funcțiile în modul următor:
        - cu ***12.89%*** mai bine ca TeLU învățabilă,
        - cu ***0.01%*** mai bine ca TanhExp,
        - cu ***1.99%*** mai bine ca ReLU și
        - cu ***0.42%*** mai bine ca Mish.

- În cazul setului de date *CIFAR-100* folosind arhitectura **MobileNetV2** (Experimentul 4), creșterea performanței pentru funcția TeLU a fost și mai semnificativă, depășind funcțiile în modul următor:
  - cu ***9.13%*** mai bine ca LReLU,
  - cu ***23.47%*** mai bine ca PReLU,
  - cu ***4.05%*** mai bine ca Softplus,
  - cu ***17.06%*** mai bine ca ELU,
  - cu ***14.32%*** mai bine ca SELU,
  - cu ***16.15%*** mai bine ca GELU,
  - cu ***30.29%*** mai bine ca ReLU și
  - cu ***0.77%*** mai bine ca Swish-1. (Swish-1 este cazul când funcția Swish are parametru predefinit).
- Din tabelul 7.2.4.2. cu acuratețea de validare pe setul de date *CIFAR-100* în cazul arhitecturii **AlexNet** (Experimentul 4) se poate vedea că funcția TeLU (cu parametru predefinit) a condus la obținerea celei mai bune performanțe astfel:
  - cu ***13.90%*** mai bine ca TeLU învățabilă,
  - cu ***1.20%*** mai bine ca TanhExp,
  - cu ***1.39%*** mai bine ca ReLU și
  - cu ***1.18%*** mai bine ca Mish.
- Din tabelul 7.2.4.2. cu acuratețea de validare pe setul de date CIFAR-100 în cazul arhitecturii LeNet-5 (Experimentul 4) se poate vedea că funcția TeLU învățabilă a condus la obținerea celei mai bune performanțe astfel:
  - cu ***1.64%*** mai bine ca TeLU,
  - cu ***0.12%*** mai bine ca TanhExp,
  - cu ***0.25%*** mai bine ca ReLU și
  - cu ***1.30%*** mai bine ca Mish.

4. Din studiul *„Îmbunătățirea preciziei rețelelor neuronale profunde prin dezvoltarea de noi funcții de activare”* putem trage concluziile din scenariile testate astfel:
   - Din punct de vedere al acurateței pe setul de validare pentru setul de date CIFAR-10 folosind arhitectura **ResNet18** (Experimentul 1), funcția de activare TSReLU a condus la o îmbunătățire de performanță depășind funcțiile în modul următor:
     - cu ***47.77%*** mai bine ca Sigmoida,
     - cu ***16.62%*** mai bine ca tanh,
     - cu ***4.21%*** mai bine ca ReLU,
     - cu ***2.08%*** mai bine ca Swish și
     - cu ***4.51%*** mai bine ca TSReLU învățabilă.

   - Din punct de vedere al acurateței pe setul de validare pentru setul de date CIFAR-10 folosind arhitectura **Resnet34** (Experimentul 1), funcția de activare TSReLU învățabilă a condus la o îmbunătățire de performanță depășind funcțiile în modul următor:
     - cu ***71.17%*** mai bine ca Sigmoida,
     - cu ***17.33%*** mai bine ca tanh,
     - cu ***5.34%*** mai bine ca ReLU,

- cu ***0.02%*** mai bine ca Swish și
- cu ***2.03%*** mai bine ca TSReLU.

- Din punct de vedere al acurateței pe setul de validare pentru setul de date CIFAR-100 folosind arhitectura **Resnet18** (Experimentul 2), funcția de activare TSReLU învățabilă a condus la o îmbunătățire de performanță depășind funcțiile în modul următor:
  - cu ***39.97%*** mai bine ca Sigmoida,
  - cu ***17.55%*** mai bine ca tanh,
  - cu ***4.62%*** mai bine ca ReLU,
  - cu ***0.27%*** mai bine ca Swish și
  - cu ***0.07%*** mai bine ca TSReLU.

- Din punct de vedere al acurateței pe setul de validare pentru setul de date CIFAR-100 folosind arhitectura **Resnet34** (Experimentul 2), funcția de activare TSReLU învățabilă a condus la o îmbunătățire de performanță depășind funcțiile în modul următor:
  - cu ***37.90%*** mai bine ca Sigmoida,
  - cu ***19.58%*** mai bine ca tanh,
  - cu ***3.09%*** mai bine ca ReLU,
  - cu ***0.45%*** mai bine ca Swish și
  - cu ***2.40%*** mai bine ca TSReLU.

- Din punct de vedere al acurateței pe setul de validare pentru setul de date Câini și Pisici folosind arhitectura **VGG16 pre-antrenat** (Experimentul 3), funcția de activare TSReLU a condus la o îmbunătățire de performanță depășind funcțiile în modul următor:
  - cu ***1.99%*** mai bine ca Sigmoida,
  - cu ***0.13%*** mai bine ca tanh,
  - cu ***0.35%*** mai bine ca ReLU,
  - cu ***0.55%*** mai bine ca Swish,
  - cu ***0.22%*** mai bine ca TSReLU învățabilă,
  - cu ***0.48%*** mai bine ca TBSRelu și
  - cu ***0.99%*** mai bine ca TBSRelu învățabilă.

5. Din studiul *„P-Swish: Funcție de activare cu parametri învățabili bazată pe funcția de activare Swish în Învățarea Profundă”*, putem trage concluziile din scenariile testate astfel:

   - Din punct de vedere al acurateței pe setul de validare pentru setul de date CIFAR-10 folosind arhitectura **LeNet-5** (Experimentul 1), funcția de activare P-swish învățabilă parțial (α învățabil (1) și β = 0, δ =1) a condus la o îmbunătățire de performanță depășind funcțiile în modul următor:
     - cu ***1.89%*** mai bine ca ReLU,
     - cu ***15.11%*** mai bine ca Sigmoida,
     - cu ***0.14%*** mai bine ca Swish,
     - cu ***1.22%*** mai bine ca P-swish învățabilă (α și β învățabili și δ = 1),
     - cu ***0.67%*** mai bine ca P-swish constantă (α = 1, β = 0 și δ=1) și

- cu ***2.35%*** mai bine ca P-swish învățabilă parțial (α învățabil (1), β = 0 și δ învățabil (1)).

- Din punct de vedere al acurateței pe setul de validare pentru setul de date CIFAR-100 folosind arhitectura **LeNet-5** (Experimentul 2), funcția de activare P-swish constantă (α = 1, β = 0 și δ=1) a condus la o îmbunătățire de performanță depășind funcțiile în modul următor:
  - cu ***2.63%*** mai bine ca ReLU,
  - cu ***16.46%*** mai bine ca Sigmoida,
  - cu ***0.54%*** mai bine ca Swish,
  - cu ***1.48%*** mai bine ca P-swish învățabilă (α și β învățabili și δ = 1),
  - cu ***1.58%*** mai bine ca P-swish învățabilă parțial (α învățabil (1) și β = 0, δ =1) și
  - cu ***2.38%*** mai bine ca P-swish învățabilă parțial (α învățabil (1), β = 0 și δ învățabil (1)).

- Din punct de vedere al acurateței pe setul de validare pentru setul de date CIFAR-100 folosind arhitectura **NiN** (Experimentul 4), funcția de activare P-swish constantă (α = 1, β = 0 și δ=1) a condus la o îmbunătățire de performanță depășind funcțiile în modul următor:
  - cu ***1.74%*** mai bine ca ReLU,
  - cu ***39.53%*** mai bine ca Sigmoida,
  - cu ***0.23%*** mai bine ca Swish,
  - cu ***0.65%*** mai bine ca P-swish învățabilă (α și β învățabili și δ = 1),
  - cu ***0.92%*** mai bine ca P-swish învățabilă parțial (α învățabil (1) și β = 0, δ =1) și
  - cu ***0.19%*** mai bine ca P-swish învățabilă parțial (α învățabil (1), β = 0 și δ învățabil (1)).

- Din punct de vedere al acurateței pe setul de validare pentru setul de date CIFAR-10 folosind arhitectura **ResNet34** (Experimentul 5), funcția de activare P-swish învățabilă parțial (α învățabil (1), β = 0 și δ învățabil (1)) a condus la o îmbunătățire de performanță depășind funcțiile în modul următor:
  - cu ***2.32%*** mai bine ca ReLU,
  - cu ***12.94%*** mai bine ca Sigmoida,
  - cu ***0.51%*** mai bine ca Swish,
  - cu ***0.70%*** mai bine ca P-swish învățabilă (α și β învățabili și δ = 1),
  - cu ***0.39%*** mai bine ca P-swish constantă (α = 1, β = 0 și δ=1) și
  - cu ***0.27%*** mai bine ca P-swish învățabilă parțial (α învățabil (1) și β = 0, δ =1).

- Din punct de vedere al acurateței pe setul de validare pentru setul de date CIFAR-100 folosind arhitectura **ResNet34** (Experimentul 6), funcția de activare P-swish învățabilă (α și β învățabili și δ = 1) a condus la o îmbunătățire de performanță depășind funcțiile în modul următor:
  - cu ***0.80%*** mai bine ca ReLU,
  - cu ***10.45%*** mai bine ca Sigmoida,

- cu ***0.11%*** mai bine ca Swish,
- cu ***0.13%*** mai bine ca P-swish constantă (α = 1, β = 0 și δ=1),
- cu ***0.33%*** mai bine ca P-swish învățabilă parțial (α învățabil (1) și β = 0, δ =1) și
- cu ***0.54%*** mai bine ca P-swish învățabilă parțial (α învățabil (1), β = 0 și δ învățabil (1)).

- Din punct de vedere al acurateței pe setul de validare pentru setul de date CIFAR-10 folosind arhitectura **Xception** (Experimentul 7), funcția de activare P-swish învățabilă parțial (α învățabil (1) și β = 0, δ =1) a condus la o îmbunătățire de performanță depășind funcțiile în modul următor:
  - cu ***0.48%*** mai bine ca ReLU,
  - cu ***0.25%*** mai bine ca Sigmoida,
  - cu ***0.43%*** mai bine ca Swish,
  - cu ***0.49%*** mai bine ca P-swish învățabilă (α și β învățabili și δ = 1),
  - cu ***0.67%*** mai bine ca P-swish constantă (α = 1, β = 0 și δ=1) și
  - cu ***0.22%*** mai bine ca P-swish învățabilă parțial (α învățabil (1), β = 0 și δ învățabil (1)).
- Din punct de vedere al acurateței pe setul de validare pentru setul de date CIFAR-100 folosind arhitectura **Xception** (Experimentul 8), funcția de activare P-swish constantă (α = 1, β = 0 și δ=1) a condus la o îmbunătățire de performanță depășind funcțiile în modul următor:
  - cu ***0.98%*** mai bine ca ReLU,
  - cu ***1.54%*** mai bine ca Sigmoida,
  - cu ***0.15%*** mai bine ca Swish,
  - cu ***2.38%*** mai bine ca P-swish învățabilă (α și β învățabili și δ = 1),
  - cu ***0.15%*** mai bine ca P-swish învățabilă parțial (α învățabil (1) și β = 0, δ =1) și
  - cu ***5.53%*** mai bine ca P-swish învățabilă parțial (α învățabil (1), β = 0 și δ învățabil (1)).

- Din punct de vedere al acurateței pe setul de test pentru o arhitectură personalizată pentru o sarcină de NLP pe setul de date Reuters (Experimentul 11), funcția de activare P-Swish învățabilă parțial (α învățabil (1), β = 0 și δ învățabil (1)) a condus la o îmbunătățire de performanță depășind funcțiile în modul următor:
  - cu ***1.73%*** mai bine ca ReLU,
  - cu ***22.03%*** mai bine ca Sigmoida,
  - cu ***1.73%*** mai bine ca Swish,
  - cu ***0.89%*** mai bine ca P-Swish învățabilă (α și β învățabili δ = 1),
  - cu ***2.40%*** mai bine ca P-Swish constantă (α = 1, β = 0 și δ=1) și
  - cu ***1.77%*** mai bine ca P-Swish învățabilă parțial (α învățabil (1) și β = 0, δ =1).

- Din punct de vedere al acurateței pe setul de validare pentru o arhitectură personalizată pentru o sarcină de NLP pe setul de date IMDB (Experimentul 12), funcția de activare P-Swish învățabilă parțial (α

învățabil (1) și β = 0, δ =1) a condus la o îmbunătățire de performanță depășind funcțiile în modul următor:

- cu ***0.09%*** mai bine ca ReLU,
- cu ***36.64%*** mai bine ca Sigmoida,
- cu ***0.33%*** mai bine ca Swish,
- cu ***0.12%*** mai bine ca P-Swish învățabilă (α și β învățabili δ = 1),
- cu ***0.06%*** mai bine ca P-Swish constantă (α = 1, β = 0 și δ=1) și
- cu ***5.98%*** mai bine ca P-Swish învățabilă parțial (α învățabil (1), β = 0 și δ învățabil (1)).

- Din punct de vedere al acurateței pe setul de validare pentru o arhitectură personalizată folosind LSTM pentru o sarcină de NLP pe setul de date FastText crawl 300d 2M (Experimentul 13), funcția de activare P-Swish învățabilă (α și β învățabili δ = 1) a condus la o îmbunătățire de performanță depășind funcțiile în modul următor:
  - cu ***1.65%*** mai bine ca ReLU,
  - cu ***1.46%*** mai bine ca Sigmoida,
  - cu ***1.39%*** mai bine ca Swish,
  - cu ***1.33%*** mai bine ca P-Swish constantă (α = 1, β = 0 și δ=1),
  - cu ***1.63%*** mai bine ca P-Swish învățabilă parțial (α învățabil (1) și β = 0, δ =1) și
  - cu ***0.96%*** mai bine ca P-Swish învățabilă parțial (α învățabil (1), β = 0 și δ învățabil (1)).

- Din punct de vedere al acurateței pe setul de validare pentru o arhitectură U-Net pe o sarcină de segmentare semantică (Experimentul 14), funcția de activare P-Swish constantă (α = 1, β = 0 și δ=1) a condus la o îmbunătățire de performanță depășind funcțiile în modul următor:
  - cu ***0.23%*** mai bine ca ReLU,
  - cu ***16.09%*** mai bine ca Sigmoida,
  - cu ***1.35%*** mai bine ca Swish,
  - cu ***0.03%*** mai bine ca P-Swish învățabilă (α și β învățabili δ = 1),
  - cu ***0.09%*** mai bine ca P-Swish învățabilă parțial (α învățabil (1) și β = 0, δ =1) și
  - cu ***0.58%*** mai bine ca P-Swish învățabilă parțial (α învățabil (1), β = 0 și δ învățabil (1)).

  De remarcat faptul că funcția P-Swish învățabilă parțial (α învățabil (1) și β = 0, δ =1) a obținut cel mai bun procent ***56,14%*** pentru IOU pe setul de validare în comparație cu restul funcțiilor de activare.

6. Din studiul *„Noi funcții de activare bazate pe funcția de activare TanhExp în Învățarea Profundă”,* putem trage concluziile din scenariile testate astfel:
   - În ceea ce privește acuratețea pe setul de validare a setului de date MNIST pentru o arhitectură ca în cazul funcției de activare TanhExp (Experimentul 1), funcția de activare *a_TanhExp învățabilă* a condus la o îmbunătățire de performanță depășind funcțiile în modul următor:
     - cu ***0.01%*** mai bine ca ReLU,
     - cu ***0.29%*** mai bine ca tanh,
     - cu ***0.02%*** mai bine ca TanhExp,

- cu ***0.15%*** mai bine ca TanhExp învățabilă,
- cu ***0.44%*** mai bine ca a_TanhExp

- În ceea ce privește acuratețea pe setul de validare a setului de date Fashion-MNIST pentru o arhitectură ca în cazul funcției de activare TanhExp (Experimentul 2), funcția de activare *a_TanhExp învățabilă* a condus la o îmbunătățire de performanță depășind funcțiile în modul următor:
  - cu ***0.04%*** mai bine ca ReLU,
  - cu ***0.30%*** mai bine ca tanh,
  - cu ***0.19%*** mai bine ca TanhExp,
  - cu ***9.13%*** mai bine ca TanhExp învățabilă,
  - cu ***0.10%*** mai bine ca a_TanhExp

- Din punct de vedere al acurateței pe setul de validare pentru setul de date CIFAR-10 pentru arhitectura **LeNet-5** (Experimentul 3), funcția de activare *a_TanhExp învățabilă* a condus la o îmbunătățire de performanță depășind funcțiile în modul următor:
  - cu ***0.25%*** mai bine ca ReLU,
  - cu ***5.37%*** mai bine ca tanh,
  - cu ***1.23%*** mai bine ca TanhExp,
  - cu ***26.07%*** mai bine ca TanhExp învățabilă,
  - cu ***0.54%*** mai bine ca a_TanhExp
- Din punct de vedere al acurateței pe setul de validare pentru setul de date CIFAR-10 pentru arhitectura **AlexNet** (Experimentul 3), funcția de activare *a_TanhExp* a condus la o îmbunătățire de performanță depășind funcțiile în modul următor:
  - cu ***1.65%*** mai bine ca ReLU,
  - cu ***9.54%*** mai bine ca tanh,
  - cu ***0.99%*** mai bine ca TanhExp,
  - cu ***5.69%*** mai bine ca TanhExp învățabilă,
  - cu ***1.78%*** mai bine ca a_TanhExp învățabilă

- Din punct de vedere al acurateței pe setul de validare pentru setul de date CIFAR-100 pentru arhitectura **LeNet-5** (Experimentul 3), funcția de activare *TanhExp învățabilă* a condus la o îmbunătățire de performanță depășind funcțiile în modul următor:
  - cu ***0.77%*** mai bine ca ReLU,
  - cu ***3.09%*** mai bine ca tanh,
  - cu ***0.45%*** mai bine ca TanhExp,
  - cu ***2.88%*** mai bine ca a_TanhExp,
  - cu ***0.03%*** mai bine ca a_TanhExp învățabilă

- Din punct de vedere al acurateței pe setul de validare pentru setul de date CIFAR-100 pentru arhitectura **AlexNet** (Experimentul 3), funcția de activare *a_TanhExp învățabilă* a condus la o îmbunătățire de performanță depășind funcțiile în modul următor:

- cu ***0.74%*** mai bine ca ReLU,
- cu ***7.79%*** mai bine ca tanh,
- cu ***0.37%*** mai bine ca TanhExp,
- cu ***4.89%*** mai bine ca TanhExp învățabilă,
- cu ***0.29%*** mai bine ca a_TanhExp

- Din punct de vedere al funcției de cost pe setul de validare pentru setul de date Numenta Anomaly Benchmark (NAB) [287] pentru detecția anomaliilor în seriile de timp pe o arhitectură personalizată (pornind de la o arhitectură existentă [289]) (Experimentul 4), funcția de activare *a_TanhExp* a condus la obținerea celei mai mici funcții de cost depășind funcțiile în modul următor:
  - cu ***0,0037*** mai puțin ca ReLU,
  - cu ***0,0007*** mai puțin ca tanh,
  - cu ***0,0009*** mai puțin ca TanhExp,
  - cu ***0,0037*** mai puțin ca TanhExp învățabilă,
  - cu ***0,0006*** mai puțin ca a_TanhExp învățabilă.

7. Din studiul *„Dezvoltarea de noi funcții de activare în detecția anomaliilor în serii de timp cu autoencoder LSTM”* [290], putem trage concluziile din scenariul testat astfel:
   - Din punct de vedere al funcției de cost pe setul de validare pentru setul de date index S&P 500 [291] în detecția anomaliilor în serii de timp cu autoencoder LSTM pe o arhitectură personalizată (pornind de la o arhitectură existentă [292]), funcția de activare *P_Talu* a condus la obținerea celei mai mici funcții de cost depășind funcțiile în modul următor:
   - Dimensiunea lotului = 64
     - cu ***0,0622*** mai puțin ca ReLU,
     - cu ***0,0021*** mai puțin ca tanh,
     - cu ***0,0046*** mai puțin ca TaLu,
     - cu ***0,0019*** mai puțin ca TaLu învățabilă.

   - Dimensiunea lotului = 32
     - cu ***0,066*** mai puțin ca ReLU,
     - cu ***0,0051*** mai puțin ca tanh,
     - cu ***0,0725*** mai puțin ca TaLu,
     - cu ***0,0022*** mai puțin ca TaLu învățabilă.

8. Din studiul *„Soft Clipping Mish - o nouă funcție de activare”,* pe baza scenariilor testate putem trage concluziile:
   - În ceea ce privește acuratețea pe setul de validare a setului de date *MNIST* (Experimentul 1) pentru arhitectura LeNet cu normalizarea lotului, funcțiile de activare ***SC Mish și SC Mish învățabilă*** au condus la o îmbunătățire de performanță depășind toate celălalte funcții cu care ne-am comparat: Sigmoida, tanh, ReLU, Softplus, Mish și Swish.

- În ceea ce privește acuratețea pe setul de validare a setului de date *Fashion-MNIST* (Experimentul 2) pentru arhitectura LeNet cu normalizarea lotului, funcția de activare ***SCL Mish învățabilă*** a obținut cea mai bună acuratețe depășind toate celălalte funcții cu care ne-am comparat: Sigmoida, tanh, ReLU, Softplus, Mish și Swish. În cazul acestui set de date *SC Mish* a dat rezultate mai slabe ca ReLu și SCL Mish dar totuși a dat rezultate mai bune ca Sigmoida, tanh, Softplus, Mish și Swish.

- În ceea ce privește acuratețea pe setul de validare a setului de date *CIFAR-10* (Experimentul 3) pentru arhitectura LeNet cu normalizarea lotului, funcțiile de activare *SC Mish* și *SC Mish învățabilă* au obținut top 3 cea mai bună acuratețe depășind funcții precum: Sigmoida, tanh și Softplus. Din graficul funcției de cost s-a putut vedea că *SC Mish învățabilă* a condus la cea mai mică funcție de cost.

- În ceea ce privește acuratețea pe setul de validare a setului de date *CIFAR-100* (Experimentul 4) pentru arhitectura LeNet cu normalizarea lotului, funcția de activare *SC Mish* a obținut cea mai bună acuratețe, depășind astfel Sigmoida, tanh, ReLU, Softplus, Mish și Swish, iar *SC Mish învățabilă* a dat rezultate mai slabe ca ReLU și Swish dar totuși a depășit funcțiile Sigmoida, tanh, Softplus și Mish.

- În ceea ce privește funcția de cost pe setul de validare a setului de date *Beijing PM2.5.* (Experimentul 5) pentru arhitecturile LSTM și GRU fără straturi acunse, funcțiile de activare *SC Mish* și *SCL Mish* au obținut cea mai bună funcție de cost (cea mai mică), depășind astfel Sigmoida, tanh, ReLU, Softplus, Mish și Swish. Iar în cazul arhitecturii LSTM cu un strat ascuns funcția propusă SCL Mish a obținut cea mai bună funcție de cost, depășind funcțiile Sigmoida, tanh, ReLU, Softplus, Mish și Swish. Iar pe arhitectura GRU, SC Mish s-a clasat pe locul 2 în top 3 depășind funcțiile Sigmoida, tanh, ReLU, Softplus, Swish și Mish.
  De asemenea s-a putut vedea că funcția *SCL Mish* a obținut cea mai mică valoare pentru RMSE față de toate celălalte funcții de activare.

În cadrul acestui capitol s-a putut vedea că cele 14 funcții de activare propuse aduc îmbunătățiri semnificative în comparație cu cele mai utilizate funcții de activare pentru seturi de date variate și tipuri de sarcini diferite.

8. Concluzii

Alegerea corectă a funcției de activare conduce la evitare celor două probleme foarte populare: dispariția și explozia gradienților. Primul caz este cauzat de gradienți prea mici în timpul propagării înapoi. În cel de-al doilea caz gradientul poate face ca ponderile să devină prea mari și, conducând la erori "nan" (not a number – nu este număr).

Influența naturii setului de date nu mai este reflectată în ponderile rețelei neuronale, dar este reflectată în funcția de activare. Din acest motiv, modificarea dinamică a funcției de activare poate reprezenta o modalitate semnificativă de a îmbunătăți rețeaua neuronală de clasificare într-o gamă largă de direcții unde poate avansa acest concept.

Funcția sigmoidă este cea mai bună alegere în clasificare pentru modul său aplicabil și eroarea medie pătrată (**M**ean **S**quared **E**rror - MSE). De asemenea, [294] s-a arătat că performanța este afectată de dimensiunea setului de date, numărul de clase și timpul petrecut pentru antrenament. De obicei, sigmoida este folosită pe stratul de ieșire al unei clasificări binare, unde rezultatul este 0 sau 1, deoarece valoarea pentru funcția sigmoidă este cuprinsă între 0 și 1, deci rezultatul poate fi 1 dacă valoarea este mai mare de 0,5 și 0 altfel. De asemenea, din experimente am văzut că funcția *tanh* este de obicei folosită în straturile ascunse ale unei rețele neuronale, deoarece valorile sale se situează între -1 și 1, prin urmare, media pentru stratul ascuns este de 0 sau foarte aproape de 0, proprietate care ajută la centrarea datelor în apropierea de valoarea medie. De asemenea, antrenarea următorului strat se face mult mai ușor decât în cazul funcției sigmoide.

ReLU este mai puțin costisitoare din punct de vedere al calculului decât *tanh* și sigmoida, deoarece presupune operații matematice mai simple. De remarcat este că la un moment dat, doar câțiva neuroni sunt activi, ceea ce face ca rețeaua să scadă, făcând-o eficientă și ușoară pentru calcul. Softmax se folosește de obicei atunci când avem mai multe clase. Funcția Softmax adună ieșirile pentru fiecare clasă între 0 și 1 și care se împart la suma ieșirilor. Funcția Softmax este folosită în mod frecvent pe stratul de ieșire al rețelei cu scopul definirii clasei fiecărei intrări.

Performanța generală a unei funcții de activare într-o rețea neuronală nu depinde doar de viteza de execuție a acesteia, ci și de gradul în care forma acesteia este favorabilă utilizării într-un context de rețea în general, precum și de orice obiectiv sau sarcină date. Caracteristici particulare, cum ar fi minimul și maximul, monotonicitatea, continuitatea, tipul de abordare a asimptotelor și gradientul în special în regiunea apropiată de 0, pot afecta eficacitatea generală a unei funcții de activare, aspecte pe care le-am analizat și eu în această teză.

Ca urmare a multiplelor funcții de activare prezentate în cadrul acestei teze, atât parametrice cât și non-parametrice putem concluziona că funcțiile de activare recomandate depind de arhitectura rețelei, de profunzimea rețelei și tipul de sarcină.

Am prezentat rezultate experimentale propunând mai multe funcții de activare diferite pentru diferite arhitecturi ale rețelelor neuronale profunde în vederea soluționării diferitelor sarcini de la CV la NLP. Aceste rezultate demonstrează că este posibilă îmbunătățirea performanței rețelelor neuronale profunde prin utilizarea unei funcții de activare mult mai eficiente.

8.1. Contribuții

Contribuţiile personale în domeniul funcțiilor de activare sunt prezentate în următoarea secțiune.

Contribuții majore

În cadrul acestei secțiuni voi prezenta contribuțiile majore din această teză și anume:

(i) *Adaptarea funcției sigmoide în Modificarea dinamică a funcției de activare folosind algoritmul propagarea înapoi în rețelele neuronale artificiale*.

În cadrul acestui studiu am observat că introducerea conceptului de modificare dinamică în timpul procesului de antrenament duce la creșterea performanței de clasificare. Putem remarca faptul că un algoritm de formare clasic, cum ar fi algoritmul propagarea înapoi, devine mai flexibil împreună cu modificarea dinamică a funcției de activare, în sensul că algoritmul de învățare este ghidat mai repede către o eroare de clasificare, în general mai mică. Metoda propusă în subcapitolul 6.6.1. propune să monitorizeze eroarea și să verifice evoluția performanței pentru diferite seturi de date. În [294] s-a arătat că, în cazul unei arhitecturi neuronale fără straturi ascunse, folosind funcția sigmoidă ca funcție de activare, s-a observat că valoarea parametrului β scade dacă eroarea crește în timpul unei epoci. Eroarea curentă a neuronului se compară cu eroarea anterioară a aceluiași neuron. În funcție de această comparație, parametrul β se schimbă asincron și se păstrează eroarea curentă pentru compararea acesteia cu eroarea următoare, comparație care va determina sensul de modificare a acestui parametru în vederea creșterii performanței. Rezultatele experimentatele în vederea determinării eficacității unei funcții de activare mi-au confirmat ipotezele care sugerează că, atât intervalul nelimitat, cât și centrarea în zero sunt calități benefice în funcțiile de activare. Drept urmare în dezvoltarea noilor funcții de activare am urmărit ca aceste funcții să beneficieze de aceste proprietăți. De asemenea, în [295] s-a arătat că o funcție sigmoidă adaptivă cu un singur strat ascuns conduce la o modelare mai bună a datelor.

(ii) *Cele mai utilizate funcții de activare: clasic versus curent*, care constituie a doua propunere, clasifică funcțiile de activare în clasice și curente prin sublinierea importanței fiecărei funcții cu plusurile și minusurile aferente. S-a putut vedea că funcții curente vin ca o rezolvare la funcțiile clasice (tradiționale).

(iii) *TeLU: o nouă funcție de activare pentru Învățarea Profundă*.

O nouă funcție de activare TeLU, pe care am introdus-o în această teză, aduce performanțe suplimentare pe seturi de date precum MNIST, Fashion-MNIST, CIFAR-10 și CIFAR-100. Această funcție de activare este o combinație de ReLU, *tanh* și ELU. Simplitatea și rezultatele sale prin creșterea în timp real a datelor o fac adecvată atunci când dorim să

îmbunătățim performanța rețelelor neuronale profunde. Am testat pe mai multe sarcini de Vedere Artificială care validează această propunere. Totodată, proprietățile funcției de activare TeLU, că este nemărginită la partea superioară și centrată în zero conduc la o îmbunătățire a performanței în raport cu celălalte funcții de activare, motiv pentru care pentru a crește performanța recomand folosirea acestei funcții dar trebuie să ținem cont de timpul de execuție mai ridicat. Este recomandat ca atunci când alegem funcția de activare, derivata acesteia să nu fie aproape sau egală cu 0. Ambele funcții arată un gradient mai abrupt aproape de zero, care poate accelera actualizarea parametrilor din rețea.

(iv) Îmbunătățirea preciziei rețelelor neuronale profunde prin dezvoltarea de noi funcții de activare: TSReLU, TSReLU învățabilă, TBSReLU și TBSReLU învățabilă.

În acest studiu, am propus patru noi funcții de activare, care aduc performanțe suplimentare pe seturi de date, cum ar fi CIFAR-10, CIFAR-100 și Câini și Pisici. Aceste funcții de activare, le-am numit TSReLU, TSReLU învățabilă, TBSReLU și TBSReLU învățabilă, deoarece sunt o combinație de ReLU, tanh, sigmoidă și sigmoidă bipolară. Sunt ușor de folosit și rezultatele lor cu mărirea în timp real a datelor ne arată că sunt potrivite să fie alese atunci când dorim să îmbunătățim performanța rețelelor neuronale profunde. Funcții de activare TSReLU și TSReLU învățabilă sunt curbe netede, ceea ce înseamnă că ieșirea lor va fi de asemenea lină. Această proprietate oferă o mulțime de avantaje atunci când optimizăm rețelele neuronale pentru a conduce la o convergență spre funcția de cost minimă. Există, de asemenea și un dezavantaj în funcțiile propuse legat de nevoile lor de calcul ridicat, care se datorează funcției exponențiale. Dar cu toate astea, modul lor ușor de utilizare și rezultatele acestora cu mărirea în timp real a datelor le fac potrivite pentru a le alege atunci când dorim să îmbunătățim performanța rețelelor neuronale profunde. Principalul avantaj al funcțiilor mele constă în proprietatea lor de nemărginire care este foarte dorită pentru funcțiile de activare, deoarece evită încetinirea antrenării pentru gradienții care sunt aproape de zero.

(v) P-Swish: Funcție de activare cu parametri învățabili bazată pe funcția de activare Swish în Învățarea Profundă.

În subcapitolul 6.5, în care am propus funcția de activare *P-Swish*, idee susținută de o varietate de experimente variind atât tipul de arhitectură cât și tipul de sarcină folosit, de la clasificare, segmentare semantică până la analiza sentimentului folosind Procesarea Limbajului Natural pe mai multe seturi de date. S-a putut remarca că această funcție adaptivă aduce îmbunătățiri de acurateţe comparativ cu alte funcții care reprezintă state-of-the-art și anume: sigmoida, ReLU și Swish.

Potrivit studiului meu experimental, pot spune că funcția de activare P-Swish poate fi folosită în toate rețelele neuronale profunde pentru orice tip de sarcini, fiind o bună alegerea pentru a obține precizii ridicate competitve cu funcții precum ReLU, Swish, ELU și așa mai departe.

Nemărginirea funcției P-Swish în partea superioară este o proprietate mult dorită pentru orice funcție de activare, deoarece evită saturația, care, în general, face ca antrenamentul să devină brusc lent datorită gradienților care sunt aproape de zero. De asemenea, o altă proprietate a acestei funcții este dată de mărginirea în partea inferioară care constituie un avantaj, deoarece are efect de regularizare puternic și reduce supraspecializarea. ReLU nu este o funcție de activare diferențiabilă continuu, lucru care conduce la probleme în optimizarea bazată pe gradient, problemă care nu se regăsește în cazul funcției P-Swish.

(vi) *Noi funcții de activare bazate pe funcția de activare TanhExp în Învățarea Profundă: TanhExp învățabilă, a_TanhExp și a_TanhExp învățabilă*.

Potrivit studiului meu experimental, pot spune că funcțiile de activare, TanhExp învățabilă, a_TanhExp și a_TanhExp învățabilă aduc îmbunătățiri ale performanței pe seturi de date variate, variind tipul sarcinii în Învățarea Profundă. Simplitatea și proprietățile lor le fac semnificative, aducând precizie competitivă în sarcinile de Vedere Artificială dar și pe sarcini de detecție a anomaliilor din serii de timp univariate. Datele experimentale întăresc convingerile mele că aceste funcții pot fi utilizate cu succes în acest tip de sarcini. De asemenea, sunt o soluție robustă pentru rețelele neuronale ușoare.

(vii) *Dezvoltarea de noi funcții de activare în detecția anomaliilor în serii de timp cu autoencoder LSTM: Talu învățabilă și P_Talu învățabilă*.

În cadrul acestui studiu, am propus două noi funcții de activare numite *Talu învățabilă* și *P_Talu învățabilă*. Scopul acestui studiu a fost investigarea și identificarea unei soluții de optimizare pentru îmbunătățirea performanțelor rețelei utilizând arhitectura de tip LSTM din Învățarea Profundă pentru detecția anomaliilor. Funcțiile pe care le-am propus, în urma testelor am obsevat că sunt capabile să scadă funcția de cost pe setul de validare. Comportamentul lor pe porțiuni, aspect studiat în puține studii, s-a dovedit a fi semnificativ, aducând performanțe competitive în detecția anomaliilor în seriile de timp. Datele experimentale întăresc convingerile mele că aceste funcții pot fi utilizate cu succes în acest tip de sarcini pentru seturi de date univariate.

(viii) *Soft Clipping Mish - o nouă funcție de activare*.

În cadrul acestui studiu, am propus două noi funcții de activare numite *Soft Clipping Mish (SC Mish)* și *Soft Clipping învățabilă (Soft Clipping Mish Learnable - SCL Mish)*. Aceste funcții sunt convergente spre o creștere a performanței respectiv o minimizare a funcției de cost în diferite scenarii. Ca soluție, am găsit două funcții noi de activare derivate din funcția Mish, dar care s-au dovedit a fi o soluție robustă. Din datele experimentale putem remarca că aceste funcții pot fi utilizate cu succes în diferite tipuri de sarcini de la clasificare până la predicții pentru seturi de date multivariate, spre deosebire de funcțiile *Talu învățabilă și P_Talu învățabilă* testate tot pe un tip de arhitectură LSTM dar pe un set de date univariat.

8.2. Direcții de cercetare viitoare

Pentru cercetări viitoare, aș dori să văd cum se comportă funcțiile propuse în cadrul capitolului 6:

- pe alte arhitecturi,
- în comparație cu alte funcții de activare decât cele cu care m-am comparat în studiul curent,
- și folosind și alte seturi de date pentru ca să pot varia și tipul de sarcini.

Alte direcții pe care aș vrea să le analizez ar fi:

1. Investigarea funcțiilor de cost.
2. Compararea ratelor de convergență.
3. Propunerea unei constante de normalizare pentru funcția TeLU pentru a elimina utilizarea normalizării lotului.
4. Analiza numărului de straturi și dimensiunea lotului ca să văd modul de impact asupra preciziei setului de test pe diferite sarcini pentru a vedea comportamentul funcțiilor de activare propuse.
5. Extinderea studiului curent prin explicarea rezultatelor aduse de o funcție de activare în raport cu altă funcție pe baza caracteristicilor setului de date, pentru a vedea dacă aceste funcții impactează modul de acțiune al caracteristicilor din setul de date.
6. Un alt focus ar fi să văd cum să combin funcțiile de activare pentru a îmbunătăți performanța rețelei, pentru că este important să știm ce funcții de activare să utilizăm în rețeaua neuronală, profitând astfel de proprietatea că putem utiliza diferite funcții de activare pe diferite straturi.
7. Îmbunătățirea timpului de execuție al funcțiilor de activare prin vectorizarea implementărilor funcțiilor de activare pentru a crește viteza de execuție în vederea soluționării problemelor apărute în antrenarea rețelelor neuronale profunde care presupun operații multiple de adunare și multiplicare prin utilizarea masivă a funcțiilor de activare de-a lungul straturilor. Cu toate că, o creștere a vitezei este oferită și de paralelizarea sarcinilor de muncă (unități de procesare grafică GPU) sau procesoare SIMD (Single Instruction Multiple-Data). [295]
8. O lucrare viitoare cu compararea tuturor funcțiilor noi apărute cu arhitecturile de top, utilizând seturi de date standard pentru a vedea impactul asupra performanței.
9. De asemenea, conform unui studiu recent, folosind optimizarea evoluției [296] ca o o direcție de cercetare prin combinarea cerințelor mai multor sarcini pentru a vedea cum se comportă funcțiile de activare.
10. Un studiu comparativ al funcțiilor Talu învățabilă și P_Talu învățabilă pe seturi de date multivariate atât pentru LSTM cât și în contextul unei altei arhitecturi precum GRU.
11. Un studiu al funcțiilor *SC Mish* și *SCL Mish învățabilă* pe arhitecturi mai profunde și alte seturi de date, deoarece în studiul curent am făcut experimente pe arhitecturi fără straturi ascunse și cu un singur strat ascuns. De asemenea, să extind acest studiu și pe alte seturi de date.

8.3. Publicații

Până în prezent, am următoarele lucrări acceptate și prezentate la conferințele internaționale respectiv jurnal internațional din domeniul Automaticii și Calculatoarelor:

1. Marina Adriana Mercioni, Ștefan Holban, "*The recognition of the architectural style using Data Mining techniques*", Conferința: 12th IEEE International Symposium on Applied Computational Intelligence and Informatics (SACI), Timisoara, Romania, 17-19 mai 2018, Pg: 331-337 Publicată: 2018, indexată ISI, Scopus.

2. Marina Adriana Mercioni, Ștefan Holban, "*Evaluating hierarchical and non-hierarchical grouping for develop a smart system*", Conferința: 13th International Symposium on Electronics and Telecommunications (ISETC), Timisoara, Romania, 8-9 noiembrie 2018, Pg: 114-117 Publicată: 2018, indexată ISI, Scopus.

3. Marina Adriana Mercioni, Alexandru Tiron, Ștefan Holban, "*Dynamic Modification of Activation Function using the Backpropagation Algorithm in the Artificial Neural Networks*", Jurnal: International Journal of Advanced Computer Science and Aplications, Volum: 10 Issue: 4 Pg: 51-56 Publicată: Aprilie 2019, indexată ISI, Scopus.

4. Marina Adriana Mercioni, Nina Holban, Vlad Virgiliu Todea, "*Wireless Routers and their impact on the environment*", Global and Regional in Environmental Protection, Conferența GLOREP 2018, 15- 17 Noiembrie 2018, Timisoara, Romania.

5. Marina Adriana Mercioni, Ștefan Holban, "*A survey of distance metrics in clustering data mining techniques*", ICGSP '19 Proceedings of the 2019 3rd International Conference on Graphics and Signal Processing, Pages 44-47, Hong Kong, Publicată: 01 – 03 Iunie 2019, indexată Scopus.

6. Marina Adriana Mercioni, Ștefan Holban, "*A study on hierarchical clustering and the distance metrics for identifying architectural styles*", 9 th International Conference on Energy and Environment 2019, 17-18 Octombrie 2019, Timisoara Romania.

7. Marina Adriana Mercioni, Ștefan Holban, "*The Most Used Activation Functions: Classic Versus Current*", 15th International Conference on Development and Application Systems, Suceava, Romania, May 21-23, 2020.

8. Marina Adriana Mercioni, Angel Marcel, Tat, Ștefan Holban, "*Improving the Accuracy of Deep Neural Networks Through Developing New Activation Functions*", 2020 IEEE 16th International Conference on Intelligent Computer

Communication and Processing (ICCP 2020), Cluj-Napoca, Romania, September 3-5, 2020.

9. Marina Adriana Mercioni, Ștefan Holban, "*Novel Activation Functions Based on TanhExp Activation Function in Deep Learning*", International Journal of Future Generation Communication and Networking, Vol. 13 No. 4, 2020.

10. Marina Adriana Mercioni, Ștefan Holban, "*P-Swish: Activation Function With Learnable Parameters Based on Swish Activation Function in Deep Learning*", International Symposium on Electronics and Telecommunications (ISETC 2020), Timisoara, Romania, November 05 – 06, 2020.

11. Marina Adriana Mercioni, Ștefan Holban, "*TeLU: A New Activation Function for Deep Learning*", ISETC 2020, Timisoara, Romania, November 05 – 06, 2020.

12. Marina Adriana Mercioni, Ștefan Holban, *„Soft Clipping Mish – A Novel Activation Function for Deep Learning*", The 4th International Conference on Information and Computer Technologies (ICICT 2021), Kahului, Maui Island, Hawaii, United States, March 11-14, 2021.

13. Marina Adriana Mercioni, Ștefan Holban, "*Developing Novel Activation Functions in Time Series Anomaly Detection with LSTM Autoencoder*", IEEE 15th International Symposium on Applied Computational Intelligence and Informatics (SACI), May 19-21, 2021.

Bibliografie

[1] A. Panigrahi et al., *Efect of activation functions on the training of overparametrized neural nets*, 2020.

[2] Y. Goodfellow et al., *Deep Learning*, 2016.

[3] P. Ramachandran et al., *Searching for Activation Functions*, 2017.

[4] Y. Bengio, Learning Deep Architectures for AI, 2009.

[5] F. Chollet, *Deep Learning with Python*, 2018.

[6] S. Ding, et al., *Evolutionary artificial neural networks: A review*, China, Artif Intell Rev 39, pp. 251–260, 2013.

[7] D. Whitley, *Genetic Algorithms and Neural Networks*, 1990.

[8] N. Phan et al., *A deep learning approach for human behavior prediction with explanations in health social networks: social restricted Bolzmann machine*, 2016.

[9] C. E. Nwankpa, et al., *Comparison of Trends in Practice and Research for Deep Learning*, UK, 2018.

[10] H. J. Kelley, *Gradient theory of optimal flight paths*, Ars Journal, 30, pp. 947-954, 2019.

[11] S. Dreyfus, *Artificial Neural Networks, Back Propagation and the Kelley-Bryson Gradient Procedure*, 1990.

[12] J. Schmidhuber, *Deep Learning*, Scholarpedia, 10, 2015.

[13] A. G Ivakhnenko , *The group method of data handling - A rival of the method of stochastic approximation*, Avtomatika, Vol. 3, 1968.

[14] Y. LeCun et al., *Backpropagation Applied to Handwritten Zip Code Recognition*, Neural Computation, pp. 541-551, 1989.

[15] Fukushima, K., *Neocognitron: A hierarchical neural network capable of visual pattern recognition*, Neural Networks, 1, pp. 119–130, 1988.

[16] Fukushima, *A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position*, Biological Cybernetics pp. 193–202, 1980.

[17] M. Lokanan et al., *Detecting anomalies in financial statements using machine learning algorithm: The case of Vietnamese listed firms*, Asian Journal of Accounting Research, 2019.

[18] G. Palshikar et al., *Analytics for Detection of Money Laundering*, TCS Technical Architects Conference, 2014.

[19] M. Carletti et al., *A deep learning approach for anomaly detection with industrial time series data: a refrigerators manufacturing case study*, 2020.

[20] G. Koch et al., *Siamese Neural Networks for One-shot Image Recognition, Proceedings of the 32nd International Conference on Machine Learning*, 2015.

[21] Y. LeCun et al., *Gradient-Based Learning Applied to Document Recognition*, 1998.

[22] D. G. Lowe, *Object Recognition from Local Scale-Invariant Features*, Proceedings of the International Conference on Computer Vision, 1999.

[23] M. Jones et al., *Robust Real-time Face Detection*, Proceedings of the Eighth IEEE International Conference on Computer Vision, 2001.

[24] Z. Zhao et al., *Object Detection with Deep Learning: A Review*, 2019.

[25] S. Balaban, *Deep learning and face recognition: the state of the art*, 2015.

[26] A. Krizhevsky et al., *ImageNet Classification with Deep Convolutional Neural Networks*, Neural Information Processing Systems, 2012.

[27] A. Levinshtein et al., *Real-time deep hair matting on mobile devices*, 2018.

[28] Y. Xiang et al., *Learning to Track: Online Multi-Object Tracking by Decision Making*, 2015.

[29] J. Wu et al., *Quantized Convolutional Neural Networks for Mobile Devices*, CVPR, 2016.

[30] A. Krizhevsky, *Learning Multiple Layers of Features from Tiny Images*, 2009.

[31] K. Simonyan et al., *Very Deep Convolutional Networks for LargeScale Image Recognition*, 2014.

[32] C. Szegedy et al., *Going Deeper with Convolutions*, 2014.

[33] K. He et al., *Deep Residual Learning for Image Recognition*, 2015.

[34] M. A. Anusuya et al., *Speech Recognition by Machine: A Review, International Journal of Computer Science and Information Security*, Vol. 6, No. 3, 2009.

[35] I. Sutskever et al., *Sequence to Sequence Learning with Neural Networks*, NIPS, 2014.

[36] W. Yih et al., *Multi-Document Summarization by Maximizing Informative Content-Words*, Proceedings of the 20th International Joint Conference on Artificial Intelligence, 2007.

[37] S. Hayou et al., *On the Impact of the Activation Function on Deep Neural Networks Training*, 2019.

[38] S. Siegelmann et al., *Turing computability with neural networks*, 1991.

[39] C. Özkan et al., *The Comparison of Activation Functions for Multispectral Landsat TM Image Classification*, 2003.

[40] X. Glorot et al., *Deep Sparse Rectifier Neural Networks*, 2015.

[41] M. VREJOIU, *Reţele neuronale convoluţionale, Big Data şi Deep Learning în analiza automată de imagini*, Romanian Journal of Information Technology and Automatic Control, Vol. 29, No. 1, 91-114, 2019.

[42] A. Karpathy et al., *Convolutional Neural Networks for Visual Recognition*, Stanford CS Class (CS231n): http://cs231n.github.io/convolutional-networks, 2015.

[43] F. Rosenblatt, *The perceptron: A probabilistic model for information storage and organization in the brain*, Psychological Review 65: pp. 386-408, 1958.

[44] W. McCulloch et al., *A logical calculus of the ideas immanent in nervous activity*, Bulletin of Mathematical Biophysics, Vol. 5(4), pp. 115-133, 1943.

[45] S. Skansi, *Introduction to Deep Learning From Logical Calculus to Artificial Intelligence*, 2018.

[46] Y. LeCun et al., *Deep learning*, Nature, 521, pp. 436–444, 2015.

[47] C. M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, New York, 1995.

[48] Debes et al., *Transfer Functions in Artificial Neural Networks - A Simulation-Based Tutorial*, Germany, 2005.

[49] Iulian B. Ciocoiu, *Hybrid Feedforward Neural Networks for Solving Classification Problems*, Neural Processing Letters 16: pp. 81–91, 2002.

[50] A. R. Barron, *Statistical properties of artificial neural networks*, Proceedings of Conference on Decision and Control, 1989.

[51] G. Cybenko, *Approximations by superpositions of sigmoidal functions*, Mathematics of Control, Signals, and Systems, 1989.

[52] D. W. Ruck et al., *Feature Selection Using a Multilayer Perceptron*, 1993.

[53] P. J. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, PhD thesis, Harvard University, 1974.

[54] D. E. Rumelhart et al., *Back propagation: A degenerate Kalman filter*, IEEE Transactions on Pattern Analysis and Machine Intelligence , 1989.

[55] D. B. Parker, *Learning-logic*, Invention Report 581-64, Stanford University, Stanford, CA, 1982.

[56] S, Singhal, *Training multilayer perceptrons with the extended Kalman algorithm*, Advances in Neural Information Processing Systems 1, pp. 133–140, 1989.

[57] H. Bourlard et al., *Auto-association by multilayer perceptrons and singular value decomposition*, Biological Cybernetics 59, pp. 291-294, 1988.

[58] C. C. Aggarwal, *Neural Networks and Deep Learning*, Springer, 2018.

[59] A. Zhang et al., *Dive into Deep Leaning*, 2020.

[60] M. Sopena, et al., *Neural networks with periodic and monotonic activation functions: a comparative study in classification problems*, Artificial Neural Networks Conference, 1999.

[61] B. C. Csáji, *Approximation with Artificial Neural Networks*, 2001.

[62] D. Wlodzislaw, et al., *Survey of Neural Transfer Functions*, pp. 163, Poland, 1999.

[63] X. Glorot, et al., *Understanding the difficulty of training deep feedforward neural networks*, 13th International Conference on Artificial Intelligence and Statistics, Volume 9, 2010.

[64] J. Turian et al., *Quadratic features and deep architectures for chunking*, The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, pp. 245–248, 2009.

[65] J. Han et al., *The influence of the sigmoid function parameters on the speed of backpropagation learning*, Natural to Artificial Neural Computation, Springer, pp. 195–201, 1995.

[66] R. M. Neal, *Connectionist learning of belief networks*, Artificial Intelligence, vol. 56, no. 1, pp. 71–113, 1992.

[67] S. Hochreiter et al., *Long short-term memory*, Neural Computation, 9: pp. 1735–1780, 1997.

[68] Cho K. et al., *Learning phrase representations using RNN encoder–decoder for statistical machine translation*, 2014 Conference on Empirical Methods in Natural Language Processing, pp. 1724–1734, 2014.

[69] K. Srinivasan et al., *An Efficient Implementation of Artificial Neural Networks with K-fold Cross-validation for Process Optimization*, 2019.

[70] A. V. Olgac et al., *Performance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks*, International Journal of Artificial Intelligence And Expert Systems, Vol. 1: Issue (4), Turkey, 2011.

[71] A. Maas et al., *Rectifier Nonlinearities Improve Neural Network Acoustic Models*, 2013.

[72] R. Collobert et al., *Natural Language Processing (Almost) from Scratch*, 2011.

[73] V. Nair et al., *Rectified linear units improve restricted boltzmann machines*, 2010.

[74] M. D. Zeiler, *On rectified linear units for speech processing*, 2013.

[75] G. E. Dahl et al., *Improving deep neural networks for LVCSR using rectified linear units and dropout*, 2013.

[76] T. Poggio et al., *Theoretical Issues in Deep Networks: Approximation, Optimization and Generalization*, United States, 2019.

[77] S. Srinivas et al., *Full-Jacobian Representation of Neural Networks*, Town of Martigny, the State of Valais, 2019.

[78] Y. Bengio et al., *Sharp Minima Can Generalize For Deep Nets*, Proceedings of the 34 th International Conference on Machine Learning, 2017.

[79] G. Koch et al., *Siamese neural networks*, ICML Deep Learning Workshop, volume 2, 2015.

[80] S. Scardapane et al., *Kafnets: kernelbased non-parametric activation functions for neural networks*, 2017.

[81] S. Jadon et al., *Improving Siamese Networks for One Shot Learning using Kernel Based Activation functions*, 2019.

[82] Y. LeCun, *The mnist database of handwritten digits*, 1998.

[83] AT&T Laboratories Cambridge, *The database of faces*.

[84] B. M Lake et al., Humanlevel concept learning through probabilistic program induction, Science, 350(6266): pp. 1332–1338, 2015.

[85] Y. Tian, *OVER-PARAMETERIZATION AS A CATALYST FOR BETTER GENERALIZATION OF DEEP RELU NETWORK*, 2019.

[86] K. He et al., *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*, 2015.

[87] D. A. Clevert et al., *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs),* 2015.

[88] G. Klambauer et al., *Self-Normalizing Neural Networks*, 2017.

[89] D. Hendrycks et al., *BRIDGING NONLINEARITIES AND STOCHASTIC REGULARIZERS WITH GAUSSIAN ERROR LINEAR UNITS*, ICLR , 2017.

[90] B. Zoph et al., *Learning transferable architectures for scalable image recognition*, 2017.

[91] C. Szegedy et al., *Inception-v4, inception-resnet and the impact of residual connections on learning*, AAAI, pp. 4278–4284, 2017.

[92] I. Bello et al., *Neural optimizer search with reinforcement learning*, 2017.

[93] E. Alcaide, *E-swish: Adjusting Activations to Different Network Depths*, 2018.

[94] M. Basirat, *The Quest for the Golden Activation Function*, 2018.

[95] G.J. Gordon, *Stable function approximation in dynamic programming*, 1995.

[96] H. H. Chieng et al., *Flatten-T Swish: a thresholded ReLU-Swish-like activation function for deep learning*, International Journal of Advances in Intelligent Informatics, Vol. 4, No. 2, pp. 76-86, 2018.

[97] C. Dugas et al., *Incorporating Second-Order Functional Knowledge for Better Option Pricing*, 2001.

[98] H. Zheng et al., *Improving Deep Neural Networks Using Softplus Units*, 2015.

[99] D. Misra, *Mish: A Self Regularized Non-Monotonic Neural Activation Function*, 2019.

[100] N. Patwardhan et al., *ARiA: Utilizing Richard's Curve for Controlling the Non-monotonicity of the Activation Function in Deep Neural Nets*, 2018.

[101] L. P. Littman et al., *Reinforcement Learning: A Survey, 1996*.

[102] S. Elfwing et al., *Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning*, 2017.

[103] I. Szita et al., *SZ-Tetris as a benchmark for studying key problems of reinforcement learning*, ICML, 2010.

[104] I. Sahin et al. , *Design and Implementation of Neural Networks Neurons with RadBas, LogSig, and TanSig Activation Functions on FPGA*, 2012.

[105] S. Himavathi et al., *Feedforward Neural Network Implementation in FPGA Using Layer Multiplexing for Effective Resource Utilization*, 2007.

[106] D.L. Elliott at al., *A better Activation Function for Artificial Neural Networks*, 1993.

[107] A. Wuraola et al., *SQNL: A New Computationally Efficient Activation Function*, 2018.

[108] B. Carlile et al., *IMPROVING DEEP LEARNING BY INVERSE SQUARE ROOT LINEAR UNITS (ISRLUs),* 2017.

[109] M. D. Klimek et al., *Neural Network-Based Approach to Phase Space Integration*, 2018.

[110] X. Jin et al., *Deep Learning with S-shaped Rectified Linear Activation Units*, 2015.

[111] L. H. Eidnes at al., *SHIFTING MEAN ACTIVATION TOWARDS ZERO WITH BIPOLAR ACTIVATION FUNCTIONS*, 2018.

[112] D. Mishkin et al., *All you need is a good init*, 2015.

[113] W. Shang, et al., *Understanding and improving convolutional neural networks via concatenated rectified linear units*, 2016.

[114] D. Balduzzi, et al. *The shattered gradients problem: If resnets are the answer, then what is the question?,* 2017.

[115] M. Mirza et al., *Maxout networks,* 2013.

[116] Y. Netzer et al., *Reading digits in natural images with unsupervised feature learning*, 2011.

[117] G. E. Hinton et al., *Improving neural networks by preventing co-adaptation of feature detectors*, 2012.

[118] R. M. Neal, *Bayesian Learning for Neural Networks*, 1996.

[119] A. N. Chernodub et al., *Norm-preserving orthogonal permutation linear unit activation functions (OPLU),* 2016.

[120] F. Agostinelli et al., *LEARNING ACTIVATION FUNCTIONS TO IMPROVE DEEP NEURAL NETWORKS*, 2015.

[121] X. Yao, *Evolving Artificial Neural Networks*, 1999.

[122] A. Turner et al., *Neuroevolution: Evolving heterogeneous artificial neural networks*, 2014.

[123] M. Lin, *Network in network*, 2013.

[124] I. Goodfellow et al., *Softmax Units for Multinoulli Output Distributions*, 2016.

[125] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

[126] W. Liu et al., *Large-Margin Softmax Loss for Convolutional Neural Networks*, 2017.

[127] B. Chen et al., *Noisy Softmax: Improving the Generalization Ability of DCNN via Postponing the Early Softmax Saturation*, CVPR, 2017.

[128] G. B. Huang et al., *Labeled faces in the wild: A database forstudying face recognition in unconstrained environments*, 2008.

[129] N. Zhang et al., *Fine-grained lfw database*, 2016.

[130] L. Wolf et al., *Face recognition in unconstrained videos with matched background similarity*, 2011.

[131] A. F. T. Martins at al., *From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification*, 2016.

[132] H. B. Lee et al., *DropMax: Adaptive Variational Softmax*, 2018.

[133] J. Bergstra et al., *Quadratic polynomials learn better image features*, Technical Report 1337, 2009.

[134] W. Huaiqin, *Global stability analysis of a general class of discontinuous neural networks with linear growth activation functions*, 2009.

[135] S. Jan, *Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms*, Springer, 2005.

[136] D. Sussillo et al., *Random Walk Initialization for Training Very Deep Feedforward Networks*, 2014.

[137] S. Scardapane et al., *Kafnets: kernel-based non-parametric activation functions for neural networks*, 2017.

[138] M. Goyal et al., *Learning Activation Functions: A new paradigm for understanding Neural Networks*, 2019.

[139] D. Harrison et al., *Hedonic prices and the demand for clean air*, 1978.

[140] Lang and Witbrock, *Two spiral data set*, 1988.

[141] M. S. Gashler et al., *Training Deep Fourier Neural Networks To Fit Time-Series Data*, 2014.

[142] L. Trottier et al., *Parametric exponential linear unit for deep convolutional neural networks*, 2017.

[143] L. B. Godfrey, *An Evaluation of Parametric Activation Functions for Deep Learning*, 2019.

[144] Y. Wang et al., *The Influence of the Activation Function in a Convolution Neural Network Model of Facial Expression Recognition*, 2020.

[145] M.J. Lyons et al., *The Japanese female facial expression (JAFFE),* In Proceedings of the Third International Conference on Automatic Face and Gesture Recognition, pp. 14–16, 1998.

[146] P.L. Carrier et al., *FER-2013 Face Database*, Technical report, 2013.

[147] S.Balaji et al., *Learn-able parameter guided Activation Functions*, 2019.

[148] lessw2020, *Threshold Relu*, https://github.com/lessw2020/TRelu, Last accessed 28 July 2020.

[149] S. Zagoruyko et al., *Wide Residual Networks*, 2016.

[150] M. Courbariaux et al., *BinaryConnect: Training Deep Neural Networks with binary weights during propagations*, 2015.

[151] A. Bismark, *Simulation and Characterization of a LXeTPC for DARWIN R&D*, Master thesis, 2019.

[152] S. Qiu et al., *FReLU: Flexible Rectified Linear Units for Improving Convolutional Neural Networks*, 2017.

[153] *Neural Networks Fail to Learn Periodic Functions and How to Fix It*, 2020.

[154] R. Rojas, *Neural Networks*, Berlin, Springer-Verlag, 1996.

[155] J. Roman et al., *Backpropagation and Recurrent Neural Networks in Financial Analysis of Multiple Stock Market Returns*, 1996.

[156] P. Koehn, *Combining Genetic Algorithms and Neural Networks: The Encoding Problem*, A Master Thesis, 1994.

[157] V. N. Manohar et al., *Function approximation using back propagation algorithm in artificial neural networks*, 2007.

[158] A. Howard et al., *Mobilenets: Efficient convolutional neural networks for mobile vision applications*, 2017.

[159] F. Chollet, *Xception: Deep Learning with Depthwise Separable Convolutions*, 2016.

[160] C. Lemaréchal et al., *Cauchy and the Gradient Method(PDF),* Doc Math Extra: pp. 251–254., 2012.

[161] A. Galkin, *What is the difference between test set and validation set?,* 2011.

[162] D. M. Allen, *The Relationship between Variable Selection and Data Agumentation and a Method for Prediction*, 1974.

[163] M. Stone, *Cross-Validatory Choice and Assessment of Statistical Predictions*, 1974.

[164] M. Stone, *An Asymptotic Equivalence of Choice of Model by Cross-Validation and Akaike's Criterion*, 1977.

[165] G. C. Cawley et al., *On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation*, 2010.

[166] Q. Ren et al., *Tectonic discrimination of olivine in basalt using data mining techniques based on major elements: a comparative study from multiple perspectives*, 2019.

[167] P. Bühlmann et al., *Statistics for High-Dimensional Data*, 2011.

[168] Y. Jiang*, Variable selection with prior information for generalized linear models via the prior lasso method*, Journal of the American Statistical Association. 111: pp. 355–376, 2016.

[169] A.N. Tikhonov, *On the stability of inverse problems*, 1943.

[170] D.H. Ackley et al., *A learning algorithm for boltzmann machines*, 1985.

[171] P. Kennedy, *A Guide to Econometrics*, 2003.

[172] M. Gruber, *Improving Efficiency by Shrinkage: The James–Stein and Ridge Regression Estimators*, 1998.

[173] L. Prechelt et al., *Early Stopping — But When?,* In Grégoire Montavon; Klaus-Robert Müller (eds.). Neural Networks, pp. 53–67, 2012.

[174] Y. Yao et al., *On Early Stopping in Gradient Descent Learning*, 2007.

[175] G. E. Hinton, *System and method for addressing overfitting in a neural network*, 2016.

[176] G.E. Hinton et al., *Improving neural networks by preventing co-adaptation of feature detectors*, 2012.

[177] G.E. Dahl et al., *Improving deep neural networks for LVCSR using rectified linear units and dropout*, 2013.

[178] N. Srivastava et al., *Dropout: a simple way to prevent neural networks from overfitting*, 2014.

[179] J. Hertz et al., *Introduction to the Theory of Neural Computation*, 1991.

[180] D. Warde-Farley et al., *An empirical analysis of dropout in piecewise linear networks*, 2013.

[181] C. C. Aggarwal, *Neural Networks and Deep Learning*, Springer, 2018.

[182] L. Wan et al., *Regularization of neural networks using dropconnect*, ICML, 2013.

[183] Y. LeCun et al., *Gradient-based learning applied to document recognition*, 1998.

[184] Md Zahangir Alom et al., *State-of-the-Art Survey on Deep Learning Theory*, 2019.

[185] Yan Chen et al., *Network in network based weakly supervised learning for visual tracking*, 2015.

[186] M. Sandler et al., *MobileNetV2: Inverted Residuals and Linear Bottlenecks*, 2018.

[187] A. Howard et al., *Searching for MobileNetV3*, 2019.

[188] M. Thoma, *A Survey of Semantic Segmentation*, 2016.

[189] *2018 Data Science Bowl*, https://www.kaggle.com/c/data-science-bowl-2018/data, 2018.

[190] A. Turing, *Computing Machinery and Intelligence*, 1950.

[191] O. Levy et al., *Linguistic Regularities in Sparse and Explicit Word Representations*, 2014.

[192] *Reuters Text Categorization Collection Data Set*, https://archive.ics.uci.edu/ml/datasets/reuters-21578+text+categorization+collection, 1987.

[193] A. L. Maas et al., *Learning Word Vectors for Sentiment Analysis*, The 49th Annual Meeting of the Association for Computational Linguistics, 2011.

[194] T. Mikolov, *Advances in Pre-Training Distributed Word Representations*, International Conference on Language Resources and Evaluation, 2018.

[195] R. Collobert et al., *A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning*, ICML, 2008.

[196] R. J. Williams et al., *Learning representations by back-propagating errors*, 1986.

[197] F. A. Gers et al., *Learning to forget: Continual prediction with lstm*, 1999.

[198] G. Lample et al., *Neural architectures for named entity recognition*, 2016.

[199] Y. Bengio et al., *Representation Learning: A Review and New Perspectives*, 2013.

[200] J. Schmidhuber, *Deep Learning in Neural Networks: An Overview*, 2015.

[201] F. Gers et al., *Learning to Forget: Continual Prediction with LSTM*, 2000.

[202] D. Pedamonti, *Comparison of non-linear activation functions for deep neural networks on MNIST classification task*, 2018.

[203] J. Kilian et al., *The Dynamic Universality of Sigmoidal Neural Networks*, 1996.

[204] J. Hu et al., *Effects of BP Algorithm-based Activation Functions on Neural Network Convergence*, 2018.

[205] V. Tiwari et al., *Hardware implementation of neural network with Sigmoidal activation functions using CORDIC*, 2015.

[206] Z.J. Jia et al., *Tracking control of nonaffine systems using bio-inspired networks with autotuning activation functions and self-growing neurons*, 2017.

[207] A. F. Sheta et al., *A Comparison between Regression, Artificial Neural Networks and Support Vector Machines for Predicting Stock Market Index*, 2015.

[208] T. Oda et al., *A Neural Network Based User Identification for Tor Networks: Comparison Analysis of Different Activation Functions Using Friedman Test*, Network-Based Information Systems, 2016.

[209] M. Cilimkovic, *Neural Networks and Back Propagation Algorithm*, Ireland, 2015.

[210] K. Hara et al., *Comparison of activation functions in multilayer neural network for pattern classification*, 1994.

[211] H. T. Siegelmann et al., *On the computational power of neural networks*, 1995.

[212] T.Jayalakshmi et al., *Statistical Normalization and Back Propagation for Classification*, 2011.

[213] A. Gupts et al., *The Weight Decay backpropagation for generalizations with missing values*, 1998.

[214] Q. Liu et al., *A convolutional click prediction model*, 2015.

[215] Y. LeCun et al., *Efficient BackProp*, Springer, 1998.

[216] Xinyu Liu et al., *TanhExp: A Smooth Activation Function with High Convergence Speed for Lightweight Neural Networks*, 2020.

[217] L. Shin, *Dying ReLU and Initialization: Theory and Numerical Examples*, 2019.

[218] K. Simonyan et al., *Very deep convolutional networks for large-scale image recognition*, 2014.

[219] M. A. Mercioni et al., *The most used activation functions: classic versus current*, 15th International Conference on Development and Applications systems, 2020.

[220] H. Mhaskar et al., *How to Choose an Activation Function*, 1994.

[221] S. J. Pan et al., *A Survey on Transfer Learning*, 2010.

[222] X. Liu et al., *TanhExp: A Smooth Activation Function with High Convergence Speed for Lightweight Neural Networks*, 2020.

[223] S. Narayan, *The generalized sigmoid activation function: Competitive supervised learning*, 1997.

[224] J. C. Chen et al., *Comparing Activation Functions in Modeling Shoreline Variation Using Multilayer Perceptron Neural Network*, 2020.

[225] D. H. Ballard et al., *Computer Vision, Prentice Hall*, 1982.

[226] Huang, T. et al., *Computer Vision: Evolution And Promise*, CERN, 1996.

[227] A. Zimek et al., *Outlier Detection*, 2017.

[228] V. N. Dornadula et al., *Credit Card Fraud Detection using Machine Learning Algorithms*, 2019.

[229] D. Thornton et al., *Categorizing and Describing the Types of Fraud in Healthcare*, 2015.

[230] S. Minaee et al., *Deep Learning Based Text Classification: A Comprehensive Review*, 2020.

[231] M. A. Kramer, *Nonlinear principal component analysis using autoassociative neural networks*, 1991.

[232] D. P. Kingma et al., *Auto-Encoding Variational Bayes*, 2013.

[233] V. J. Hodge et al., *A Survey of Outlier Detection Methodologies*, 2004.

[234] A. Garcia et al., *A review on outlier/anomaly detection in time series data*, 2020.

[235] V. Chandola et al., *Outlier detection: A survey*, 2007.

[236] X. Xie et al., *Real-time illegal parking detection system based on deep learning*, 2017.

[237] T. Schlegl et al., *Unsupervised anomaly detection with generative adversarial networks to guide marker discovery*, 2017.

[238] A. Javaid et al., *A deep learning approach for network intrusion detection system*, 2016.

[239] M. Mohammadi et al., *Deep learning for iot big data and streaming analytics: A survey*, 2017.

[240] S. J. Russell et al., *Artificial Intelligence: A Modern Approach*, 2010.

[241] G. Hinton et al., *Unsupervised Learning: Foundations of Neural Computation*, 1999.

[242] F. V. Jensen, *Introduction to Bayesian Networks*, Springer-VerlagBerlin, 1996.

[243] Driver et al., *Quantitative Expression of Cultural Relationships*, 1932.

[244] M. L. Manevitz et al., *One-Class Svms for Document Classification*, 2002.

[245] K. Gurney, *An Introduction to Neural Networks*, 1997.

[246] B. Alejandro, *Building ai applications using deep learning*, URL https://blog.easysol. net/wp-content/uploads/2017/06/image1.png, 2016.

[247] R. Chalapathy et al., *DEEP LEARNING FOR ANOMALY DETECTION: A SURVEY*, 2019.

[248] H. K. Peng et al., *Multi-scale compositionality: identifying the compositional structures of social dynamics using deep learning*, 2015.

[249] JS. Dyer, *A time-sharing computer program for the solution of the multiple criteria problem*, 1973.

[250] D. J. Wallenius et al., *Multiple criteria decision making, multiattribute utility theory: The next ten years*, 1992.

[251] D. Ciresan et al., *Multi-column deep neural networks for image classification*, 2012.

[252] D. T Shipmon et al., *Time series anomaly detection; detection of anomalous drops with limited features and sparse examples in noisy highly periodic data*, 2017.

[253] K. Hundman et al., *Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding*, 2018.

[254] L. Zhu et al., *Deep and confident prediction for time series at uber*, 2017.

[255] P. Malhotra et al., *Long short term memory networks for anomaly detection in time series*, 2015.

[256] J. P. Assendorp, *Deep learning for anomaly detection in multivariate time series data*, PhD thesis, 2017.

[257] S. Ahmad et al., *Unsupervised real-time anomaly detection for streaming data*, 2017.

[258] *Understanding LSTM Networks*, http://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015.

[259] F. A. Gers, *Learning to forget: Continual prediction with LSTM*, 1999.

[260] A. Pulver et al., *LSTM with Working Memory*, 2017.

[261] F. Beaufays, *The neural networks behind Google Voice transcription*, 2015.

[262] A. Graves et al., *A Novel Connectionist System for Unconstrained Handwriting Recognition*, 2009.

[263] P. Khaitan, *Chat Smarter with Allo*, 2016.

[264] Y. Wu et al., *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*, 2016.

[265] T. Capes et al., *Siri On-Device Deep Learning-Guided Unit Selection Text-to-Speech System*, 2017.

[266] A. Voelker et al., *Legendre Memory Units: Continuous-Time Representation in Recurrent Neural Networks*, 2019.

[267] M. Jain, *A New Hyperbolic Tangent Based Activation Function for Neural Networks*, 2018.

[268] National Institute of Diabetes and Digestive and, *Pima Indians Diabetes Database*, 1990.

[269] R. A. Fisher, *The use of multiple measurements in taxonomic problems*, 1936.

[270] H. Hofmann, *German Credit data*, Institut f"ur Statistik und "Okonometrie Universit"at Hamburg FB Wirtschaftswissenschaften Von-Melle-Park 5.

[271] R. Forsyth, *Zoo database*, 1990.

[272] G. H. Lincoff, Alfred A. Knopf, *Mushroom Database*, 1981.

[273] D. Sterling et al., *Annealing Data*, 2009.

[274] R. S. Siegler et al., *Balance-Scale Dataset*, 1976.

[275] *Glass Identification Dataset*, B. German Central Research Establishment.

[276] R. P. Gorman et al., *Analysis of Hidden Units in a Layered Network Trained to Classify Sonar Targets*, 1988.

[277] *Automobile Dataset*, Data From 1985 Ward's Automotive Yearbook.

[278] *Letter Recognition Dataset*, UCI machine learning repository.

[279] H. Xiao et al., *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*, 2017.

[280] P. K. Diederik et al., *Adam: A Method for Stochastic Optimization*, 2014.

[281] S. Ioffe et al., *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, 2015.

[282] S. Zachow et al., *3D reconstruction of individual anatomy from medical image data: Segmentation and geometry processing*, 2007.

[283] Y. Nesterov, *A method for unconstrained convex minimization problem with the rate of convergence o(1/k2),* 1983.

[284] *Fully Connected Convolutional Network (U-Net) architecture,* https://github.com/Paulymorphous/nucleus-segmentation.

[285] H. Rezatofighi et al., *Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression*, 2019.

[286] L. Perez et al., *The Effectiveness of Data Augmentation in Image Classification using Deep Learning*, 2017.

[287] *Numenta Anomaly Benchmark*, https://www.kaggle.com/boltzmannbrain/nab.

[288] V. Pascal et al., *Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising*, 2010.

[289] Pavithrasv, *Anomaly detection in time series*, https://github.com/pavithrasv. .

[290] *LSTM architecture*, https://www.curiousily.com/posts/anomaly-detection-in-time-series-with-lstms-using-keras-in-python/#lstm-autoencoder-in-keras, Online link accessed: 30.08.2020.

[291] *S&P 500 Index Data*, https://www.kaggle.com/pdquant/sp500-daily-19862018.

[292] *Anomaly detection in time series*, https://github.com/pavithrasv.

[293] X. Liang et al., *Assessing Beijing's PM2.5 pollution: severity, weather impact, APEC and winter heating*, Proceedings of the Royal Society A, 471, 2015.

[294] A. Shenouda et al., *A Quantitative Comparison of Different MLP Activation*, J. Wang et al., pp. 849 – 857, 2006.

[295] M. Cococcioni et al., *Fast Approximations of Activation Functions in Deep Neural Networks when using Posit Arithmetic*, 2020.

[296] G. Bingham et al., *Evolutionary Optimization of Deep Learning*, 2020.

[297] W. McCulloch et al., *A Logical Calculus of Ideas Immanent in Nervous Activity*, 1943.

[298] Y. Goldberg, *A Primer on Neural Network Models for Natural Language Processing*, 2016.

[299] *Dogs and Cats dataset*, https://www.kaggle.com/c/dogs-vs-cats.

[300] M. Mercioni et al., *Dynamic modification of Activation Function using the Backpropagation Algorithm in the Artificial Neural Networks*, 2019.

[301] C. Chen et al., *A Feedforward Neural Network with Function Shape Autotuning*, 1996.